Как протестировать сложный DWH и сохранить здравый рассудок

Основано на реальных событиях

Карандасов Евгений

14.09.2024

Почему я об этом говорю?

Почему этот доклад стоит послушать?

Карандасов Евгений

старший QA Auto Engineer

Почему я?

Опыт

- 12 лет в IT
- 7 лет в Т-банке
- Все это время работаю в DWH

Разнообразие задач

- Тестирование данных в хранилище
- Обеспечение качества процессов их выгрузки

Взгляд с разных сторон

- Тестирование в бизнес-линиях
- Тестирование внутри сервисной команды

Почему эта тема?

DWH – это сложно

- Зоопарк инструментов и технологий
- Огромное количество процессов
- Свои особенности в каждой системе

Почему эта тема?

DWH – это сложно

- Зоопарк инструментов и технологий
- Огромное количество процессов
- Свои особенности в каждой системе

DWH – это интересно

- Множество самых разных проектов в рамках нашей платформы данных
- Все системы очень интересно разрабатывать и интегрировать
- И очень интересно тестировать

Почему эта тема?

DWH – это сложно

- Зоопарк инструментов и технологий
- Огромное количество процессов
- Свои особенности в каждой системе

DWH – это интересно

- Множество самых разных проектов в рамках нашей платформы данных
- Все системы очень интересно разрабатывать и интегрировать
- И очень интересно тестировать

DWH – это почетно

- Самая разнообразная экспертиза у всех профессий внутри управления
- Решение нетривиальных задач
- Обеспечение руководства важной отчетностью

Немного цифр

01

Около **8500** процессов, использующих наши компоненты

Немного цифр

01

Около **8500** процессов, использующих наши компоненты

02

9 целевых систем

Немного цифр

01

Около **8500** процессов, использующих наши компоненты

02

9 целевых систем

03

16 системных компонент

Немного цифр

01

Около **8500** процессов, использующих наши компоненты

02

9 целевых систем

03

16 системных компонент

04

Эти 16 компонент проверяют:
3 тысячи Unit,
5,5 тысяч функциональных тестов

Немного цифр

01

Около **8500** процессов, использующих наши компоненты

02

9 целевых систем

03

16 системных компонент

04

Эти 16 компонент проверяют:
3 тысячи Unit,
5,5 тысяч функциональных тестов

05

5 DEV
и **2 QA** в команде

Т БАНК

Что же за инструмент мы делаем?

