



DGTL

Эволюция Data Quality

Опыт перехода от локальных
проверок к платформенному
решению и автоматизации





Всем привет!
Я — Алина Ивановичева,
QA-инженер в финансовом домене
Райффайзен Банка



Опыт

В тестировании более 6 лет
(front/back/mobile/data)



Мои достижения

Развиваюсь в T-shape направлениях
Lead QA Community



До Райффайзен Банка

Ex-Deutsche Bank Technology Center, VK



С чем работаем



Десятки тысяч событий в секунду

Обрабатываем пиковые нагрузки

Сотни терабайт

Для исторических данных в проде

Ежедневно

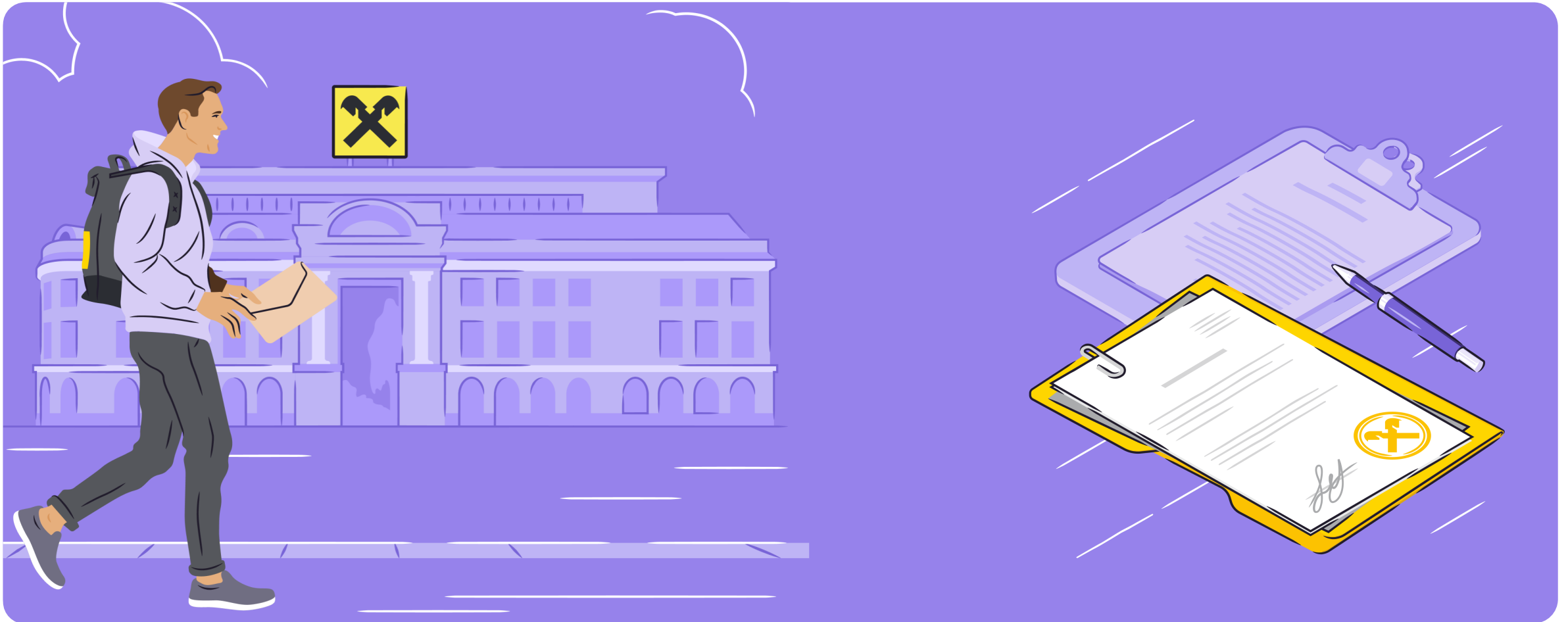
Обрабатываем данные, которые из десятков источников, раскладываются в 2 000+ таблиц и сотни тысяч сделок



Дериватив — финансовый инструмент



Стоимость или выплаты по которому зависят от базового актива, курса, ставки или другого показателя



Что в результате?



Клиент

Счастлив, что через месяц сможет получить сумму по фиксированному курсу



Банк

Заработал деньги

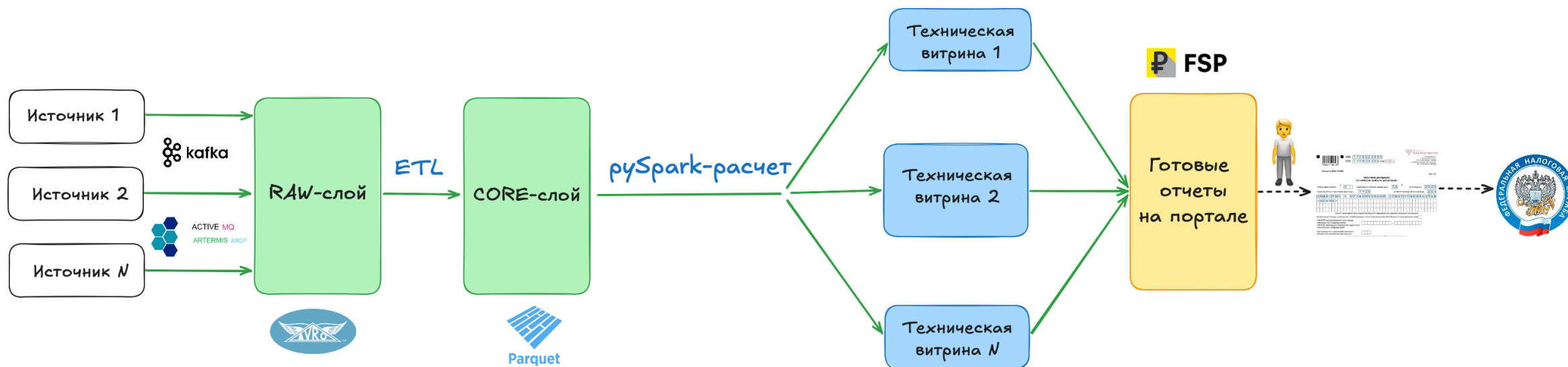


ФНС

напоминает банку, что не стоит забывать об уплате налогов с полученной прибыли 😊



Схема процесса подготовки финансовой отчетности



ИТОГОВЫЙ стек



Python 3.11

Бизнес-логика и модульная структура расчётов

PySpark 3.5

Распределённые batch-расчёты на Spark

MinIO / S3

Хранение parquet-датасетов и промежуточных витрин

Airflow

Оркестрация расписаний и зависимостей между пайплайнами

FSP-платформа

Запуск и управление расчётами

Data Lake / DWH

Источники банковских данных: сделки, бумаги, клиенты, курсы, справочники

Artifactory + Poetry

Управление зависимостями

Pytest

Тестирование расчётных сценариев

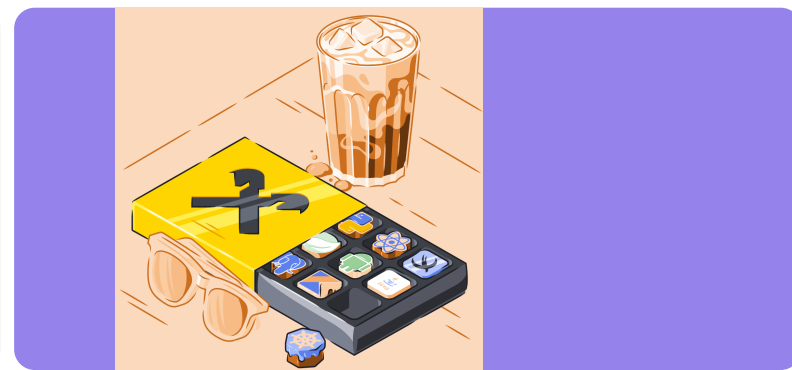
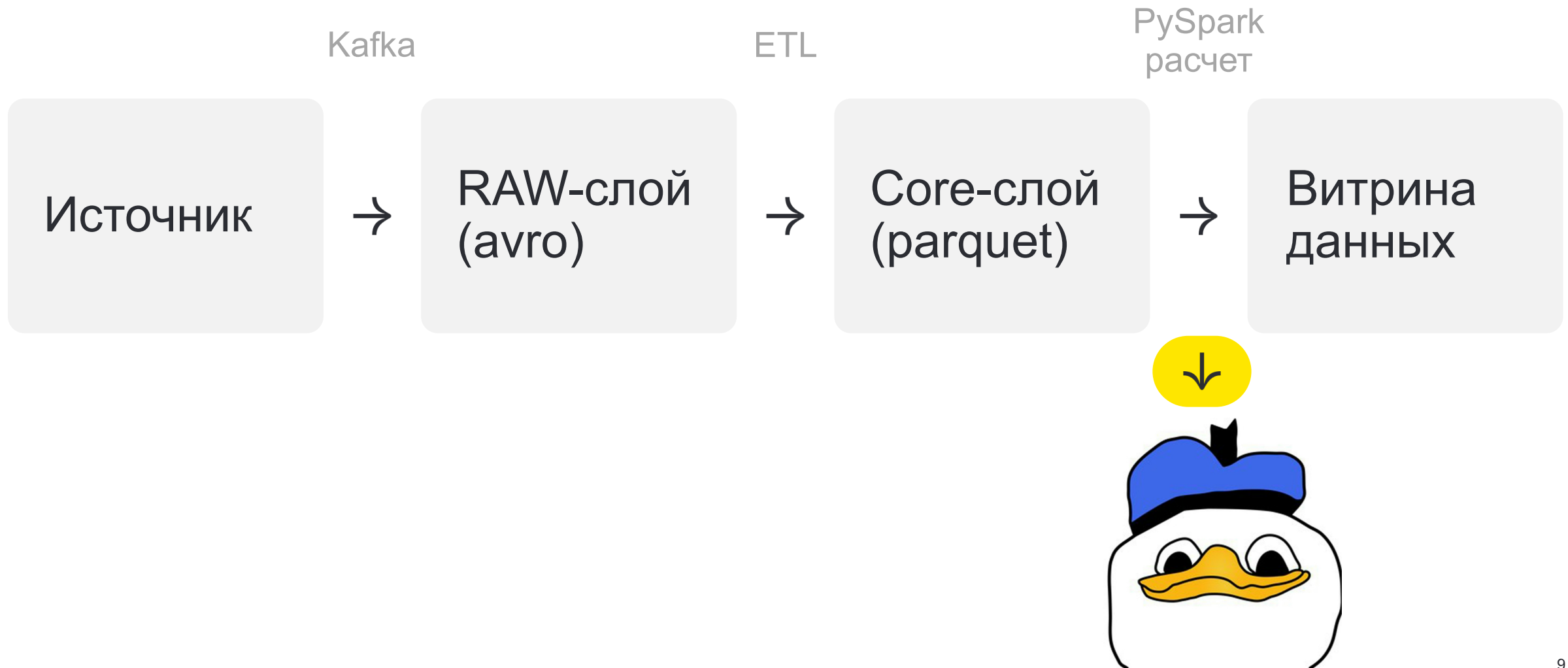