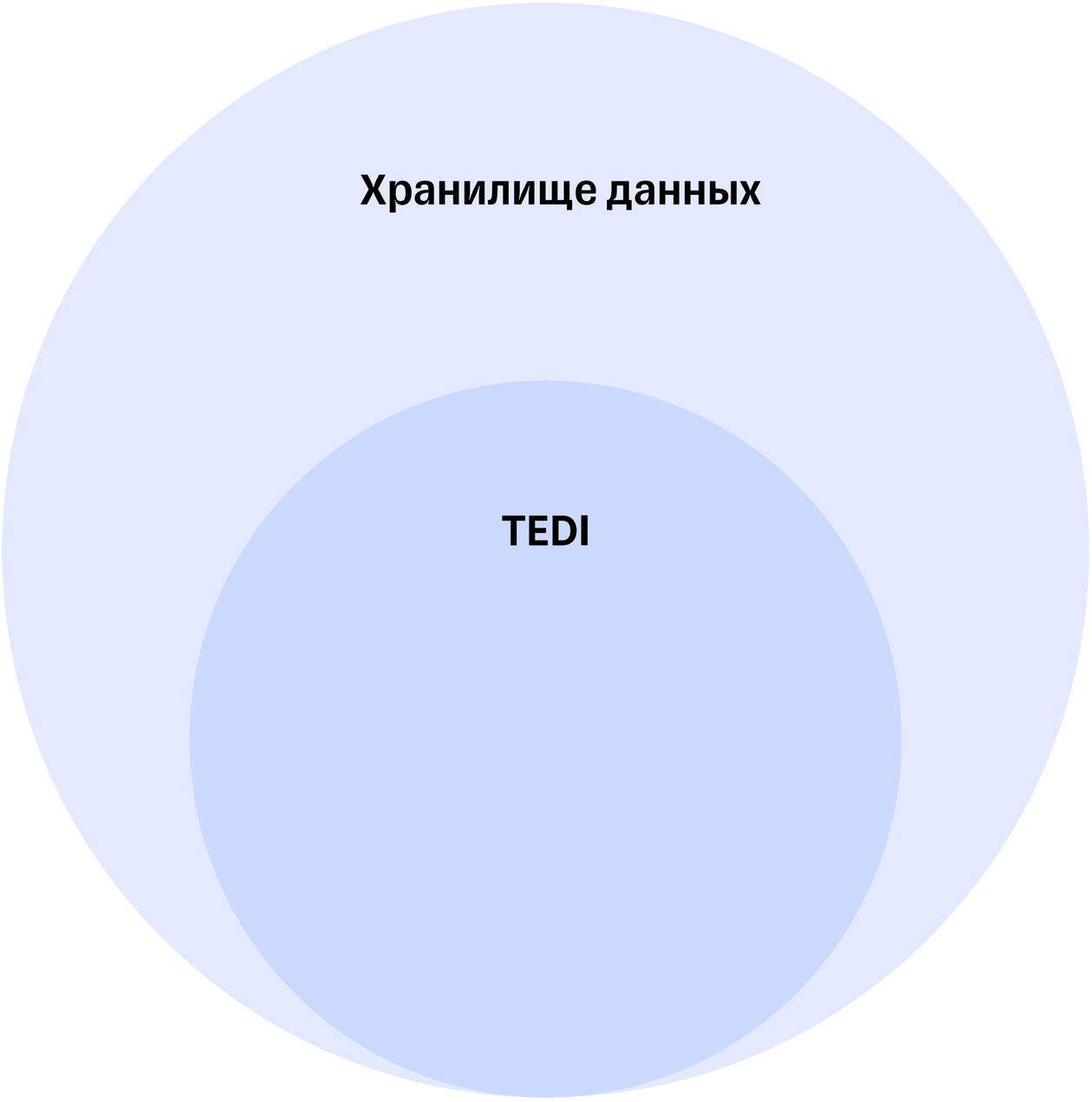
TEDI & dwh_etl_tools

Доклад М.Иванова
на SmartData

Хранилище данных

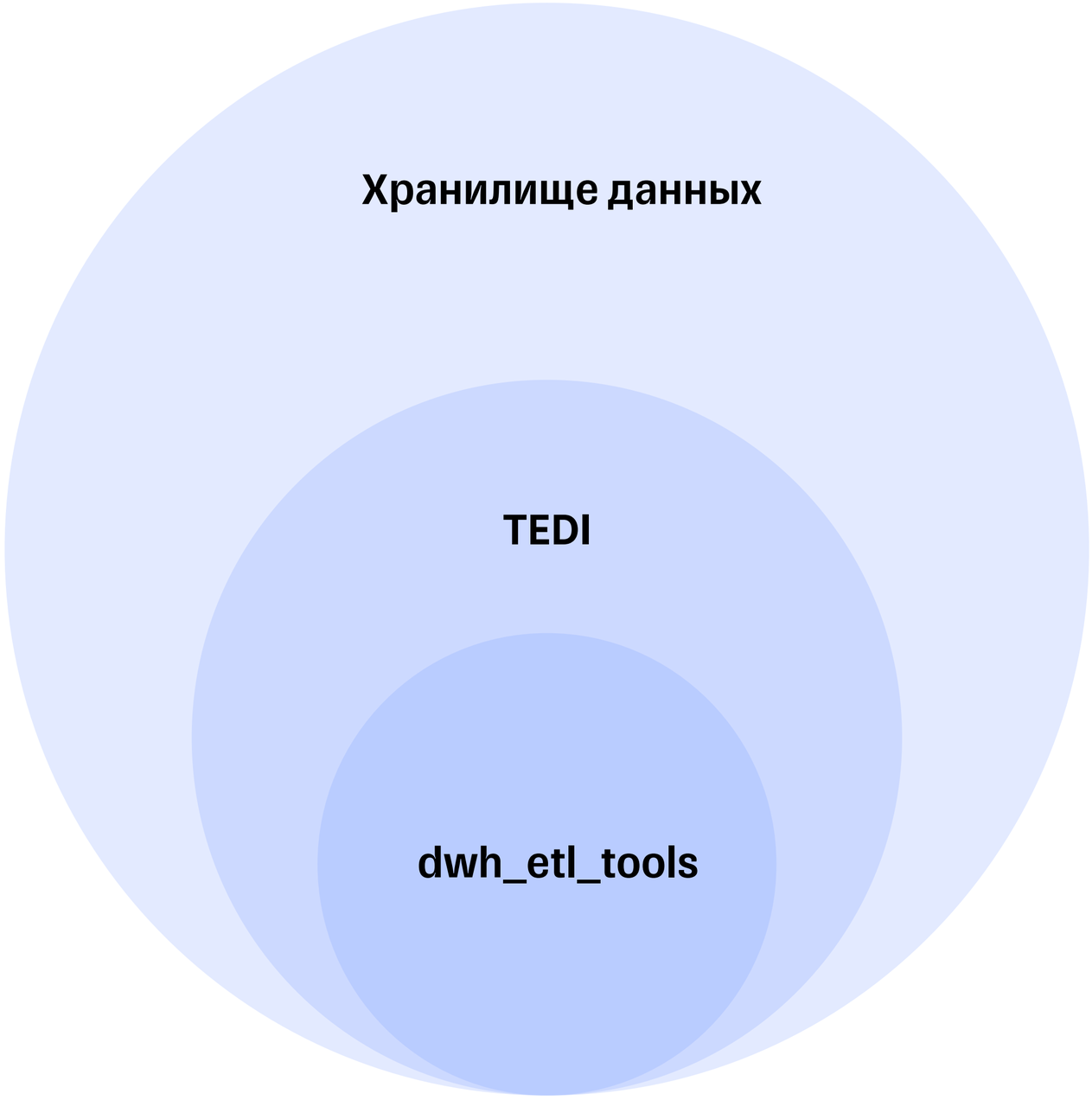
TEDI & dwh_etl_tools

Доклад М.Иванова
на SmartData



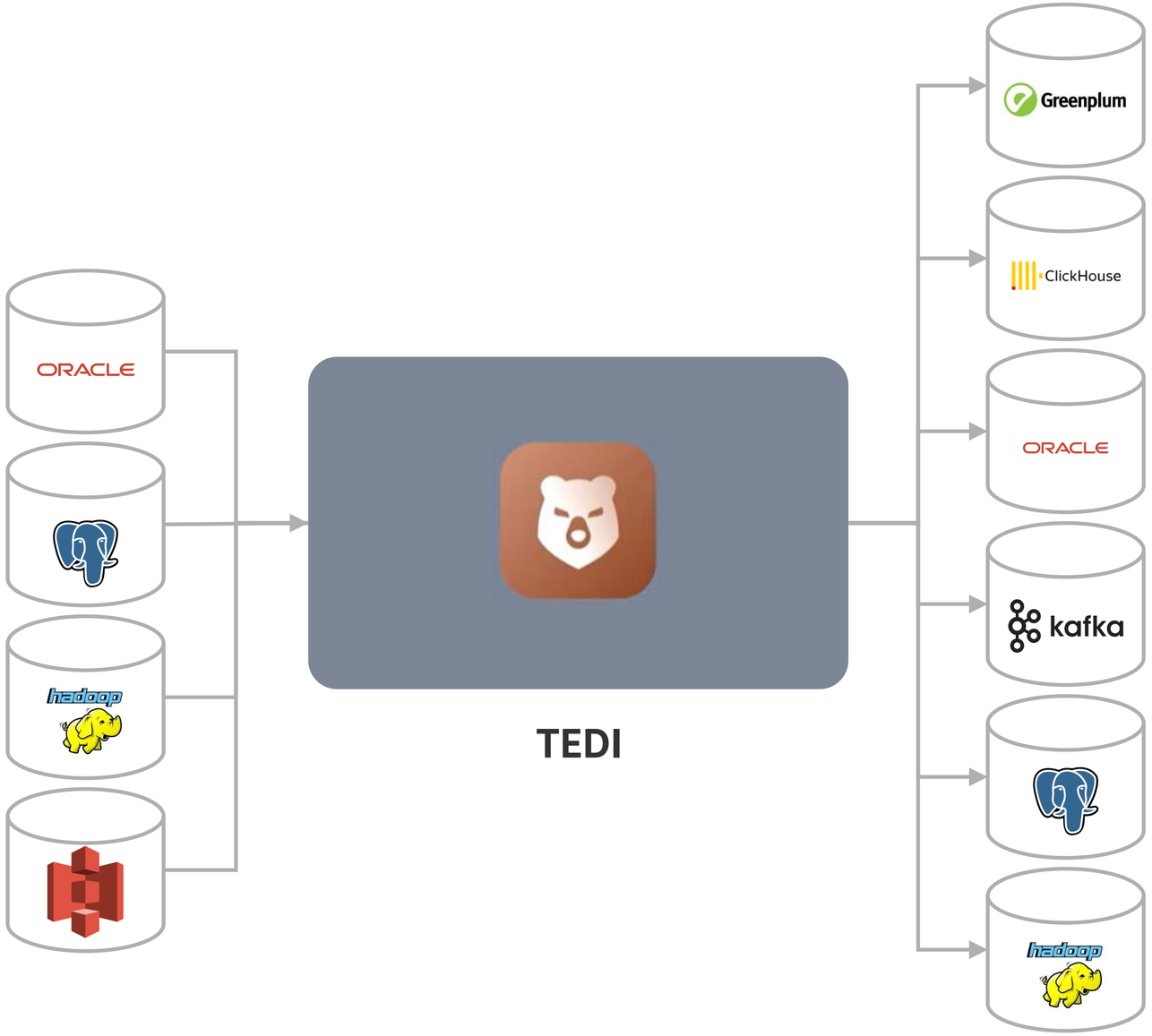
TEDI & dwh_etl_tools

Доклад М.Иванова
на SmartData



TEDI & dwh_etl_tools

Доклад М.Иванова
на SmartData



Что делает библиотека?

dwh_etl_tools – python библиотека с системными компонентами

Используется для следующих задач:

- Загрузка \ Выгрузка данных в\из различные системы

Что делает библиотека?

dwh_etl_tools – python библиотека с системными компонентами

Используется для следующих задач:

- Загрузка \ Выгрузка данных в\из различные системы
- **Преобразования данных:**
  - Схлопывание версий

Что делает библиотека?

dwh_etl_tools – python библиотека с системными компонентами

Используется для следующих задач:

- Загрузка \ Выгрузка данных в\из различные системы
- **Преобразования данных:**
  - Схлопывание версий
  - Протягивание полей

Что делает библиотека?

dwh_etl_tools – python библиотека с системными компонентами

Используется для следующих задач:

- Загрузка \ Выгрузка данных в\из различные системы
- **Преобразования данных:**
  - Схлопывание версий
  - Протягивание полей
  - Правка версионности

Что делает библиотека?

dwh_etl_tools – python библиотека с системными компонентами

Используется для следующих задач:

- Загрузка \ Выгрузка данных в\из различные системы
- **Преобразования данных:**
  - Схлопывание версий
  - Протягивание полей
  - Правка версионности
  - И т.д.

Версионность

Возможность хранения в БД истории всех изменений каждого объекта

id	city_name	valid_from	valid_to
1	Нижний Новгород	1221	1932
1	Горький	1932	1990
1	Нижний Новгород	1990	5999-01-01
2	Самара	1586	1935
2	Куйбышев	1935	1991
2	Самара	1991	5999-01-01
3	Царицын	1589	1925
3	Сталинград	1925	1961
3	Волгоград	1961	5999-01-01

Версионность

Возможность хранения в БД истории всех изменений каждого объекта

id	city_name	valid_from	valid_to
1	Нижний Новгород	1221-08-21	1932-10-06
1	Горький	1932-10-07	1990-10-21
1	Нижний Новгород	1990-10-22	5999-01-01

- **valid_from – дата начала действия версии**
- valid_to – дата окончания действия версии
- 5999-01-01 – пример метки актуальной версии(дата из далекого будущего)
- valid_from текущей версии должен быть больше на 1 секунду или 1 день, чем valid_to предыдущей версии

Версионность

Возможность хранения в БД истории всех изменений каждого объекта

id	city_name	valid_from	valid_to
1	Нижний Новгород	1221-08-21	1932-10-06
1	Горький	1932-10-07	1990-10-21
1	Нижний Новгород	1990-10-22	5999-01-01

- valid_from – дата начала действия версии
- **valid_to – дата окончания действия версии**
- 5999-01-01 – пример метки актуальной версии(дата из далекого будущего)
- valid_from текущей версии должен быть больше на 1 секунду или 1 день, чем valid_to предыдущей версии

Версионность

Возможность хранения в БД истории всех изменений каждого объекта

id	city_name	valid_from	valid_to
1	Нижний Новгород	1221-08-21	1932-10-06
1	Горький	1932-10-07	1990-10-21
1	Нижний Новгород	1990-10-22	5999-01-01

- valid_from – дата начала действия версии
- valid_to – дата окончания действия версии
- **5999-01-01 – пример метки актуальной версии(дата из далекого будущего)**
- valid_from текущей версии должен быть больше на 1 секунду или 1 день, чем valid_to предыдущей версии

Версионность

Возможность хранения в БД истории всех изменений каждого объекта

id	city_name	valid_from	valid_to
1	Нижний Новгород	1221-08-21	1932-10-**06**
1	Горький	1932-10-**07**	1990-10-21
1	Нижний Новгород	1990-10-22	5999-01-01

- valid_from – дата начала действия версии
- valid_to – дата окончания действия версии
- 5999-01-01 – пример метки актуальной версии(дата из далекого будущего)
- **valid_from текущей версии должен быть больше на 1 секунду или 1 день, чем valid_to предыдущей версии**

Версионность

Возможность хранения в БД истории всех изменений каждого объекта

id	city_name	valid_from	valid_to
1	Нижний Новгород	1221-08-21	1932-10-06
1	Горький	1932-10-07	1990-10-**21**
1	Нижний Новгород	1990-10-**22**	5999-01-01

- valid_from – дата начала действия версии
- valid_to – дата окончания действия версии
- 5999-01-01 – пример метки актуальной версии(дата из далекого будущего)
- **valid_from текущей версии должен быть больше на 1 секунду или 1 день, чем valid_to предыдущей версии**

Объединение нескольких версий с одинаковым набором полей в одну

Пример

Уже знакомая таблица истории переименования городов

id	city_name	valid_from	valid_to
1	Нижний Новгород	1221	1932
1	Горький	1932	1990
1	Нижний Новгород	1990	5999-01-01
2	Самара	1586	1935
2	Куйбышев	1935	1991
2	Самара	1991	5999-01-01
3	Царицын	1589	1925
3	Сталинград	1925	1961
3	Волгоград	1961	5999-01-01

Объединение нескольких версий с одинаковым набором полей в одну

До схлопывания

id	city_name	valid_from	valid_to
1	Нижний Новгород	1221	1729
1	Нижний Новгород	1729	1903
1	Нижний Новгород	1903	1932
1	Горький	1932	1990
1	Нижний Новгород	1990	5999-01-01
2	Самара	1586	1935
2	Куйбышев	1935	1991
2	Самара	1991	5999-01-01
3	Царицын	1589	1925
3	Сталинград	1925	1961
3	Волгоград	1961	5999-01-01

Объединение нескольких версий с одинаковым набором полей в одну

До схлопывания

id	city_name	valid_from	valid_to
1	Нижний Новгород	1221	1729
1	Нижний Новгород	1729	1903
1	Нижний Новгород	1903	1932
1	Горький	1932	1990
1	Нижний Новгород	1990	5999-01-01
2	Самара	1586	1935
2	Куйбышев	1935	1991
2	Самара	1991	5999-01-01
3	Царицын	1589	1925
3	Сталинград	1925	1961
3	Волгоград	1961	5999-01-01

После схлопывания

id	city_name	valid_from	valid_to
1	Нижний Новгород	1221	1932
1	Горький	1932	1990
1	Нижний Новгород	1990	5999-01-01
2	Самара	1586	1935
2	Куйбышев	1935	1991
2	Самара	1991	5999-01-01
3	Царицын	1589	1925
3	Сталинград	1925	1961
3	Волгоград	1961	5999-01-01

Extract & Load data


ORACLE


cassandra


PostgreSQL


ClickHouse


kafka


hadoop


Greenplum

Это всё же ещё
и тестировать надо?

Т БАНК

Но баги случаются...

Специфика нашей работы

01

Высокая цена ошибок

Поскольку
мы делаем системные
компоненты,
то ошибки могут
привести к потере
данных в хранилище

02

Техническая сложность

Необходимо
учитывать специфику
систем, вести базу
знаний, применять
проверенные практики
к другим системам

03

Не получится мокировать

Из-за высокой
цены ошибки
приходится работать
с реальными системами
и тестировать работу
на них

04

Автоматические проверки

Всё тестирование
в команде
автоматизировано

05

Общий паттерн

При добавлении
новой компоненты в
библиотеку разработка
и тестирование идут
по наработанной ранее
схеме

Специфика нашей работы

01

Высокая цена ошибок

Поскольку мы делаем системные компоненты, то ошибки могут привести к потере данных в хранилище

02

Техническая сложность

Необходимо учитывать специфику систем, вести базу знаний, применять проверенные практики к другим системам

03

Не получится мокировать

Из-за высокой цены ошибки приходится работать с реальными системами и тестировать работу на них

04

Автоматические проверки

Всё тестирование в команде автоматизировано

05

Общий паттерн

При добавлении новой компоненты в библиотеку разработка и тестирование идут по наработанной ранее схеме

Специфика нашей работы

01

Высокая цена ошибок

Поскольку мы делаем системные компоненты, то ошибки могут привести к потере данных в хранилище

02

Техническая сложность

Необходимо учитывать специфику систем, вести базу знаний, применять проверенные практики к другим системам

03

Не получится мокировать

Из-за высокой цены ошибки приходится работать с реальными системами и тестировать работу на них

04

Автоматические проверки

Всё тестирование в команде автоматизировано

05

Общий паттерн

При добавлении новой компоненты в библиотеку разработка и тестирование идут по наработанной ранее схеме

Специфика нашей работы

01

Высокая цена ошибок

Поскольку мы делаем системные компоненты, то ошибки могут привести к потере данных в хранилище

02

Техническая сложность

Необходимо учитывать специфику систем, вести базу знаний, применять проверенные практики к другим системам

03

Не получится мокировать

Из-за высокой цены ошибки приходится работать с реальными системами и тестировать работу на них

04

Автоматические проверки

Всё тестирование в команде автоматизировано

05

Общий паттерн

При добавлении новой компоненты в библиотеку разработка и тестирование идут по наработанной ранее схеме

Специфика нашей работы

01

Высокая цена ошибок

Поскольку мы делаем системные компоненты, то ошибки могут привести к потере данных в хранилище

02

Техническая сложность

Необходимо учитывать специфику систем, вести базу знаний, применять проверенные практики к другим системам

03

Не получится мокировать

Из-за высокой цены ошибки приходится работать с реальными системами и тестировать работу на них

04

Автоматические проверки

Всё тестирование в команде автоматизировано

05

Общий паттерн

При добавлении новой компоненты в библиотеку разработка и тестирование идут по наработанной ранее схеме

Реализация тестирования

Т БАНК

Как мы выгрузку из Greenplum в Kafka тестировали

Е.Фролова, Е.Карандасов (Heisenbug autumn 2023)

Про что я тут расскажу?

Подход к тестам

Как выработанный подход
к написанию тестов помог нам
тестировать самые разные системы

Про что я тут расскажу?

Подход к тестам

Как выработанный подход
к написанию тестов помог нам
тестировать самые разные системы

Тестовое окружение

Какие объекты тестового окружения
обязательно присутствуют в тестах
каждой системной компоненты

Про что я тут расскажу?

Подход к тестам

Как выработанный подход
к написанию тестов помог нам
тестировать самые разные системы

Тестовое окружение

Какие объекты тестового окружения
обязательно присутствуют в тестах
каждой системной компоненты

Как происходит сравнение

Как происходит сравнение
ожидаемого и фактического
результатов

А на чём тесты?

Python

Как сама библиотека dwh_etl_tools, так и тесты для неё написаны на языке python

А на чём тесты?

Python

Как сама библиотека dwh_etl_tools, так и тесты для неё написаны на языке python

Pytest

Все тесты написаны в тестовом фреймворке pytest + используем плагины для него

А на чём тесты?

Python

Как сама библиотека dwh_etl_tools, так и тесты для неё написаны на языке python

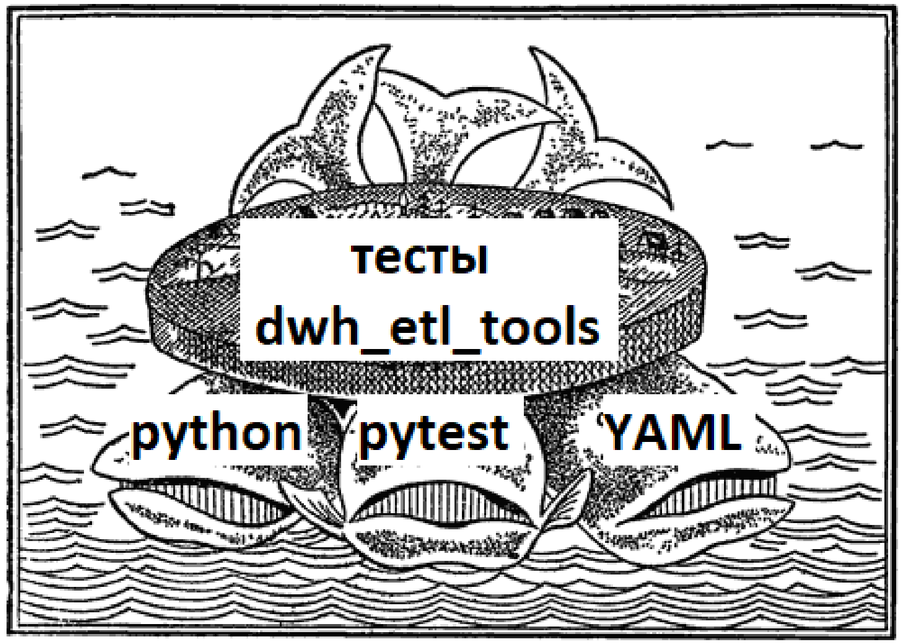
Pytest

Все тесты написаны в тестовом фреймворке pytest + используем плагины для него

YAML

Все тест-кейсы хранятся в виде объектов в файлах YAML

А на чём тесты?



Python

Как сама библиотека dwh_etl_tools, так и тесты для неё написаны на языке python

Pytest

Все тесты написаны в тестовом фреймворке pytest + используем плагины для него

YAML

Все тест-кейсы хранятся в виде объектов в файлах YAML

Пример фикстуры

Инициализировали коннект

```python
@pytest.fixture
def gpconn_tests(gp_conn_str_tests) -> Generator[GreenplumSQLFactory, None, None]:
    """“Коннект к Greenplum для создания и заполнения таблиц"""
    gpconn = GreenplumSQLFactory.create_connection(conn_str=gp_conn_str_tests)
    yield gpconn
    gpconn.close()
```

Пример фикстуры

Передали в тест

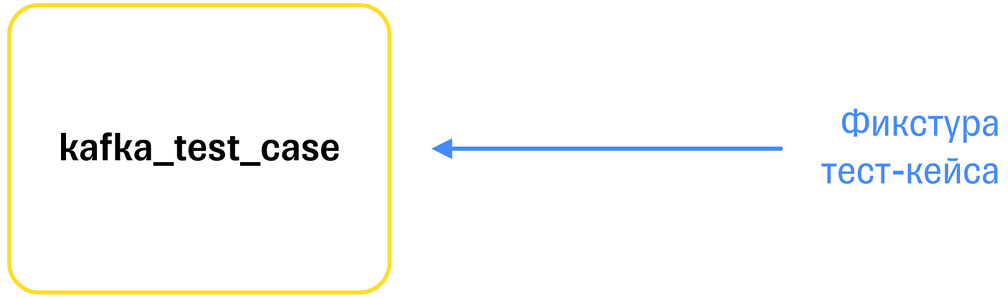
```python
@pytest.fixture
def gpconn_tests(gp_conn_str_tests) -> Generator[GreenplumSQLFactory, None, None]:
    """“Коннект к Greenplum для создания и заполнения таблиц"""
    gpconn = GreenplumSQLFactory.create_connection(conn_str=gp_conn_str_tests)
    yield gpconn
    gpconn.close()
```

Пример фикстуры

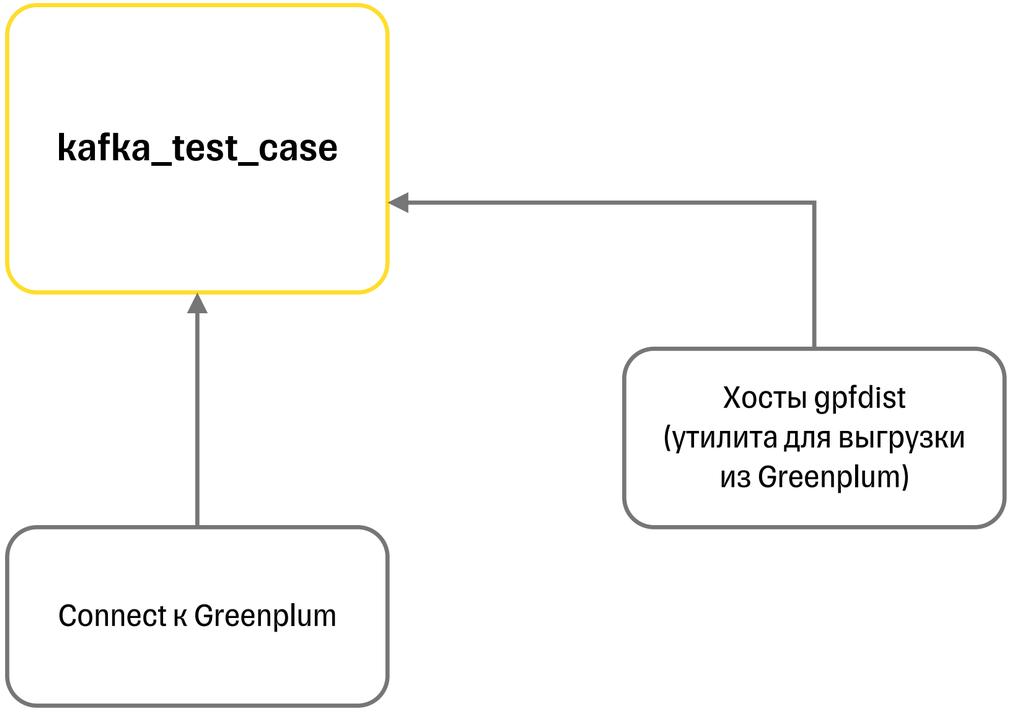
Закрыли коннект

```python
@pytest.fixture
def gpconn_tests(gp_conn_str_tests) -> Generator[GreenplumSQLFactory, None, None]:
    """“Коннект к Greenplum для создания и заполнения таблиц"""
    gpconn = GreenplumSQLFactory.create_connection(conn_str=gp_conn_str_tests)
    yield gpconn
    gpconn.close()
```