Схема процесса подготовки финансовой отчетности



Критерии качества данных



С чего мы начинали

Manual Exploratory Testing

2-3 недели авральной
сверки регистра перед
декларацией





DQ стало критичным и мы поняли, что дальше так продолжаться не может

Сроки сдачи горят

Бесконечные ручные сверки, аналитики и тестировщики забыли что такое сон

Ошибки данных в банке грозят огромными штрафами

Уровень ответственности за отчеты максимально высокий

Проверки не переиспользуются

Результаты не накапливаются

Аномалии всплывают на самом последнем этапе

Дорого искать первопричину и исправлять

Не хватает

Системного подхода и прозрачности результатов

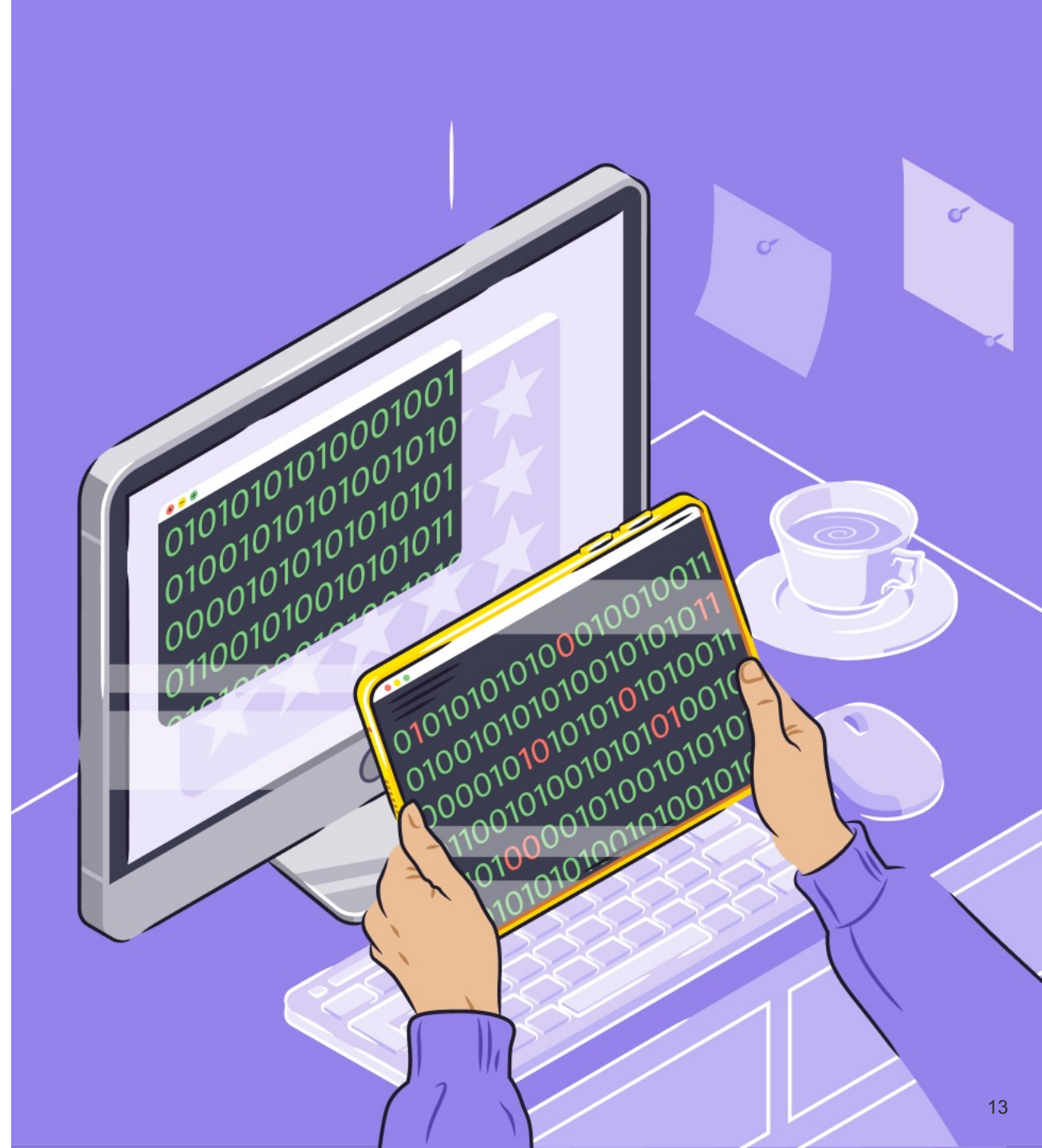


Первое решение — локальные проверки в коде

Manual Exploratory Testing



PyDeequ Checks



Почему выбрали Фреймворк PyDeequ



Простота и скорость
интеграции в наш текущий
PySpark-процесс

Проверки дата-контракта можно
встроить в существующую job
и настроить алертинг

~ 2 недели



Стоило внедрение
по ресурсам

Как выглядит проверка в коде



transformation

```
def fx_forward_amount_rate_check(deals, currency_legs):  
    return (  
        deals  
        .join(currency_legs, on="DEAL_ID")  
        .groupBy("DEAL_ID", "DEAL_RATE")  
        .agg(  
            F.max(F.when(F.col("CURRENCY_ID") == "EUR", F.col("AMOUNT"))).alias("EUR_AMOUNT"),  
            F.max(F.when(F.col("CURRENCY_ID") == "RUB", F.col("AMOUNT"))).alias("RUB_AMOUNT"),  
        )  
        .withColumn("EXPECTED_RUB_AMOUNT", F.col("EUR_AMOUNT") * F.col("DEAL_RATE"))  
        .withColumn("DIFF", F.abs(F.col("RUB_AMOUNT") - F.col("EXPECTED_RUB_AMOUNT")))  
    )
```

EUR_AMOUNT = 5 000
DEAL_RATE = 95

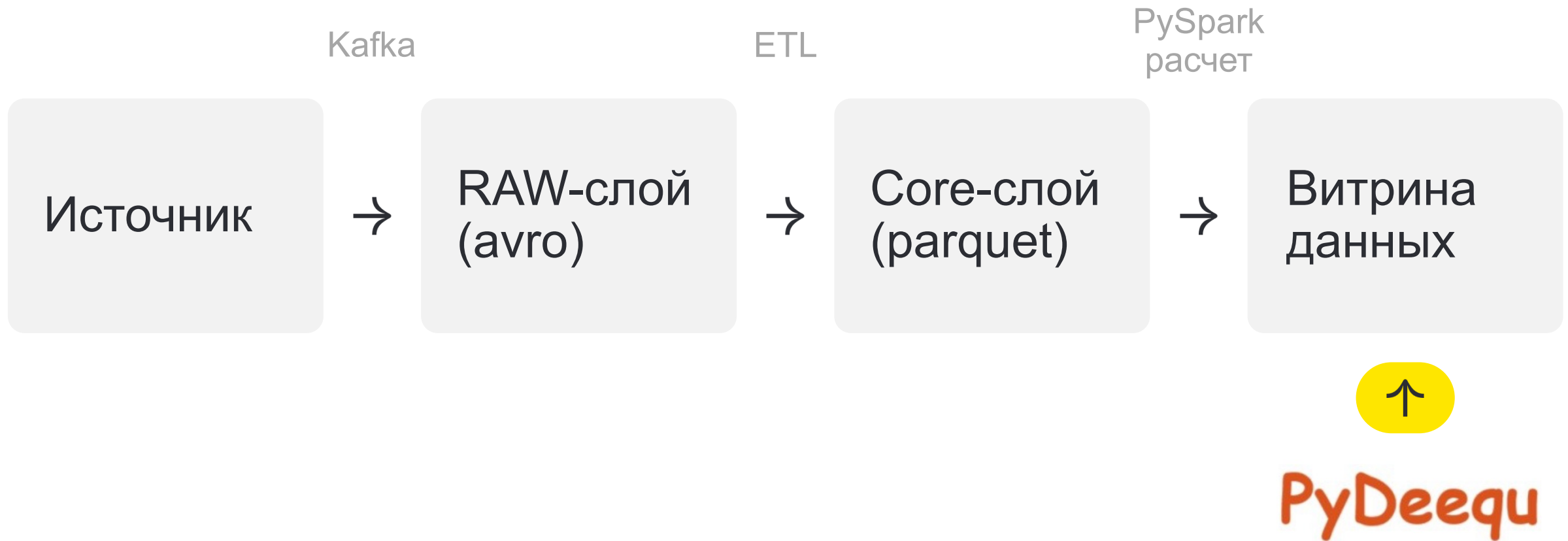
EXPECTED_RUB_AMOUNT
= 475 000

RUB_AMOUNT = 200 000
→ проверка падает

check

```
Check(spark, CheckLevel.Error, "FX_FORWARD_AMOUNT_RATE_CHECK").satisfies(  
    "DIFF <= 0.01",  
    "AmountMatchesRate",  
    hint="RUB amount does not match EUR amount multiplied by deal rate",  
)
```

Наше первое решение – локальные проверки в коде



Как выглядел готовый DQ-report в первой итерации



```
24/10/25 10:57:04 INFO data_quality.report_producer: Data Quality Check for: 3#DERIVATIVE_IR_DEAL_COUPON_DTL  
Check Level: Error | Status: Error
```

Constraints:

```
Constraint: ComplianceConstraint(Compliance(MigratedDealConstraint,MIN_COUPON_COUNT = MAX_COUPON_COUNT,None,List()),None))
```

```
Status: Failure
```

```
Message: Value: 0.96875 does not meet the constraint requirement! Schedule has changed for migrated deal!
```

```
--- End of Check ---
```



С какими проблемами столкнулись



(Проблема)

Получение сэмплов невалидных данных

Из коробки PyDeeqi не давал нам удобного способа достать примеры строк, которые не прошли проверки

(Решение)

Сделали кастомную обертку и доработку вокруг проверок

Чтобы отдельно вытаскивать такие сэмплы и использовать их для разбора инцидентов



Что делает кастомная обертка



Регистрирует проверки

- `add_check`: один источник
- `add_group_by_check`: агрегированный датафрейм
- `add_cross_check`: проверка после `join` двух источников

Готовит данные

- берет `DataFrame` из `Context`
- применяет `transformation-функцию`
- добавляет поля вроде `MATCHING_EXISTS`

Собирает результат

- `check_id`, `source`, `severity`
- `hint` для бизнес-смысла
- `samples` невалидных строк

Сэмплы выводятся в лог запуска расчета витрины



```
INFO:tax_report_lib.data_quality.report_producer:Data Quality Check for: 9#DERIVATIVE_IR_DEAL_COUPON_DTL  
Check Level: Error | Status: Error
```

Constraints:

```
Constraint: CompletenessConstraint(Completeness(MATCHING_EXISTS,None,None))
```

```
Status: Failure
```

```
Message: Value: 0.7852822580645161 does not meet the constraint requirement! Coupon doesn't have pnl details!
```

Sample:

```
--RECORD 0-----  
DEAL_ID      | 20*12*7080121  
COUPON_CODE   | leg1.coupon17  
MATCHING_EXISTS | NULL  
--RECORD 1-----  
DEAL_ID      | 20*12*7080121  
COUPON_CODE   | leg1.coupon18  
MATCHING_EXISTS | NULL  
--RECORD 2-----  
DEAL_ID      | 20*12*7080121  
COUPON_CODE   | leg1.coupon19  
MATCHING_EXISTS | NULL  
--RECORD 3-----  
DEAL_ID      | 20*12*7080121  
COUPON_CODE   | leg1.coupon20  
MATCHING_EXISTS | NULL  
--RECORD 4-----  
DEAL_ID      | 20*12*7080121  
COUPON_CODE   | leg2.coupon5  
MATCHING_EXISTS | NULL
```

Ограничение локального подхода



Правила понятны разработчику, но не всегда аналитику

Нет единого каталога проверок

Сложно масштабировать между командами

Результаты сложнее стандартизировать



Платформенное решение — полноценный DQ-сервис

Manual Exploratory Testing



PyDeequ Cheeks



SODA Cheeks



Почему выбрали именно SODA Core



— это open-source проект для проверки качества данных. Под капотом — собственный язык проверок SodaCL (Soda Check Language)

Разворачивать
инфраструктуру было
значительно дороже
по ресурсам

Чем прикрутить локальные
проверки

Низкий порог входа



Проверки на yaml

Сравнивали Soda с популярными и не очень платформами



 **elementary**

 **MONTE CARLO**



Оптимальный компромисс

SODA

Простой для быстрого внедрения

Гибкий для SQL-based проверок

Операционный для регулярного мониторинга в пайплайнах

Классная обзорная статья популярных на рынке инструментов, и цели для которых они лучше подходят

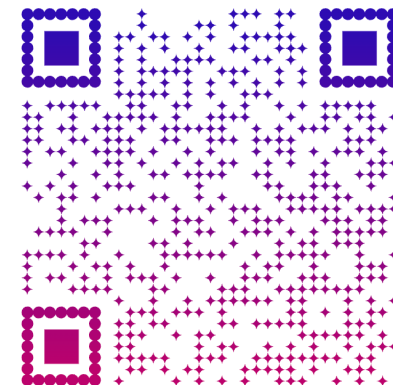
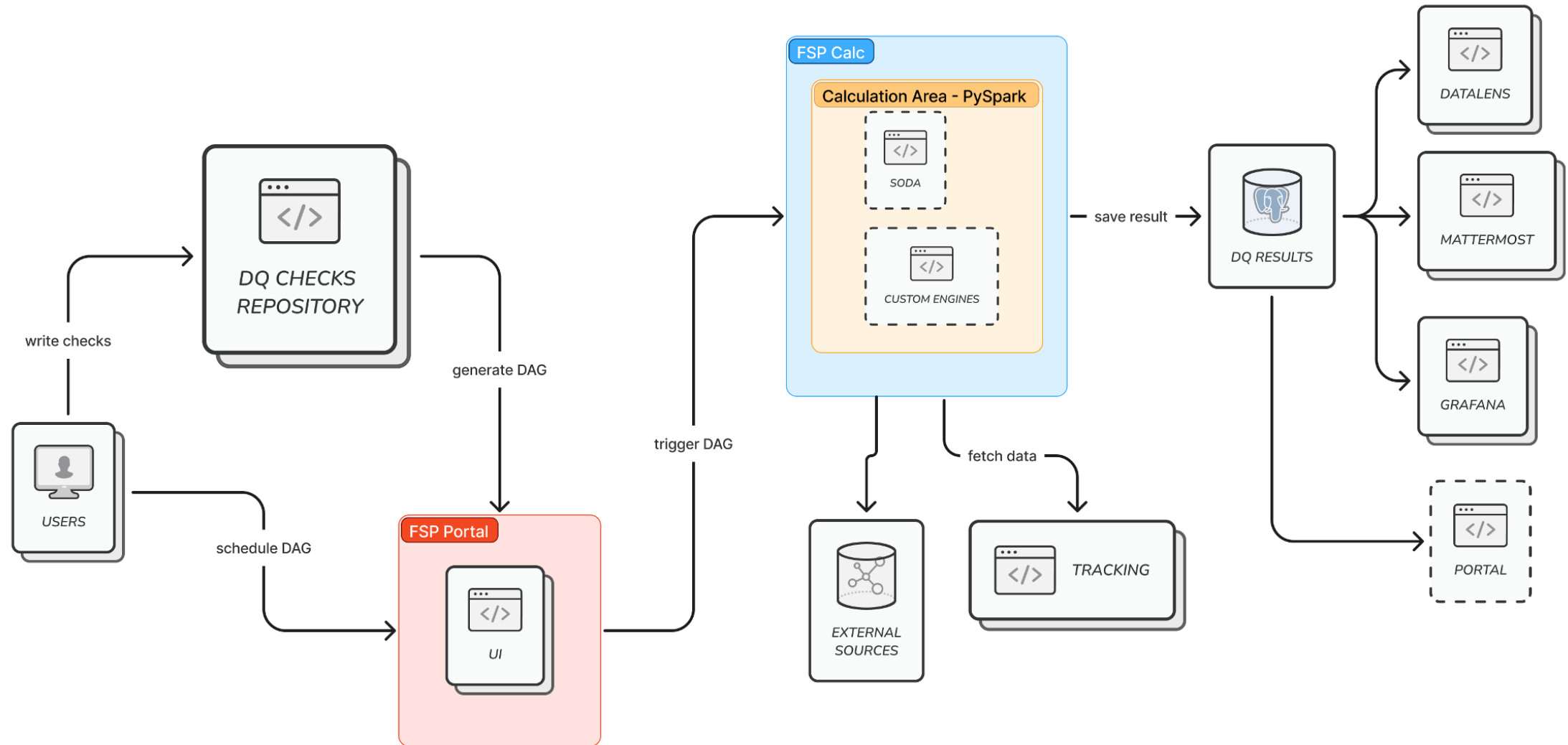


Схема платформенного DQ-сервиса



Как выглядит код проверки

- `missing_count(PARTY_ID) = 0:`
 `name: null_rows_in_required_field_PARTY_ID`
 `# description: Поле "Уникальный идентификатор контрагента" (PARTY_ID)`
 должен быть заполнен для всех записей.
- `duplicate_count(PARTY_ID) = 0:`
 `name: duplicate_rows_by_primary_key_PARTY_ID`
 `# description: В датамарте не допускаются дубликаты по первичному ключу`
 "Уникальный идентификатор контрагента" (PARTY_ID).