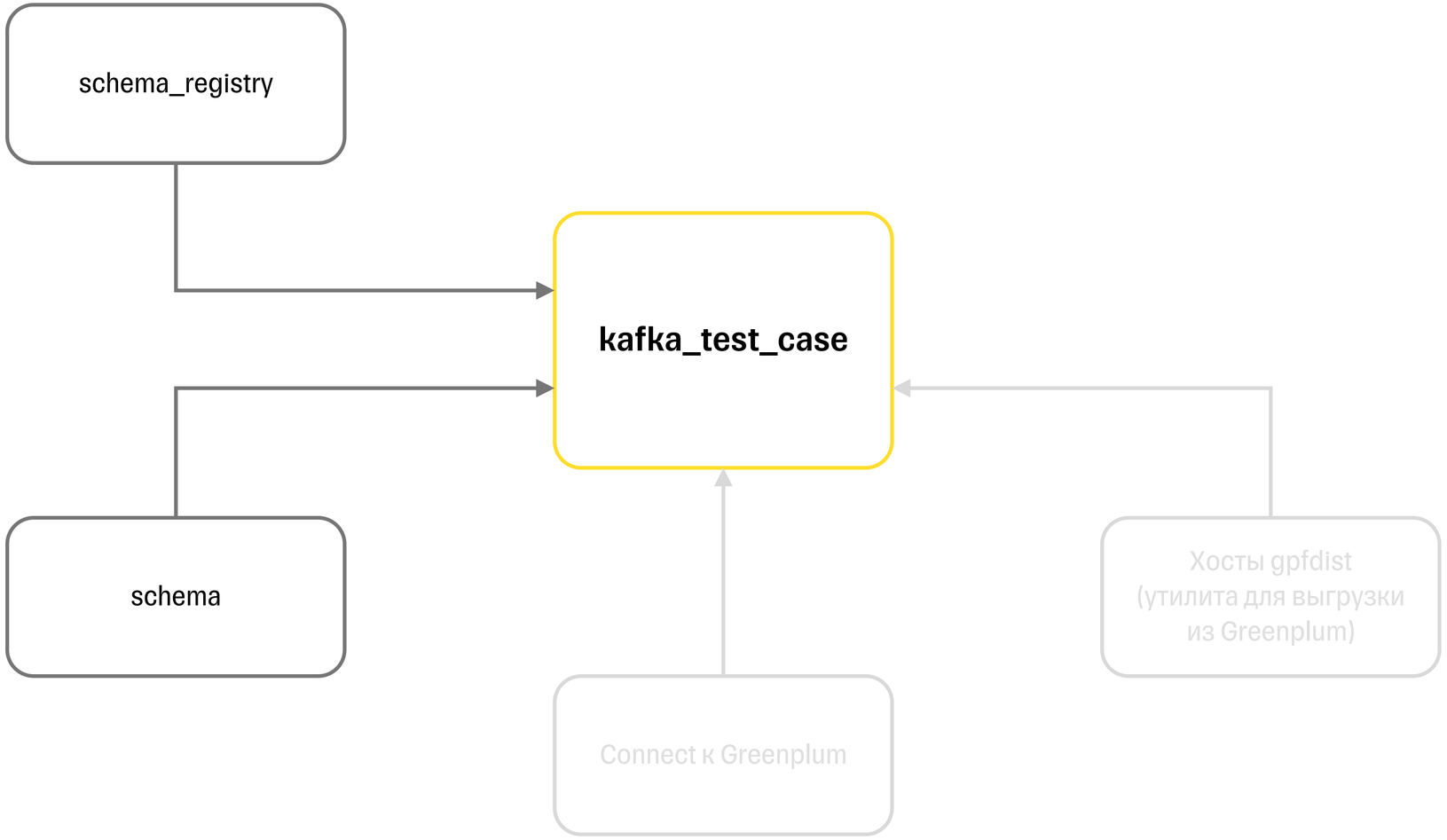
Наборы фикстур



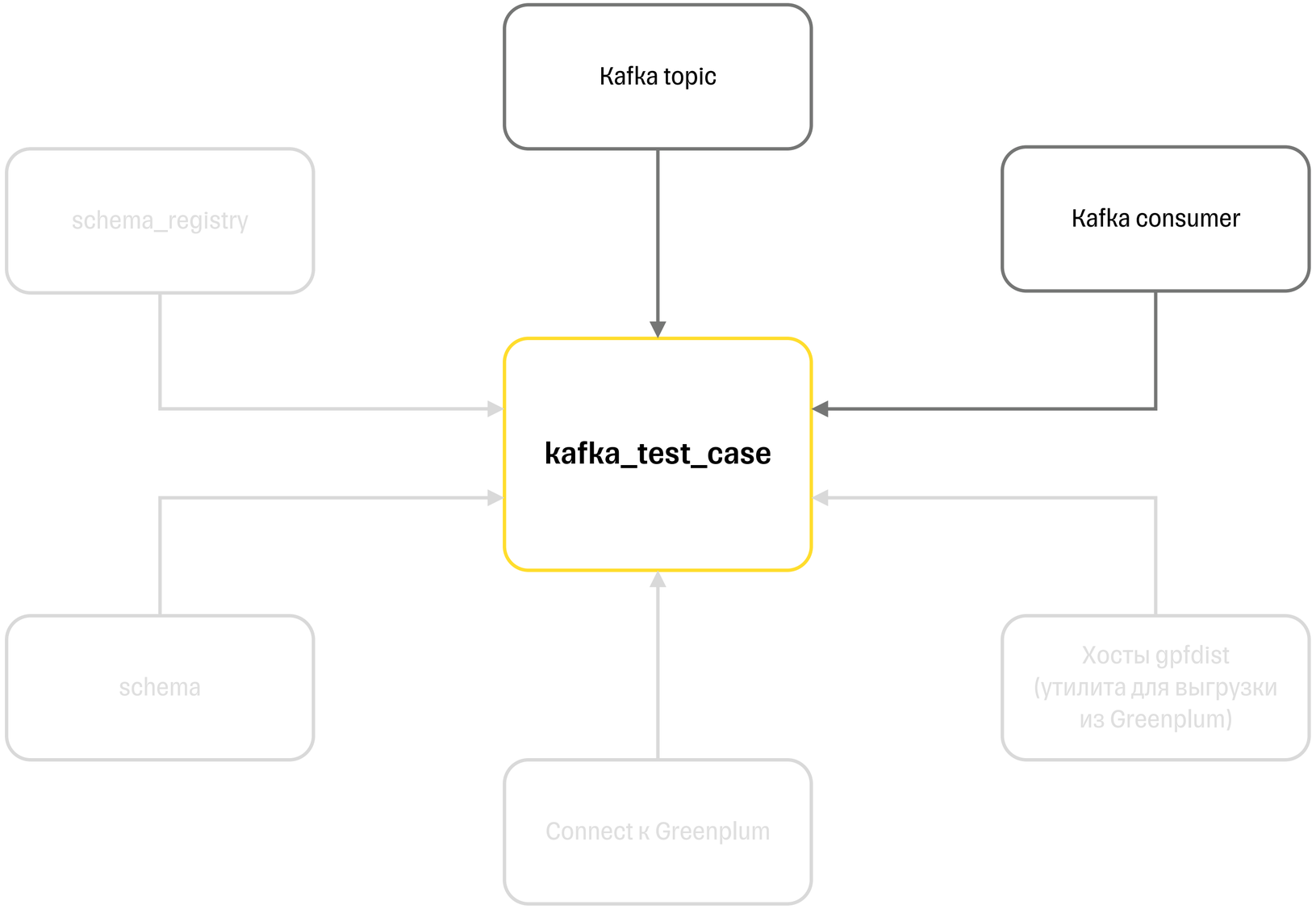
Наборы фикстур



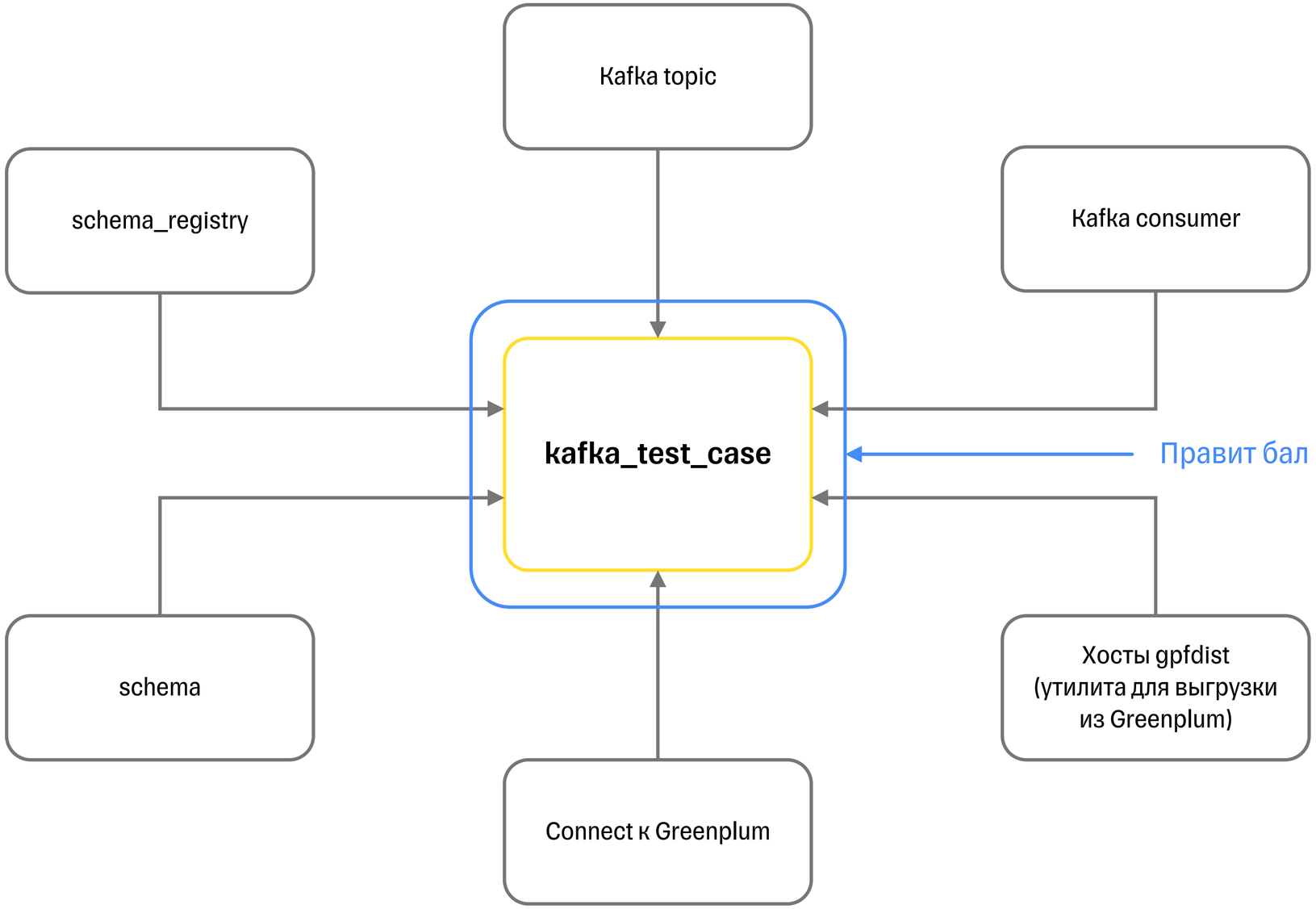
Наборы фикстур



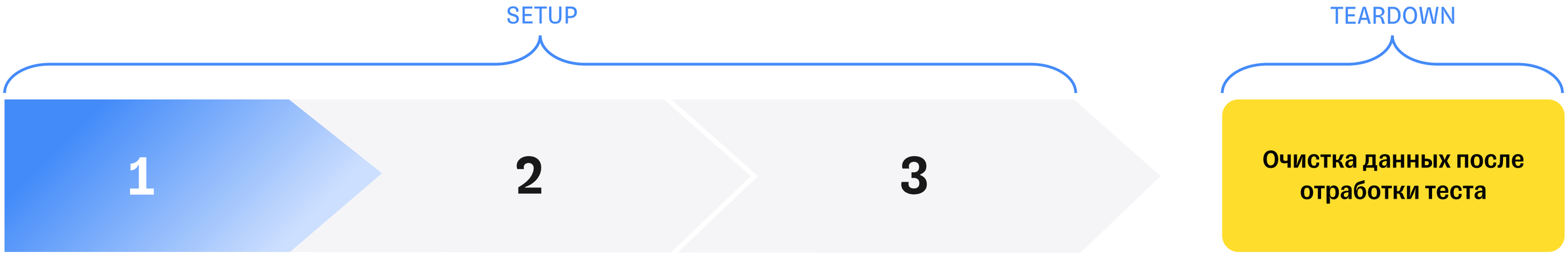
Наборы фикстур



Наборы фикстур



Фикстура тест-кейса



Prepare инстанс

Специальный подготовительный класс, в котором подготавливается окружение и реализованы проверки

Параметры запуска

Подготовка всех необходимых параметров загрузчика для конкретного тест-кейса

Формирование AVRO схемы

Формирование тест-кейса

DDL (Data Definition Language) — команды, используемые для создания и изменения структуры объектов БД.

Пример: **CREATE**, **ALTER**, **DROP**