Как выглядит код проверки

```
- row_count = 0:  
  name: business_rule_terminated_NEAR  
  # description: Если сделка терминирована (TERMINATION_DATE IS NOT NULL)  
  # и TERMINATION_DATE <= NEAR_VALUE_DATE,  
  # то FINANCIAL_RESULT_NEAR_CCY = 0 и FINANCIAL_RESULT_NEAR_RUB = 0.  
  filter: TERMINATION_DATE IS NOT NULL  
    AND NEAR_VALUE_DATE IS NOT NULL  
    AND TERMINATION_DATE <= NEAR_VALUE_DATE  
    AND (  
      FINANCIAL_RESULT_NEAR_CCY IS NULL  
      OR FINANCIAL_RESULT_NEAR_CCY <> 0  
      OR FINANCIAL_RESULT_NEAR_RUB IS NULL  
      OR FINANCIAL_RESULT_NEAR_RUB <> 0  
    )
```

Как выглядит DQ-report: приходит сообщением в мессенджер



FSP Data monitoring BOT 12:48 PM

DQ Check: `financial_result_fx_forward_checks`

Results

- Check time: 2026-08-03 09:48:18 UTC
- Tags: ['bonds']
- Scan duration: 20.00s
- Success rate: 98.1% ⚠

TOTAL

54

Passed

53

Failed

0

Warnings

1

Errors

0



1 reply | Follow

Как выглядит отчет в логах



```
INFO:soda.scan:[09:48:18]      check_value: 0
INFO:soda.scan:[09:48:18] null_rows_in_required_field_TAX_BASE_MINUS_UNREALIZED_FX_DIFF [sodacl_string.yml] [PASSED]
INFO:soda.scan:[09:48:18]      check_value: 0
INFO:soda.scan:[09:48:18] null_rows_in_required_field_TERMINATION_DATE_when_STATUS_TERMINATED [sodacl_string.yml] [PASSED]
INFO:soda.scan:[09:48:18]      check_value: 0
INFO:soda.scan:[09:48:18] null_rows_in_required_field_TERMINATION_FEE_PAYOUT_DATE_when_STATUS_TERMINATED [sodacl_string.yml] [PASSED]
INFO:soda.scan:[09:48:18]      check_value: 0
INFO:soda.scan:[09:48:18] null_rows_in_required_field_TERMINATION_FEE_when_STATUS_TERMINATED [sodacl_string.yml] [PASSED]
INFO:soda.scan:[09:48:18]      check_value: 0
INFO:soda.scan:[09:48:18] null_rows_in_required_field_TERMINATION_FEE_CCY_when_STATUS_TERMINATED [sodacl_string.yml] [PASSED]
INFO:soda.scan:[09:48:18]      check_value: 0
INFO:soda.scan:[09:48:18] null_rows_in_required_field_TERMINATION_FEE_DATE_CBR_RATE_when_STATUS_TERMINATED [sodacl_string.yml] [PASSED]
INFO:soda.scan:[09:48:18]      check_value: 0
INFO:soda.scan:[09:48:18] null_rows_in_required_field_TERM_FEE_RUB_when_STATUS_TERMINATED [sodacl_string.yml] [PASSED]
INFO:soda.scan:[09:48:18]      check_value: 0
INFO:soda.scan:[09:48:18] 1/54 checks WARNED:
INFO:soda.scan:[09:48:18]      FINANCIAL_RESULT_FX_FORWARD in FSP
INFO:soda.scan:[09:48:18]      warn_if_CPTY_FULL_NAME_ENG_is_null [sodacl_string.yml] [WARNED]
INFO:soda.scan:[09:48:18]      check_value: 2
INFO:soda.scan:[09:48:18] Only 1 warning. 0 failure. 0 errors. 53 pass.
```

Как выглядит DQ-report: приходит сообщением в мессенджер



FSP Data monitoring BOT last wk.



Passed:

object	layer	biz_dt_start	biz_dt_end	rule_name
FINANCIAL_RESULT_FX_FORWARD	DATAMART	2025-12-31	2025-12-31	FINANCIAL_RESUL
FINANCIAL_RESULT_FX_FORWARD	DATAMART	2025-12-31	2025-12-31	VALUE_DATE_grea
FINANCIAL_RESULT_FX_FORWARD	DATAMART	2025-12-31	2025-12-31	QUOTED_CCY_RAT
FINANCIAL_RESULT_FX_FORWARD	DATAMART	2025-12-31	2025-12-31	DERIVATIVE_FLAG
			2025-12-	

Как выглядят сэмплы: сделки выводятся из витрины, те, в которых обнаружены аномалии

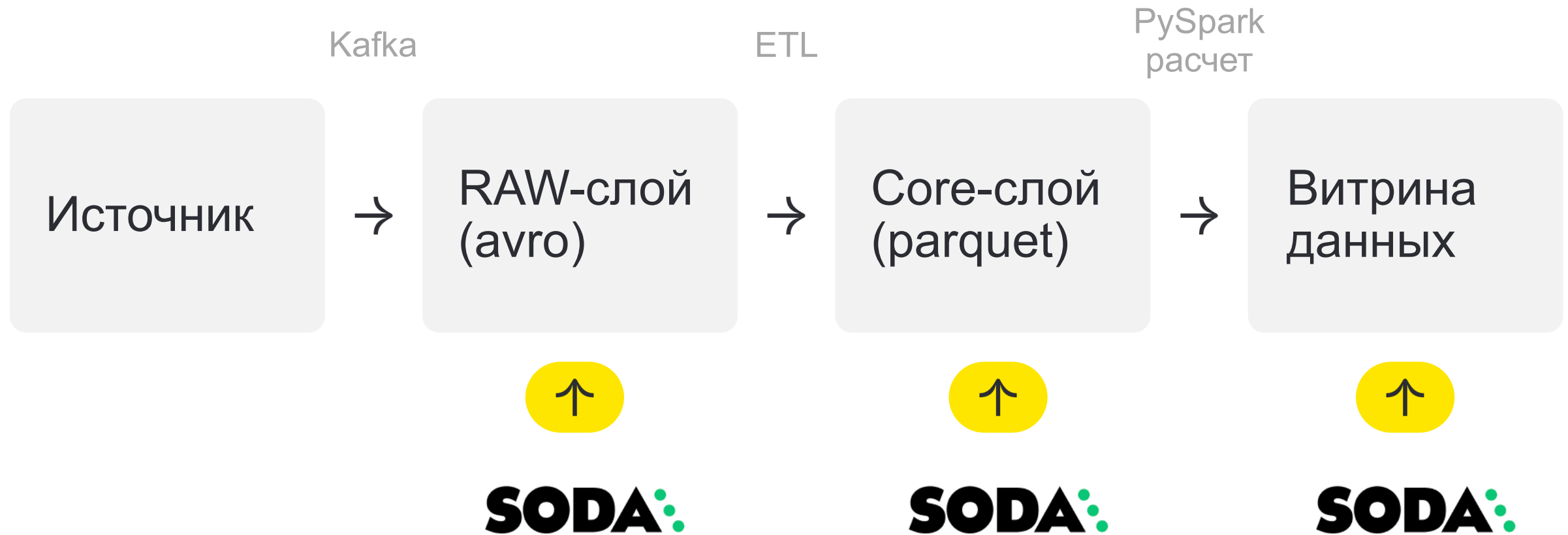


Results 1 X

`select * from read_parquet('s3://data-quality-p/financial_result_fx_forward_checks/financial_result_')` Enter a SQL expression to filter results (use Ctrl+Space)

Grid	ABC DEAL_ID	ABC DEAL_NUMBER	ABC ORIG_DEAL_NUMBER	ABC COUNTERPARTY_ID	ABC CPTY_FULL_NAME	ABC CPTY_FULL_NAME_ENG	ABC CPTY_ADDRESS	ABC CPTY_ADDRESS_CC
1	20*6*7805672	2025072901964	2025072901964	CAAHZG	Моушн Продактс	[NULL]	Российская Федерация	RU
2	20*6*7926775	2025092601516	2025092601516	CAAHZG	Моушн Продактс	[NULL]	Российская Федерация	RU

Платформенное решение – полноценный DQ-сервис



PyDeequ vs Soda: что выбрать для каких задач



Критерий	PyDeequ	Soda Core
Идеальный сценарий	Локальные гейты на критичных шагах расчёта	Платформенный DQ-сервис: стандарты, прозрачность, мониторинг
Лучшее место в потоке	Рядом с трансформациями: после join / agg / step N	После публикации витрины или таблицы
Где живут правила	В репозитории и коде конкретного расчёта	В централизованных checks / стандартах
Читаемость и ревью	Средняя: это код	Высокая: декларативная спецификация
Кастомные доменные правила	Очень сильный: можно выразить любую PySpark-логику	Хорош для типовых checks и SQL-based правил, сложный кастом дороже
Масштабирование на много витрин	Дороже при росте числа мест поддержки	Проще тиражировать как единый DQ-контур
Парадигма	DQ as code	DQ as declaration / config
Статистика и профилирование	Сильная сторона: метрики и аномалии	Чаще декларативные пороги и регулярный контроль

Эксплуатация и подводные камни



Критерий	PyDeequ	Soda Core
Риск / антипаттерн	Проверки расползлись по коду, нет единого обзора	Checks есть, но не влияют на выпуск данных, нет gating
Как лечится	Гайдлайны, общий формат метрик/логов, переиспользование	Связать с пайплайном, определить ownership и SLA на реакции
Ownership	Владельцы обычно команда разработки расчётов	Платформа поддерживает контур, домены владеют правилами
Алертинг и реакции	Интеграции руками через оркестратор/инциденты	Проще настроить уровни алертов после прогона
Мониторинг / витрина результатов	Нужно собирать самим	Проще сделать единый обзор, хорошо ложится на Grafana
Root cause	Быстро, если диагностика рядом с шагом трансформации	Видно, какой check упал, но глубина часто уходит в анализ данных
Стоимость вычислений	Можно оптимизировать внутри job	Отдельный scan, нагрузку надо планировать
Переиспользование правил	Через модули/функции, нужна дисциплина	Через единые стандарты и наборы checks

Полная автоматизация цикла с помощью DQ-генератора

Manual Exploratory Testing



PyDeequ Cheeks



SODA Cheeks



DG-generator



Переход к полной автоматизации цикла

DGTL



01

Витрин и полей
становится больше

02

Правила похожи,
но зависят от домена
и контекста

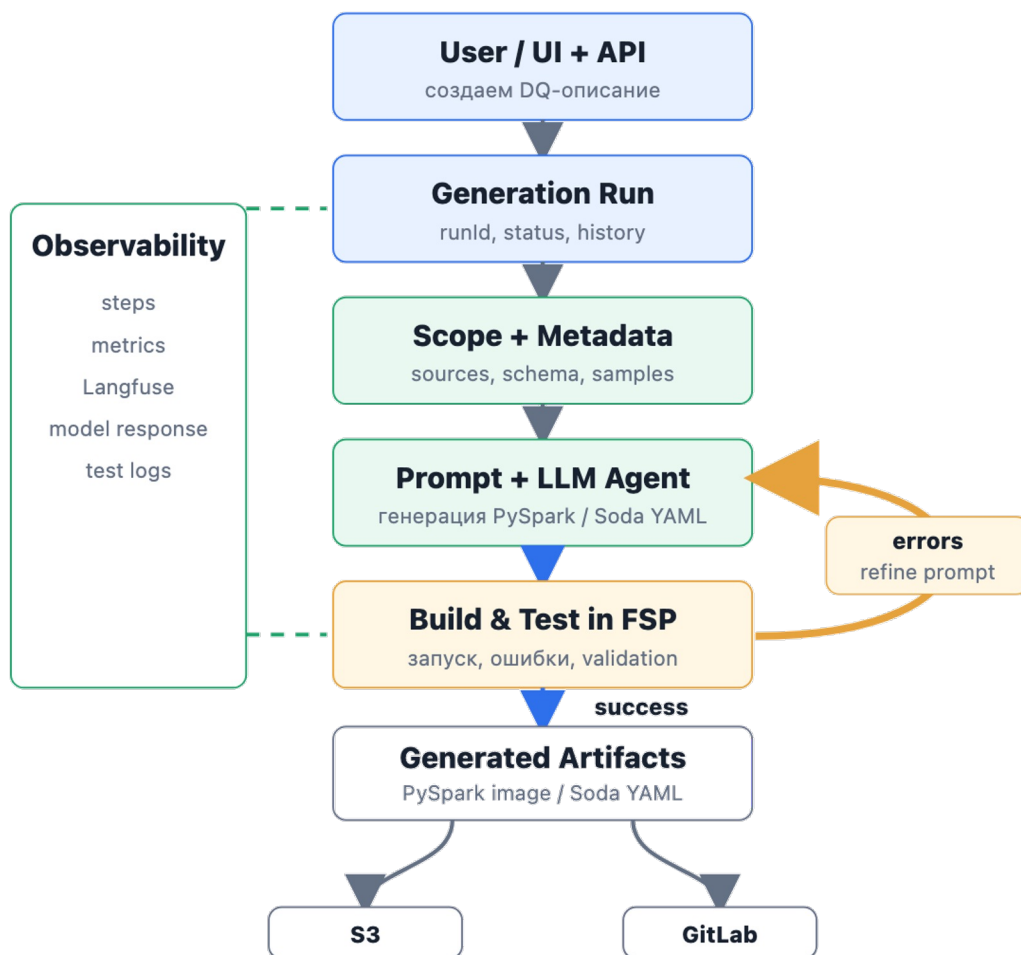
03

Аналитик знает
смысл, инженер
знает исполняемый
формат

04

Ручное покрытие
повторяемыми
checks тормозит
масштабирование

Что мы сделали



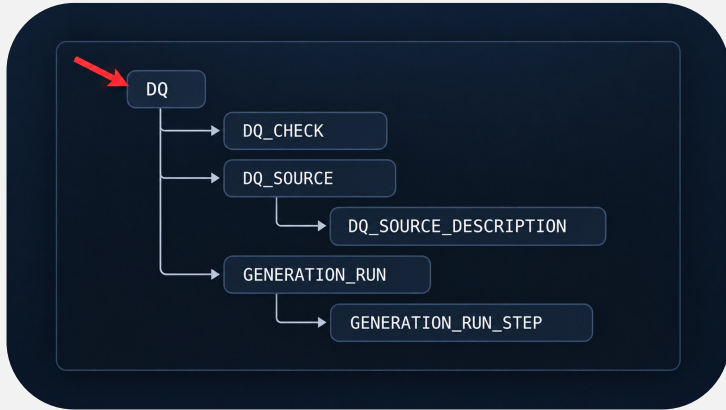
DQ-генератор — Spring Boot сервис, который помогает заводить и генерировать Data Quality проверки

Сервис хранит DQ-описания, источники данных, правила проверок и историю генераций

Затем с помощью LLM генерирует исполняемый код на PySpark или SODA YAML

Тестирует результат и сохраняет артефакты

Первый этап - DQ: создаем описание и бизнес-контекст



Первый этап - **DQ**: создаем описание и бизнес-контекст

1 DQ 2 Sources 3 Checks 4 Runs

Редактирование DQ

ID
3ea97268-c74f-491a-8eb3-2012618fb945

Название
DQ-Tax-financial_result_coupon

Бизнес описание
Функциональные DQ-проверки для регистра по купону

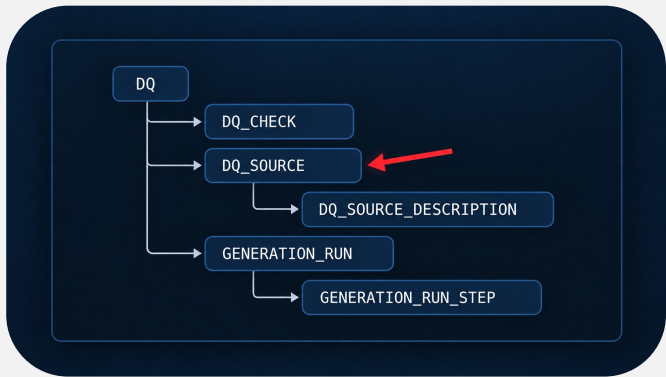
Дополнительный промпт пользователя

Название расчета
financial_result_coupon

Комментарий

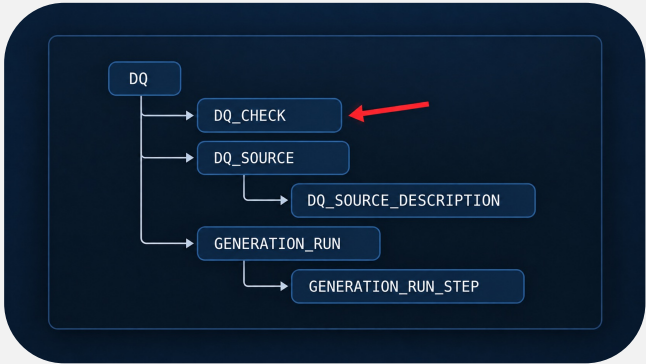
Дата создания
24.08.2026, 14:26:47

Второй этап - DQ_SOURCE: указываем источник данных



Второй этап - **DQ_SOURCE**:
указываем источник данных

Третий этап - DQ_CHECK: задаем конкретное правило



Третий этап - DQ_CHECK: задаем конкретное правило

3

Checks

4

Runs

fin-ml:finai-large

Тест

ruaicad

Checks

All sources

not_null_BOND_TAX_TYPE

source: FINANCIAL_RESULT_COUPON | type: Per Row

Поле "Признак для группировки по типу и ставке" (BOND_TAX_TYPE) должно быть заполнено для всех записей.

null_rows_in_required_field_COUPON_SUM_RUB_CUR_LAST_P_when_POS_DELIVERY_FACT_DATE_before_REPORT_START_DATE_and_NCD_CR_TYPE_accrual

source: FINANCIAL_RESULT_COUPON | type: Per Row

Если значение поля "Фактическая дата поставки" (POS_DELIVERY_FACT_DATE) < даты начала отчетного периода (REPORT_START_DATE) и поле "Признак купона по кассе/по начислению" (NCD_CR_TYPE) = 'По начислению', то поле "Сумма купона, начисленного по состоянию за 31.12.{yyyy-1}, в рублях" (COUPON_SUM_RUB_CUR_LAST_P) должно быть заполнено.

null_rows_in_required_field_OUT_POS_NCD_RUB_CUR_when_OUT_POS_DIAS_NUM_present

source: FINANCIAL_RESULT_COUPON | type: Per Row

Если поле "Номер Диасофт сделки выбытия" (OUT_POS_DIAS_NUM) заполнено, поле "Сумма НКД в сделке выбытия в рублях" (OUT_POS_NCD_RUB_CUR) должно быть заполнено.