DML (Data Manipulation Language) — группа операторов, позволяющих изменять данные

Пример: **INSERT**, **DELETE**, **UPDATE**

ddl.yaml

dml.yaml

test_cases.yaml

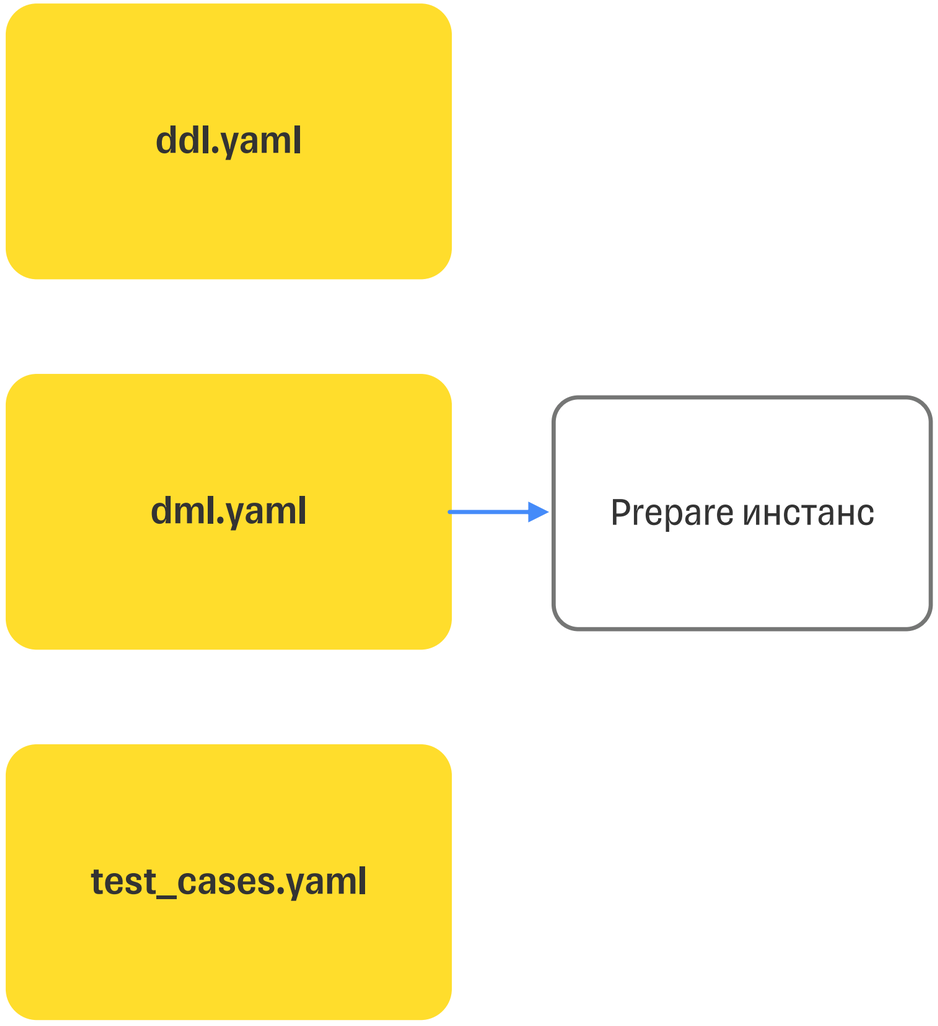
Формирование тест-кейса

DDL (Data Definition Language) — команды, используемые для создания и изменения структуры объектов БД.

Пример: **CREATE**, **ALTER**, **DROP**

DML (Data Manipulation Language) — группа операторов, позволяющих изменять данные

Пример: **INSERT**, **DELETE**, **UPDATE**



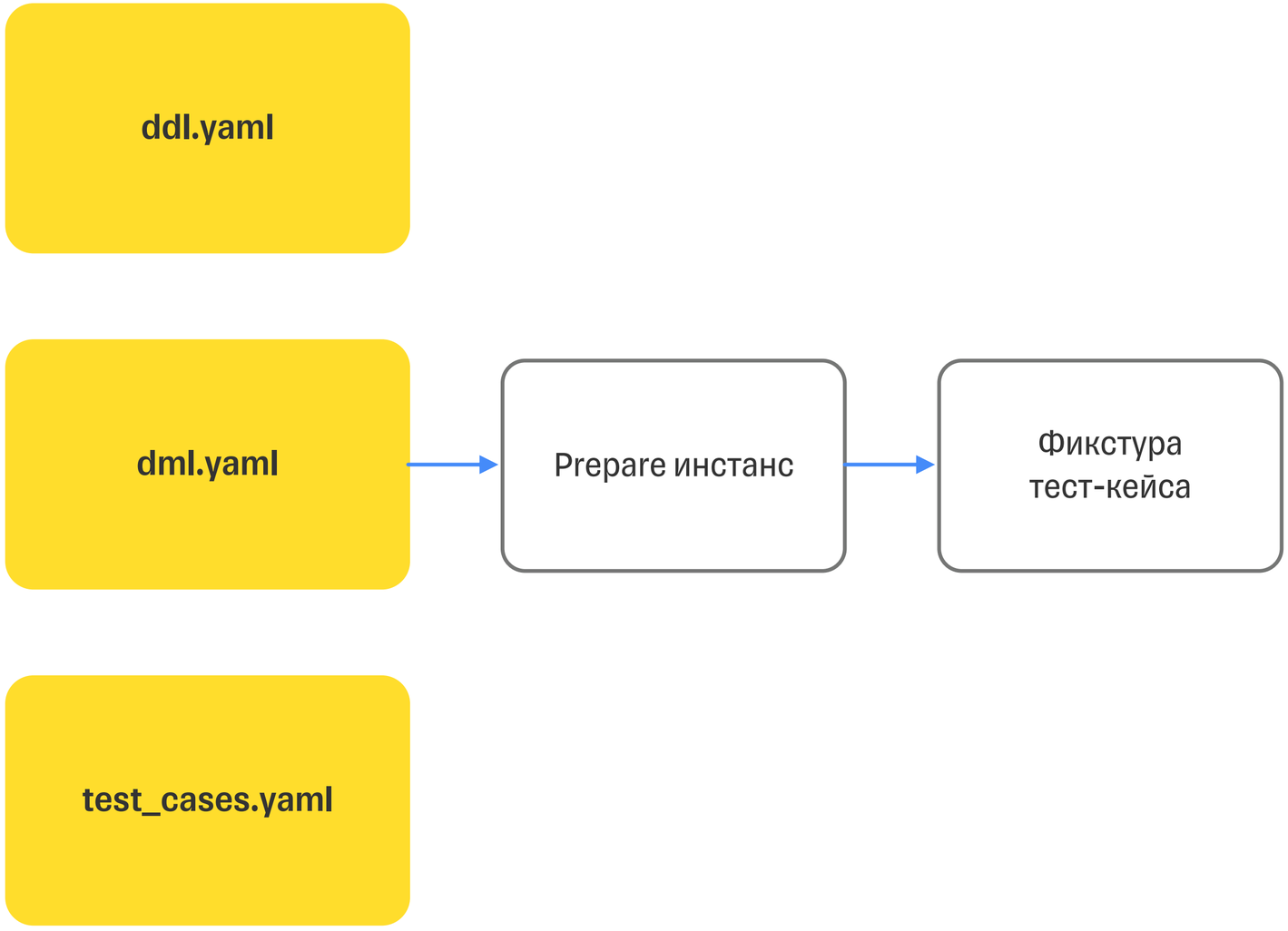
Формирование тест-кейса

DDL (Data Definition Language) — команды, используемые для создания и изменения структуры объектов БД.