Просмотр проверки

Редактировать

Код проверки

null_rows_in_required_field_COUPON_SUM_RUB_CUR_LAST_P_when_POS_DELIVERY_FACT_DATE_before_REPORT_START_DA

DQ ID

3ea97268-c74f-491a-8eb3-2012618fb945

Тип

PER_ROW

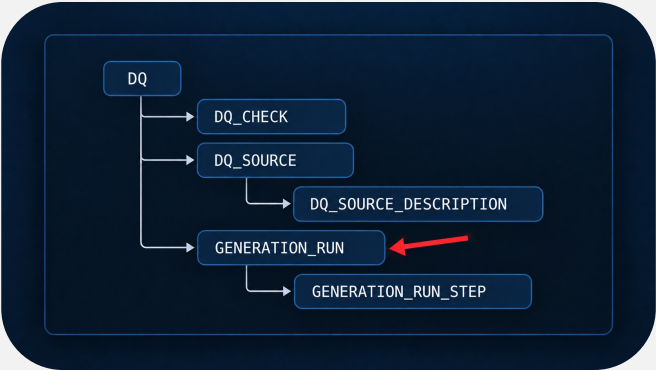
Source

FINANCIAL_RESULT_COUPON

Правило

Если значение поля "Фактическая дата поставки" (POS_DELIVERY_FACT_DATE) < даты начала отчетного периода (REPORT_START_DATE) и поле "Признак купона по кассе/по начислению" (NCD_CR_TYPE) = 'По начислению', то поле "Сумма купона, начисленного по состоянию за 31.12.{yyyy-1}, в рублях" (COUPON_SUM_RUB_CUR_LAST_P) должно быть заполнено.