Пример: **CREATE**, **ALTER**, **DROP**

DML (Data Manipulation Language) — группа операторов, позволяющих изменять данные

Пример: **INSERT**, **DELETE**, **UPDATE**



Тестовые данные

ddl.yaml

dml.yaml

test_cases.yaml

- Все в YAML
- На каждую тестируемую систему 3 файла
- В случае с kafka еще понадобилась
  папка scheme с avro-схемами

Тестовые данные

ddl.yaml

dml.yaml

test_cases.yaml

Содержит **описания структуры** следующих объектов:

- **Источник**
- Фактический результат
- Ожидаемый результат

```
"src_base": &src_base
    description: описание объекта
    schema_nm: test_kafka
    table_nm: src_{test_id}
    col_flg: k2 ->key_1, fs, fi, fdt, fa, fd
    dist_key: key_1
    create_view: False
```

Тестовые данные

ddl.yaml

dml.yaml

test_cases.yaml

Содержит **описания структуры** следующих объектов:

- Источник
- **Фактический результат**
- Ожидаемый результат

```
"tgt_base": &tgt_base
    description: описание объекта
    schema_nm: test_kafka
    table_nm: tgt_{test_id}
    col_flg: k2 ->key_1, fs, fi, fdt, fa, fd
    dist_key: key_1
```

Тестовые данные

ddl.yaml

dml.yaml

test_cases.yaml

Содержит **описания структуры** следующих объектов:

- Источник
- Фактический результат
- **Ожидаемый результат**

```
"tgt_er_base": &tgt_er_base
    description: описание объекта
    schema_nm: test_kafka
    table_nm: tgt_er_{test_id}
    col_flg: k2 ->key_1, fs, fi, fdt, fa, fd
    dist_key: key_1
```

Тестовые данные

ddl.yaml

dml.yaml

test_cases.yaml

Рассмотрим DML скрипты на примере:

- **Таргет – уже есть 1 строка**
- Источник – пришла новая строка
- Ожидание – 2 строки в таргете (существующая + новая)

```
“tgt->long_row": >
 INSERT INTO {table} VALUES
  ('TST_1' , ‘a',  1, '2024-05-31');
```

Тестовые данные

ddl.yaml

dml.yaml

test_cases.yaml

Рассмотрим DML скрипты на примере:

- Таргет – уже есть 1 строка
- **Источник – пришла новая строка**
- Ожидание – 2 строки в таргете (существующая + новая)

```
"src->long_row": >
  INSERT INTO {table} VALUES
   ('TST_2' , repeat('b', 40000),  2, '2017-01-01');
```

Тестовые данные

ddl.yaml

dml.yaml

test_cases.yaml

Рассмотрим DML скрипты на примере:

- Таргет – уже есть 1 строка
- Источник – пришла новая строка
- **Ожидание – 2 строки в таргете (существующая + новая)**

```
“tgt_er->long_row": >
 INSERT INTO {table} VALUES
  ('TST_1' , 'a',  1, '2024-05-31'),
  ('TST_2' , repeat('b', 40000),  2, '2017-01-01');
```

Тестовые данные

ddl.yaml

dml.yaml

test_cases.yaml

Содержит описания тест-кейсов, включая:

- DDL объектов

Тестовые данные

ddl.yaml

dml.yaml

test_cases.yaml

Содержит описания тест-кейсов, включая:

- DDL объектов
- DML объектов

Тестовые данные

ddl.yaml

dml.yaml

test_cases.yaml

Содержит описания тест-кейсов, включая:

- DDL объектов
- DML объектов
- Параметры запуска

Тестовые данные

ddl.yaml

dml.yaml

test_cases.yaml

Содержит описания тест-кейсов, включая:

- DDL объектов
- DML объектов
- Параметры запуска
- Исключение и текст ошибки

Тестовые данные

ddl.yaml

dml.yaml

test_cases.yaml

Содержит описания тест-кейсов, включая:

- DDL объектов
- DML объектов
- Параметры запуска
- Исключение и текст ошибки
- Описание feature и story

Тестовые данные

ddl.yaml

dml.yaml

test_cases.yaml

Ссылки на DDL объекты

```
"long_row":
    description: "длинная строка в источнике«
    ddl_in: [ "src_base" ]
    ddl_out: [ "tgt_base" ]
    ddl_out_er: [ "tgt_er_base" ]

    dml_in: [ "src->long_row" ]
    dml_out: [ "tgt->long_row" ]
    dml_out_er: [ "tgt_er->long_row" ]

    param1: value
    param2: value

    mark: [ external ]
```

Тестовые данные

ddl.yaml

dml.yaml

test_cases.yaml

Ссылки на DML объекты

```
"long_row":
    description: "длинная строка в источнике«
    ddl_in: [ "src_base" ]
    ddl_out: [ "tgt_base" ]
    ddl_out_er: [ "tgt_er_base" ]

    dml_in: [ "src->long_row" ]
    dml_out: [ "tgt->long_row" ]
    dml_out_er: [ "tgt_er->long_row" ]

    param1: value
    param2: value

    mark: [ external ]
```

Тестовые данные

ddl.yaml

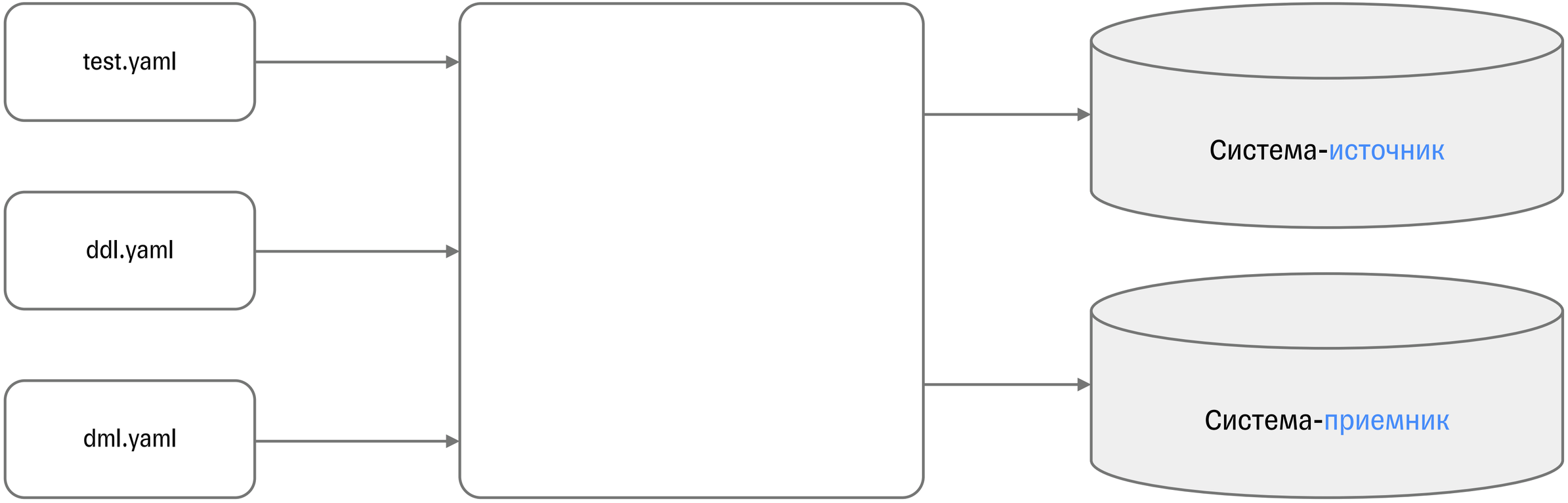
dml.yaml

test_cases.yaml

Параметры запуска

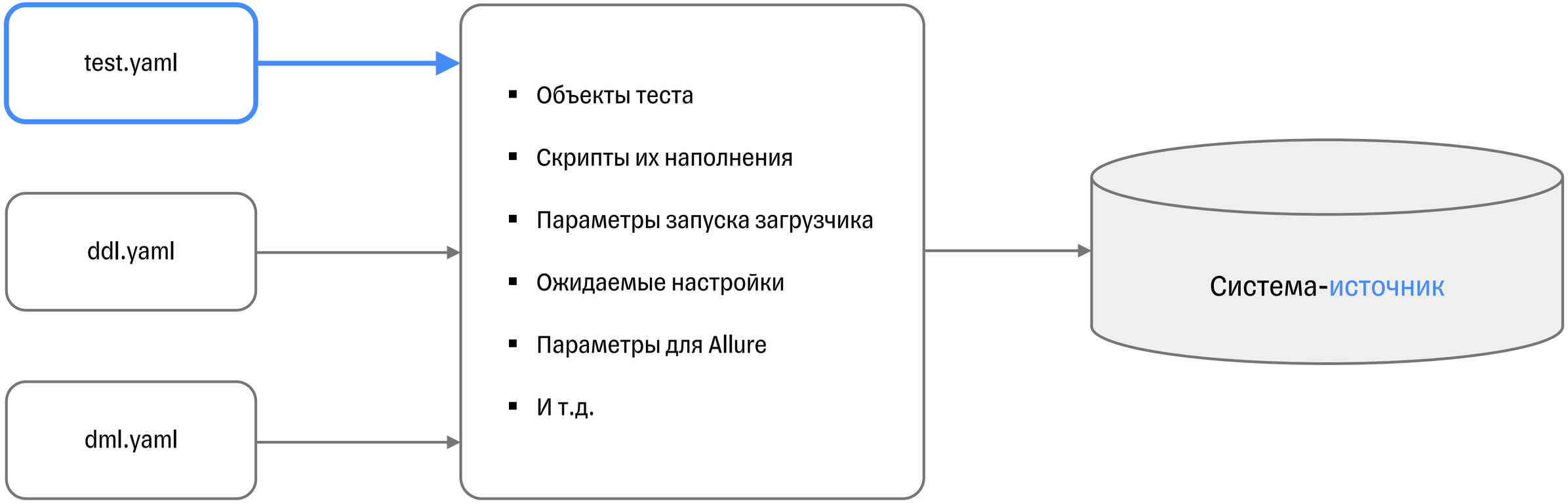
```
"long_row":
    description: "длинная строка в источнике«
    ddl_in: [ "src_base" ]
    ddl_out: [ "tgt_base" ]
    ddl_out_er: [ "tgt_er_base" ]

    dml_in: [ "src->long_row" ]
    dml_out: [ "tgt->long_row" ]
    dml_out_er: [ "tgt_er->long_row" ]

    param1: value
    param2: value

    mark: [ external ]
```