Четвертый этап - GENERATION_RUN: prompt, status, response, image



Четвертый этап -
GENERATION_RUN:
prompt, status, response,
image

✓

DQ

✓

Sources

✓

Checks

4

Runs

fin-ml:finai-large

Tect

ruaiacad

Runs

Run #d1dd0724

Status: SUCCESS | Created: 8/28/2026, 4:30:46 PM

Run #c4e509bb

Status: SUCCESS | Created: 8/28/2026, 3:32:52 PM

Run #4bda632a

Status: SUCCESS | Created: 8/25/2026, 10:27:08 AM

Run #49cbe514

Status: FAILED | Created: 8/24/2026, 4:53:59 PM

Run #111ee111

Status: FAILED | Created: 8/24/2026, 3:30:27 PM

Run #379ce07c

Status: FAILED | Created: 8/24/2026, 2:33:53 PM

Run Details

Run ID: 4bda632a-3f91-4e11-b022-00fc6f571c11

Metrics

Status: SUCCESS

Created: 8/25/2026, 10:27:08 AM

Finished: 8/25/2026, 10:30:43 AM

Used model: openai

Generation Steps

3 steps

1. LLM_API_REQUEST

SUCCESS

1m 56s

10:27:09 AM → 10:29:05 AM

2. TESTS_RUNNING

SUCCESS

1m 11s

10:29:05 AM → 10:30:16 AM

Job: 4aaf0f88-27f7-476e-8ef8-a438a3f82b64

3. LLM_API_REQUEST

SUCCESS

26s

10:30:16 AM → 10:30:43 AM

Effective Prompt

Run Branch

Ad-hoc

Calculation Line

tax-report

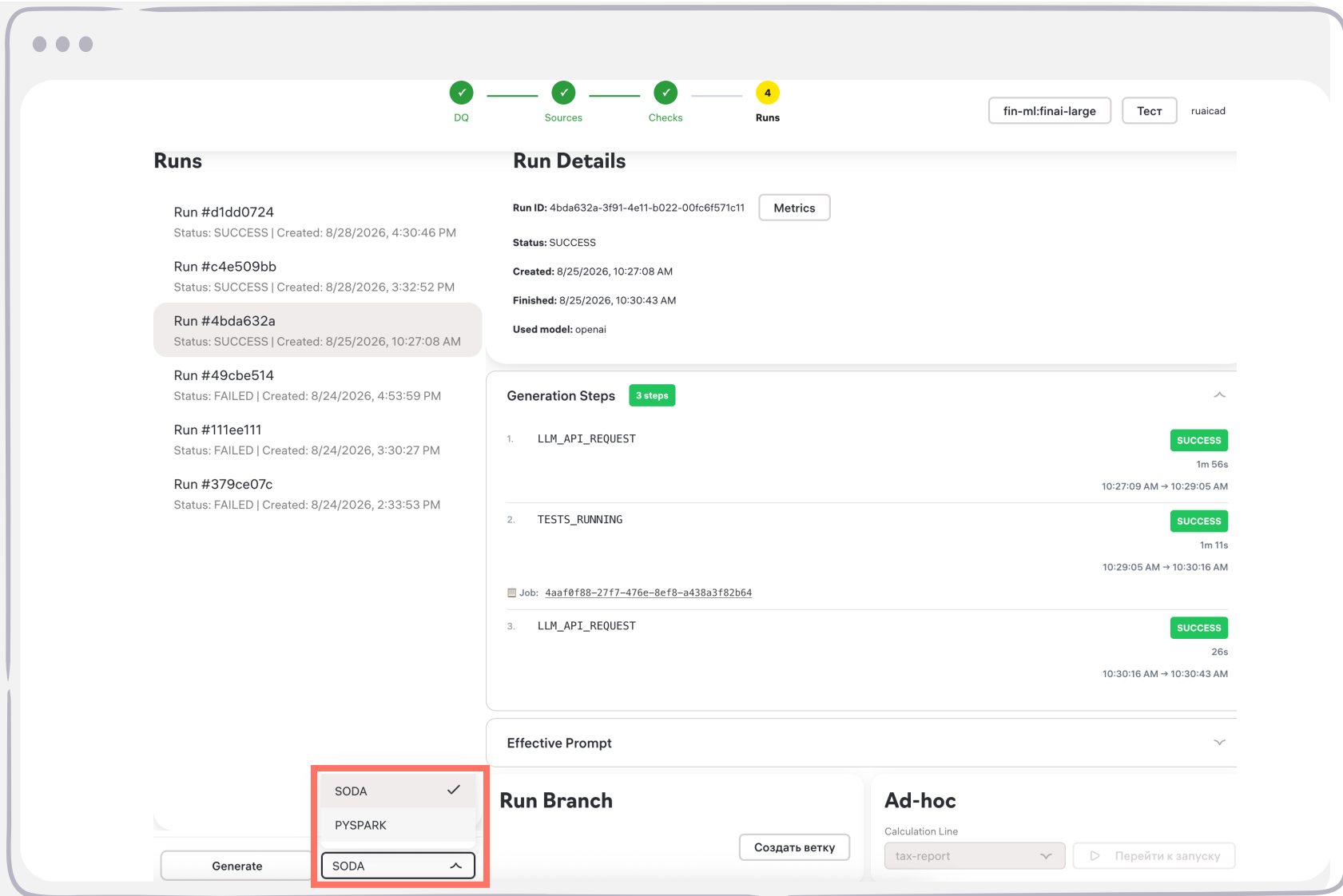
Перейти к запуску

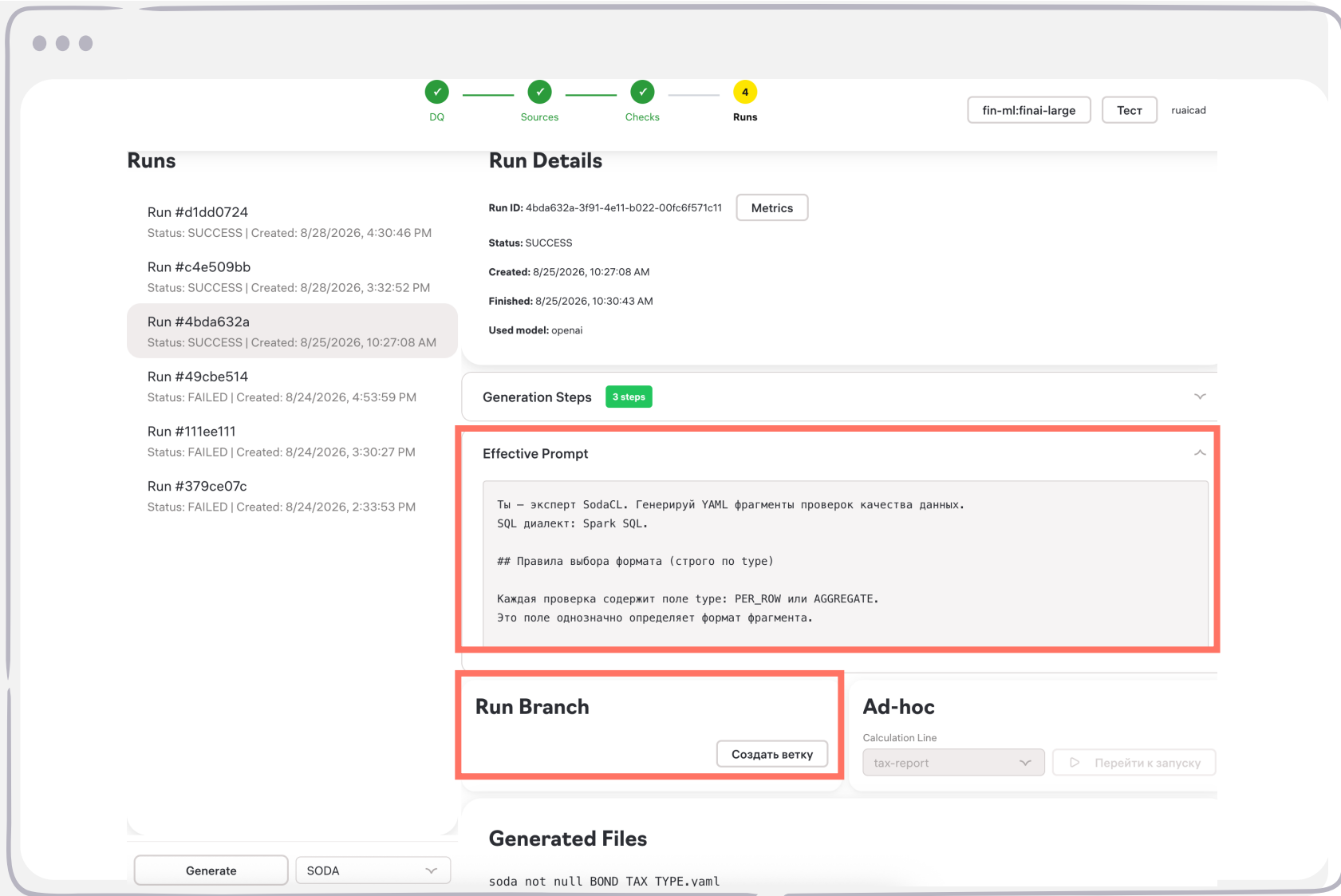
Generate

SODA

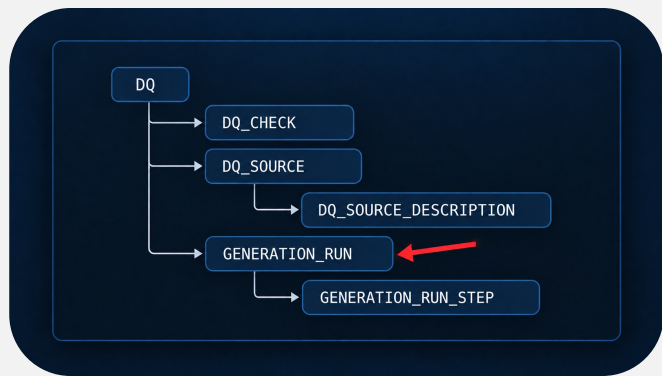
Создать ветку

41





Четвертый этап - GENERATION_RUN: prompt, status, response, image



The screenshot displays the application interface with a 'Runs' list on the left and a 'Run Details' view on the right. A modal dialog box titled 'Создать ветку запуска' (Create Run Branch) is open in the center.

Runs List:

- Run #d1dd0724: Status: SUCCESS | Created: 8/28/2026, 4:30:46 PM
- Run #c4e509bb: Status: SUCCESS | Created: 8/28/2026, 3:32:52 PM
- Run #4bda632a: Status: SUCCESS | Created: 8/25/2026, 10:27:08 AM
- Run #49cbe514: Status: FAILED | Created: 8/24/2026, 10:27:08 AM
- Run #111ee111: Status: FAILED | Created: 8/24/2026, 10:27:08 AM
- Run #379ce07c: Status: FAILED | Created: 8/24/2026, 10:27:08 AM

Run Details View:

- Run ID: 4bda632a-3f91-4e11-b022-00fc6f571c11
- Status: SUCCESS
- Created: 8/25/2026, 10:27:08 AM

Создать ветку запуска (Create Run Branch) Dialog:

- Calculation Repository: tax-report
- Git Repository: https://gitlabci.raiffeisen.ru/fsp/fsp-calc/calculations/tax-report/fsp-taxreport-ce
- Полное имя ветки (Full branch name): feature/FSP-34938
- Buttons: Отмена (Cancel), Создать (Create)

Ad-hoc View:

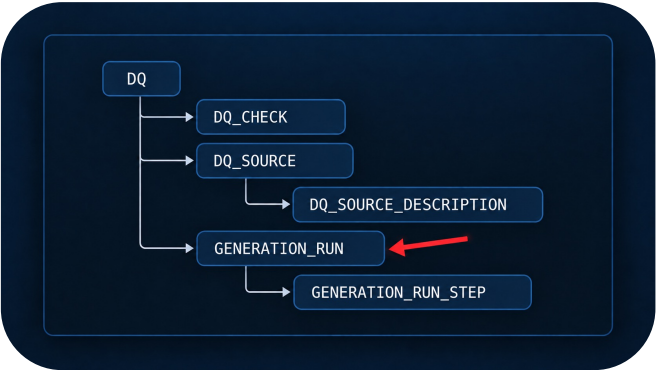
- Calculation Line: tax-report
- Buttons: tax-report, Перейти к запуску (Go to Run)

Failed Rows Query:

```
soda_not_null_BOND_TAX_TYPE.yaml

failed rows:
fail query: |
SELECT *
FROM FINANCIAL_RESULT_COUPON
WHERE BOND_TAX_TYPE IS NULL OR TRIM(BOND_TAX_TYPE) = ''
```

DQ- generator



fin-ml:finai-large Тест ruaicad

Runs

- Run #d1dd0724
Status: SUCCESS | Created: 8/28/2026, 4:30:46 PM
- Run #c4e509bb
Status: SUCCESS | Created: 8/28/2026, 3:32:52 PM
- Run #4bda632a
Status: SUCCESS | Created: 8/25/2026, 10:27:08 AM
- Run #49cbe514
Status: FAILED | Created: 8/24/2026, 4:53:59 PM
- Run #111ee111
Status: FAILED | Created: 8/24/2026, 3:30:27 PM
- Run #379ce07c
Status: FAILED | Created: 8/24/2026, 2:33:53 PM

Run Details

Run ID: 4bda632a-3f91-4e11-b022-00fc6f571c11 Metrics

Status: SUCCESS

Created: 8/25/2026, 10:27:08 AM

Finished: 8/25/2026, 10:30:43 AM

Used model: openai

Generation Steps 3 steps

Effective Prompt

Run Branch

feature/FSP-34938 Удалить ветку

Ad-hoc

Calculation Line

tax-report ▶ Перейти к запуску

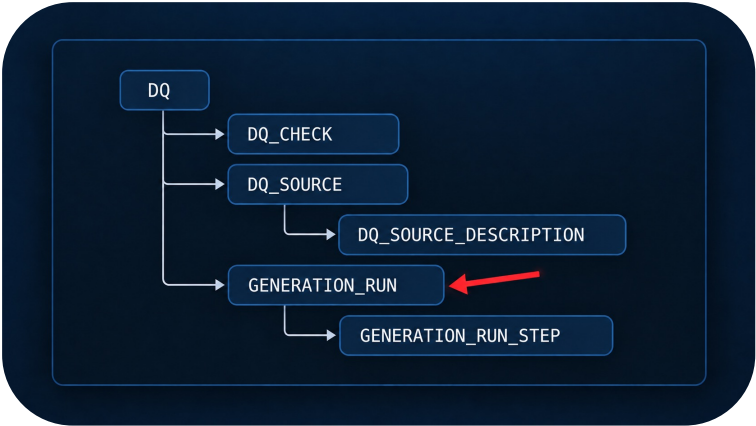
Generated Files

soda_not_null_BOND_TAX_TYPE.yaml

```
failed rows:
fail query: |
SELECT *
FROM FINANCIAL_RESULT_COUPON
WHERE BOND_TAX_TYPE IS NULL OR TRIM(BOND_TAX_TYPE) = ''
```

Generate SODA

DQ- generator



FSP-34938: Create DQ run branch code for dq_tax_financial_result_coupon

FSP-34938: Create DQ run branch code for dq_tax_financial_result_coupon run 4bda632a-3f91-4e11-b022-00fc6f571c11

parent [dc89140d](#)

Branches [feature/FSP-34938](#)

No related tags found

No related merge requests found

Pipeline [#18463749](#) running
11 seconds ago

Changes [1](#) Pipelines [1](#)

Showing [1 changed file](#) with [38 additions](#) and [0 deletions](#)

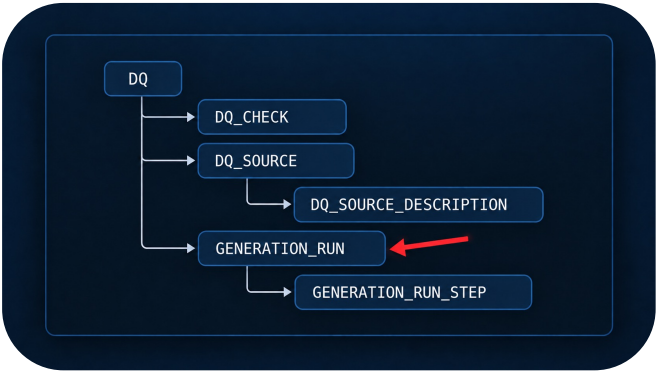
[Hide whitespace changes](#)

financial_result_coupon/dq_tax_financial_result_coupon_generated_soda_checks.yaml 0 → 100644

+38 -0

```
1 + calculation_line: fspdata
2 + owner: ''
3 + mm_channel_id: []
4 + mm_notify_rules: []
5 + tags: []
6 + input_types:
7 +   - type_name: FINANCIAL_RESULT_COUPON
8 +     mode: single
9 + checks:
10 +   checks for FINANCIAL_RESULT_COUPON:
11 +     - failed rows:
12 +       fail query: |-
13 +         SELECT *
14 +         FROM FINANCIAL_RESULT_COUPON
15 +         WHERE BOND_TAX_TYPE IS NULL OR TRIM(BOND_TAX_TYPE) = ''
16 +       name: not_null_BOND_TAX_TYPE
```

DQ- generator



✓
DQ

✓
Sources

✓
Checks

4
Runs

fin-ml:finai-large

Тест

ruicad

feature/FSP-34938

Удалить ветку

Calculation Line

tax-report

▶ Перейти к запуску

Run #d1dd0724
Status: SUCCESS | Created: 8/28/2026, 4:30:46 PM

Run #c4e509bb
Status: SUCCESS | Created: 8/28/2026, 3:32:52 PM

Run #4bda632a
Status: SUCCESS | Created: 8/25/2026, 10:27:08 AM

Run #49cbe514
Status: FAILED | Created: 8/24/2026, 4:53:59 PM

Run #111ee111
Status: FAILED | Created: 8/24/2026, 3:30:27 PM

Run #379ce07c
Status: FAILED | Created: 8/24/2026, 2:33:53 PM

Generated Files

soda_not_null_BOND_TAX_TYPE.yaml

failed rows:
fail query: |
SELECT *
FROM FINANCIAL_RESULT_COUPON
WHERE BOND_TAX_TYPE IS NULL OR TRIM(BOND_TAX_TYPE) = ''

👍

🗨

soda_null_rows_in_required_field_COUPON_SUM_RUB_CUR_LAST_P_when_POS_DELIVERY_FACT_DATE_before_REPORT_ST

failed rows:
fail query: |
SELECT *
FROM FINANCIAL_RESULT_COUPON
WHERE POS_DELIVERY_FACT_DATE < REPORT_START_DATE
AND NCD_CR_TYPE = 'По начислению'
AND COUPON_SUM_RUB_CUR_LAST_P IS NULL

👍

🗨

Комментарий

0/2000

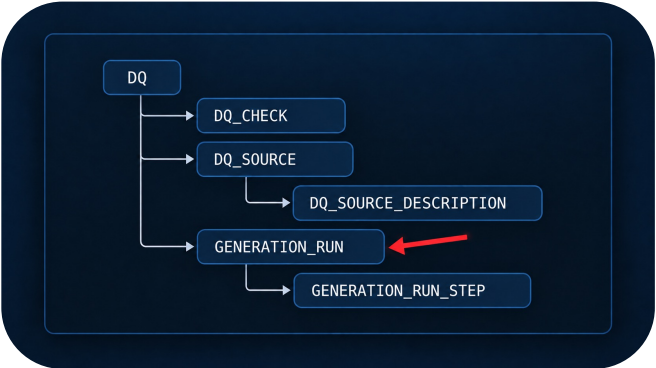
Сохранить

Отмена

Generate

SODA

DQ- generator



✓

DQ

✓

Sources

✓

Checks

4

Runs

fin-ml:finai-large

Тест

ruaicad

Runs

Run #d1dd0724
Status: SUCCESS | Created: 8/28/2026, 4:30:46 PM

Run #c4e509bb
Status: SUCCESS | Created: 8/28/2026, 3:32:52 PM

Run #4bda632a
Status: SUCCESS | Created: 8/25/2026, 10:27:08 AM

Run #49cbe514
Status: FAILED | Created: 8/24/2026, 4:53:59 PM

Run #111ee111
Status: FAILED | Created: 8/24/2026, 3:30:27 PM

Run #379ce07c
Status: FAILED | Created: 8/24/2026, 2:33:53 PM

Run Details

Run ID: 4bda632a-3f91-4e11-b022-00fc6f571c11

Status: SUCCESS

Created: 8/25/2026, 10:27:08 AM

Finished: 8/25/2026, 10:30:43 AM

Used model: openai

Metrics

Generation Steps 3 steps

Effective Prompt

Run Branch

feature/FSP-34938

Удалить ветку

Ad-hoc

Calculation Line

tax-report

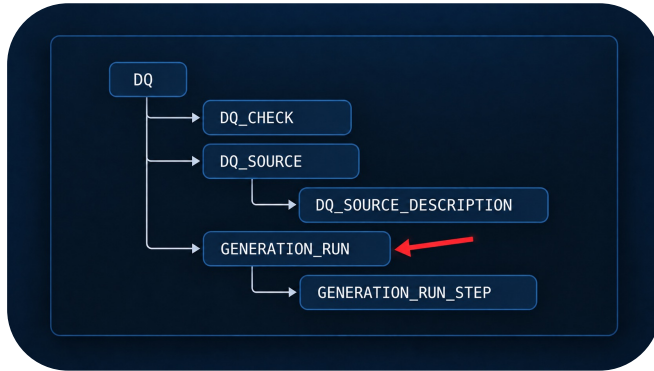
▶ Перейти к запуску

Откроется GitLab с заполненными параметрами запуска.

Generated Files

48

DQ- generator

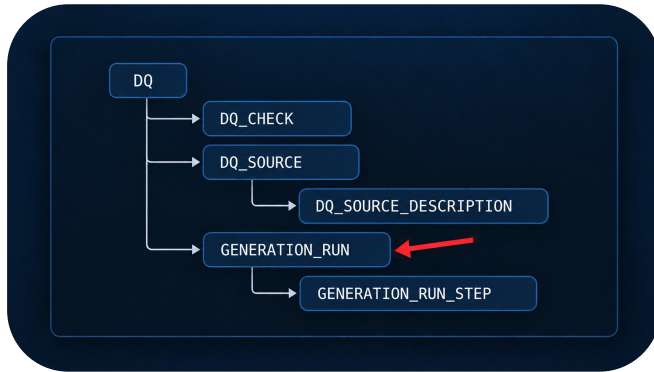


Preview Log View Scores Formatted JSON

Output

Path	Value
status	"SUCCESS"
content	" Все проверки успешно протестированы. Вот YAML фрагменты: ```yaml failed rows: fail query: SELECT * FROM FINANCIAL_RESULT_COUPON WHERE BOND_TAX_TYPE IS NULL OR TRIM(BOND_TAX_TYPE) = '' ``` ```yaml failed rows: fail query: SELECT * FROM FINANCIAL_RESULT_COUPON WHERE POS_DELIVERY_FACT_DATE < REPORT_START_DATE AND NCD_CR_TYPE = 'По начислению' AND COUPON_SUM_RUB_CUR_LAST_P IS NULL ``` ```yaml failed rows: fail query: SELECT * FROM FINANCIAL_RESULT_COUPON WHERE (OUT_POS_DIAS_NUM IS NOT NULL AND TRIM(OUT_POS_DIAS_NUM) <> '') AND OUT_POS_NCD_RUB_CUR IS NULL ``` ```yaml failed rows: fail query: SELECT * FROM FINANCIAL_RESULT_COUPON WHERE COUPON_PER_BOND_AMT = 0 AND COUPON_RATE <> 0 ```"

DQ- generator



Path	Value
status	"FAILED"
content	" Код синтаксически верен, но тестирование не прошло из-за инфраструктурной ошибки (500 Internal Server Error, InvalidAccessKeyId при сборке Docker-образа). Финальный вариант кода: **Файл 1: not_null_BOND_TAX_TYPE** ```python from typing import Mapping from pyspark.sql import DataFrame, SparkSession, functions as F, types as T def check(spark_session: SparkSession, sources: Mapping[str, DataFrame]) -> DataFrame: df_source = sources["DATAMART_FINANCIAL_RESULT_COUPON"] df_failed = df_source.where(F.col("BOND_TAX_TYPE").isNull()).select(F.lit("not_null_BOND_TAX_TYPE").alias("CHECK_ID"), F.col("PAIR_DEAL_ID").cast(T.StringType()).alias("KEY_VALUE"), F.lit("FINANCIAL_RESULT_COUPON").alias("SOURCE_NAME"), F.lit("FINANCIAL_RESULT_COUPON").alias("TARGET_NAME"), F.col("BOND_TAX_TYPE").cast(T.StringType()).alias("SOURCE_VALUE"), F.lit(None).cast(T.StringType()).alias("TARGET_VALUE"),) df_result = df_failed.select(F.col("CHECK_ID"), F.col("KEY_VALUE"), F.col("SOURCE_NAME"), F.col("TARGET_NAME"), F.col("SOURCE_VALUE"), F.col("TARGET_VALUE"),) return df_result ```

Четвертый этап - GENERATION_RUN: Реализация на pySpark



```
def check(spark_session: SparkSession, sources: Mapping[str, DataFrame]) ->
DataFrame:
    df_source = sources["DATAMART_TAX_FULL_CLIENT_INFO"]

    df_grouped = df_source.groupBy(F.col("PARTY_ID")).agg(
        F.count(F.lit(1)).alias("ROW_COUNT")
    )

    df_failed = df_grouped.where(F.col("ROW_COUNT") > F.lit(1)).select(
        F.lit("duplicate_rows_by_primary_key_PARTY_ID").alias("CHECK_ID"),
        F.col("PARTY_ID").cast(T.StringType()).alias("KEY_VALUE"),
        F.lit("TAX_FULL_CLIENT_INFO").alias("SOURCE_NAME"),
        F.lit("TAX_FULL_CLIENT_INFO").alias("TARGET_NAME"),
        F.col("ROW_COUNT").cast(T.StringType()).alias("SOURCE_VALUE"),
        F.lit(None).cast(T.StringType()).alias("TARGET_VALUE"),
    )
```

Почему использование DQ-генератора лучше, чем разовое использование LLM:



Меньше галлюцинаций

Сервис дает модели структурированный контекст: sources, sample-данные, типы колонок, бизнес-описания, конкретные checks

Генерация проверяется реальным запуском

Система сохраняет код, собирает образ, запускает тесты на платформе и возвращает ошибки обратно модели

Результат воспроизводим

У каждого запуска есть runId, effective prompt, модель, статус, шаги, model response, артефакты в S3

Жизненный цикл, а разовая генерация

DQ можно создать, обновить, посмотреть шаги, получить код, открыть adhoc, создать ветку в GitLab

Поддерживаются разные форматы

Один и тот же доменный ввод может уйти в PySpark или SODA

Максимальный эффект дала не замена инструмента, а разделение уровней



PyDeequ Cheeks

Локальные
SPARK-инварианты



SODA Cheeks

Платформенный
DQ-контур



DG-generator

Автоматизация
создания проверок



DGTL

Спасибо за внимание



Алина Ивановичева
Райффайзен Банк