Prepare класс



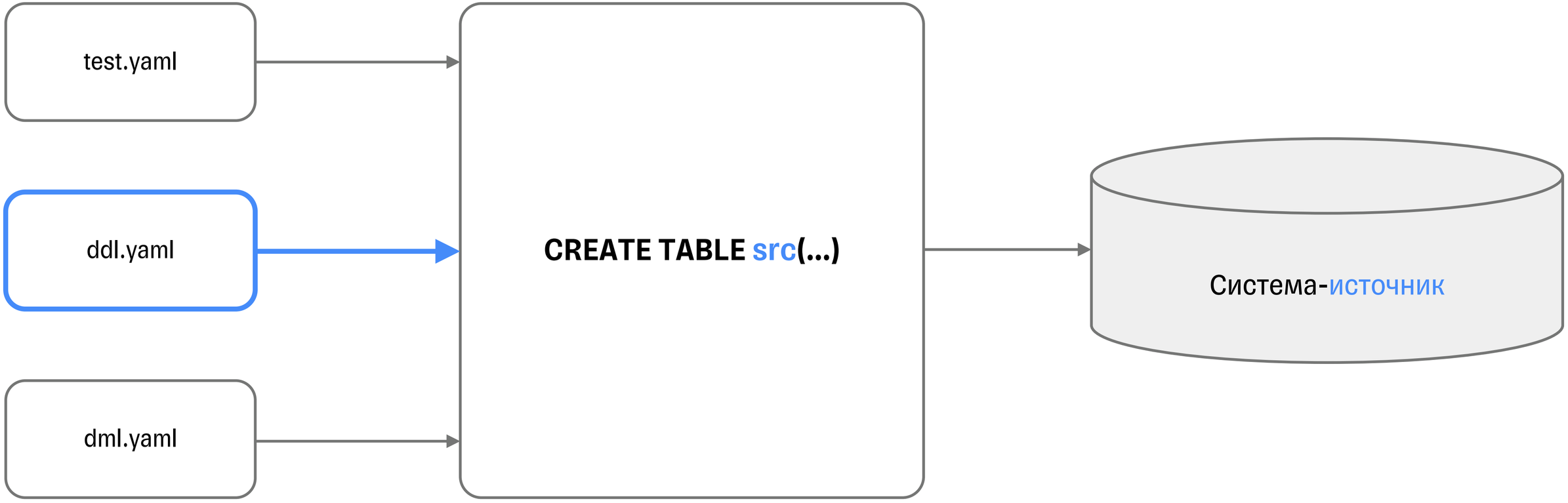
Prepare класс

Сбор информации о тесте



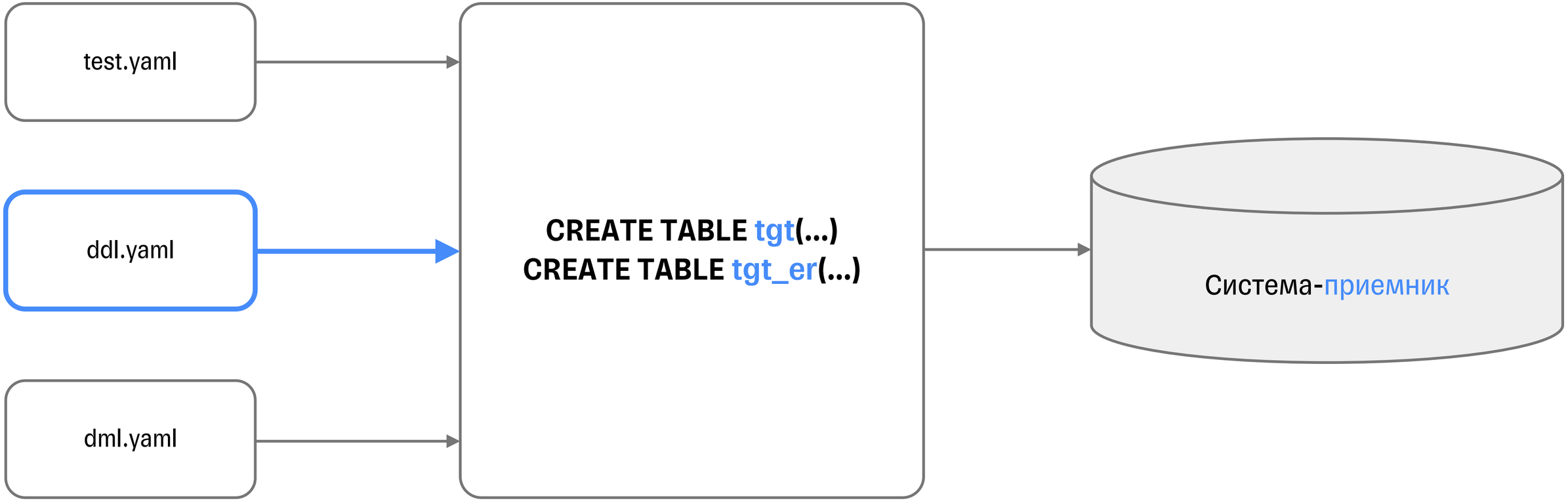
Prepare класс

Создание таблиц



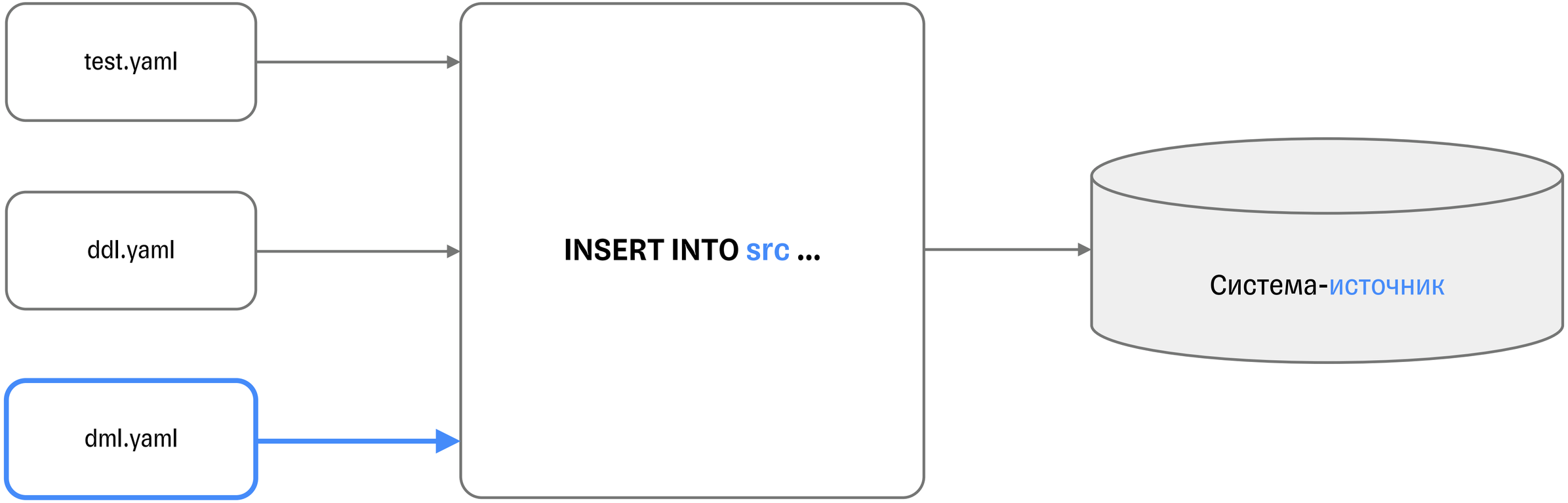
Prepare класс

Создание таблиц



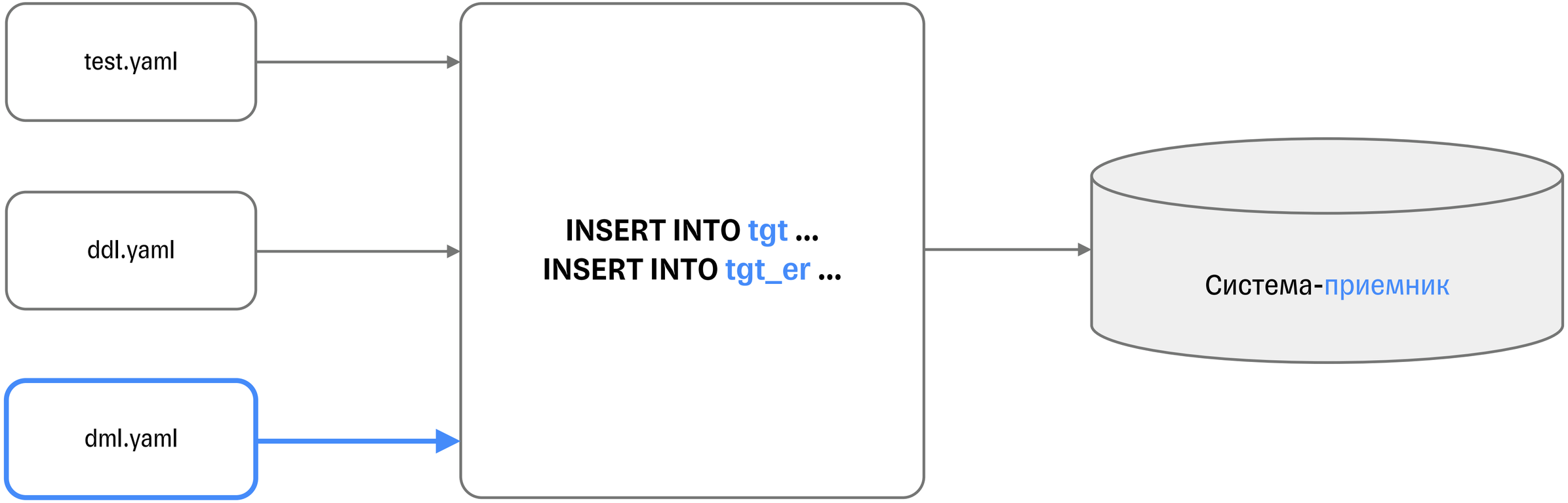
Prepare класс

Заполнение таблиц



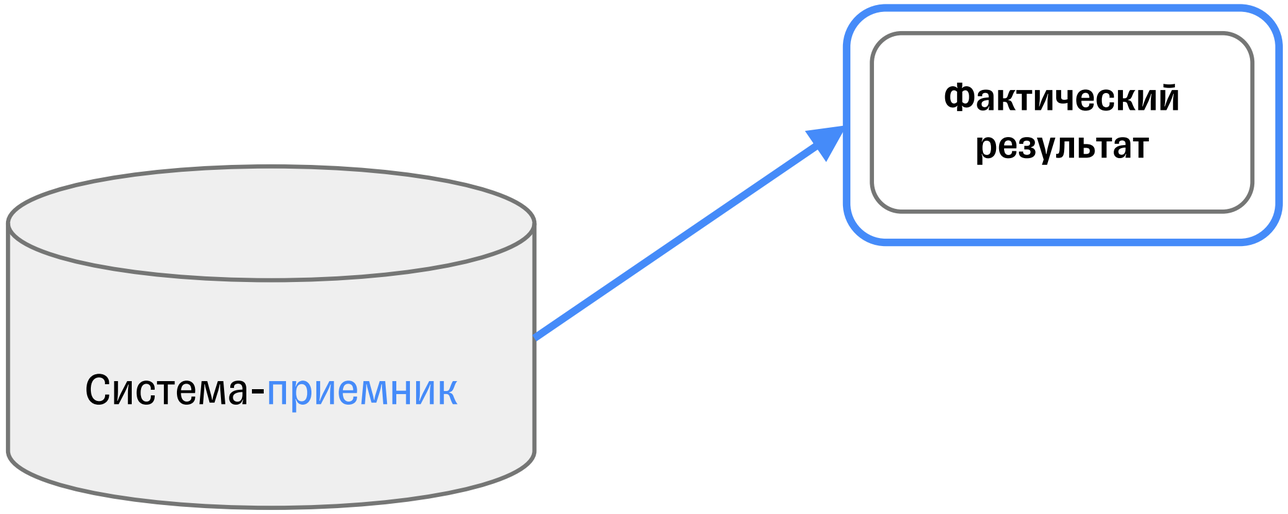
Prepare класс

Заполнение таблиц

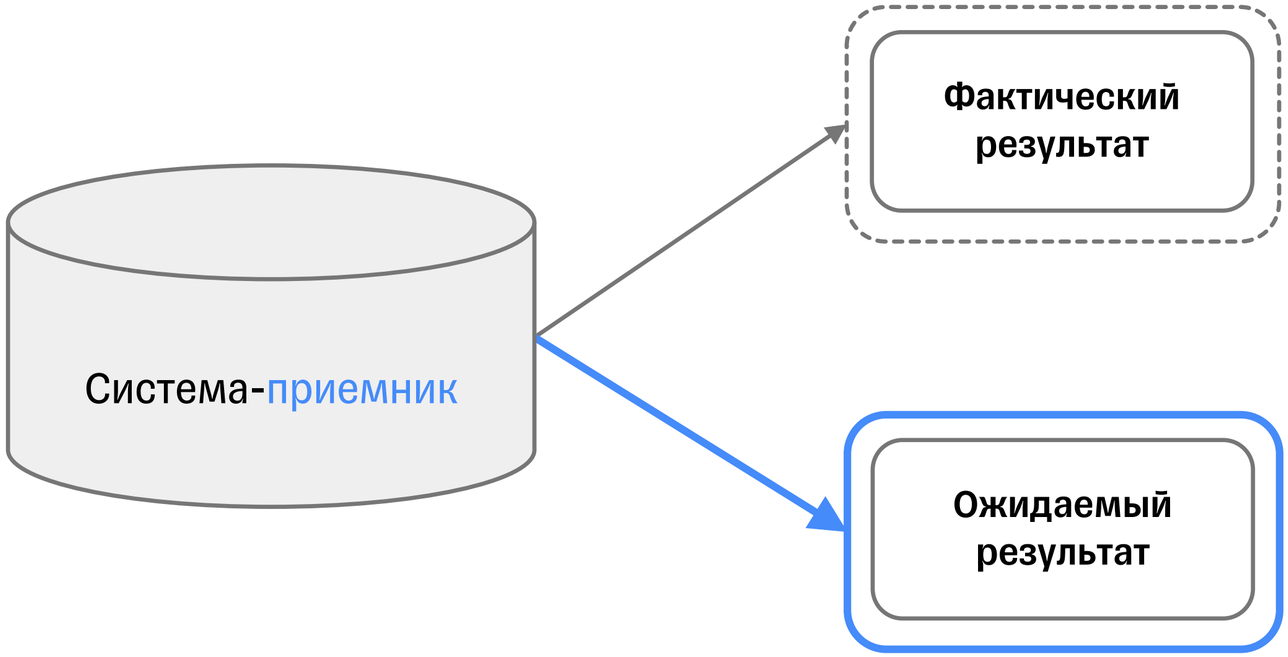


Prepare класс

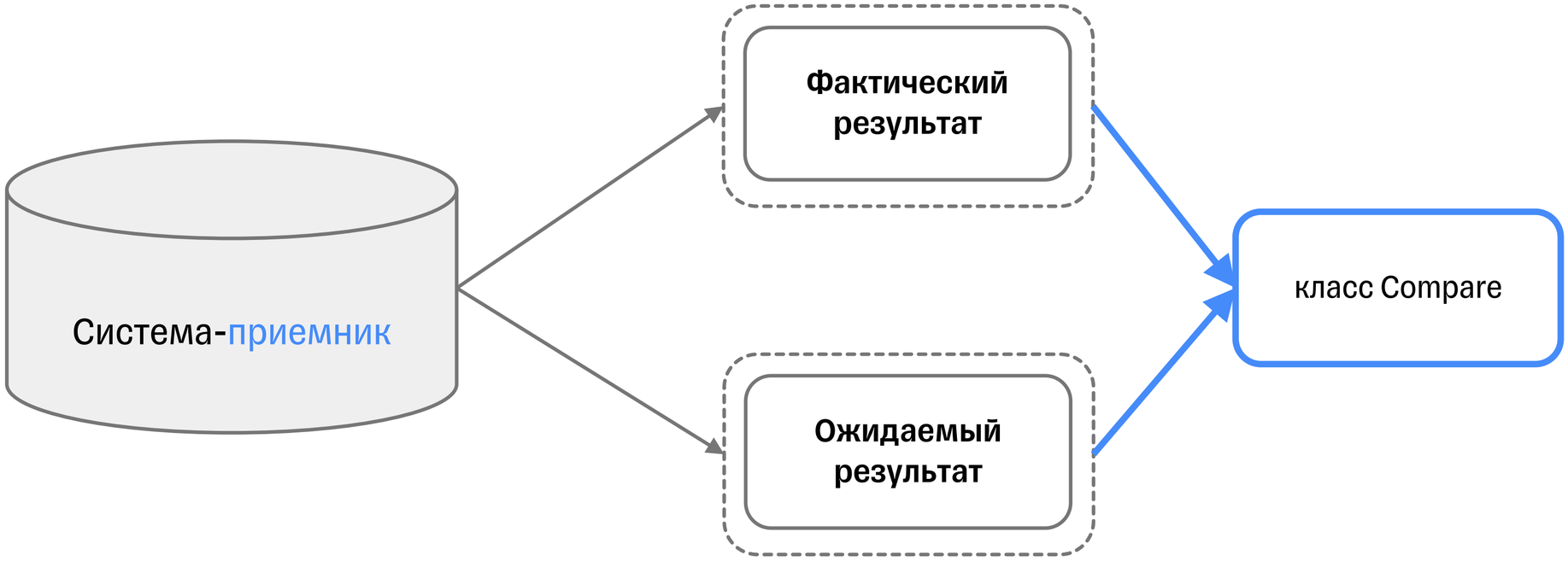
Сравнение



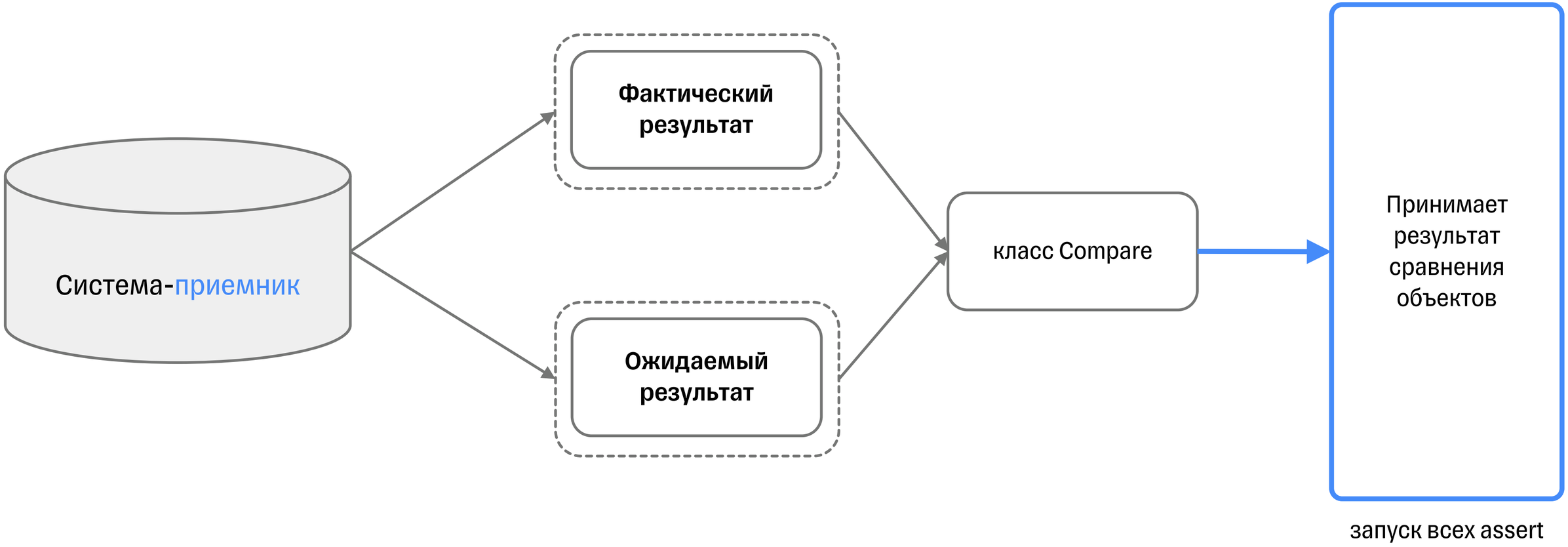
Сравнение


Фактический
результат
Система-приемник
Ожидаемый
результат

Сравнение


Фактический результат
Система-приемник
класс Compare
Ожидаемый результат

Сравнение


Система-приемник
Фактический результат
Ожидаемый результат
класс Compare
Принимает результат сравнения объектов
запуск всех assert

А что если ожидаемый результат и таргет – не таблицы?

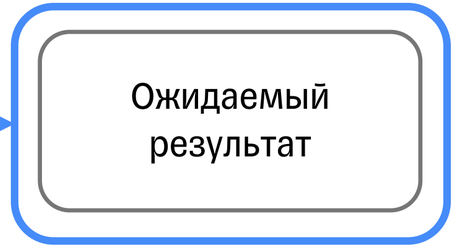
Генерация ожидаемого результата (вариант 1)

```
"tgt_er":
  titles: ['fld_dttm', 'fld_int', 'fld_str', 'key_1']
  messages:
    - ["2017-01-01 00:00:00", 1, "TST_wave_1", "TST_1"]
    - ...
    - ["2017-03-05 00:00:00", 10, "TST_wave_1", "TST_10"]
```

dml.yaml

Генерация ожидаемого результата (вариант 1)

```
"tgt_er":
  titles: ['fld_dttm', 'fld_int', 'fld_str', 'key_1']
  messages:
    - ["2017-01-01 00:00:00", 1, "TST_wave_1", "TST_1"]
    - ...
    - ["2017-03-05 00:00:00", 10, "TST_wave_1", "TST_10"]
```

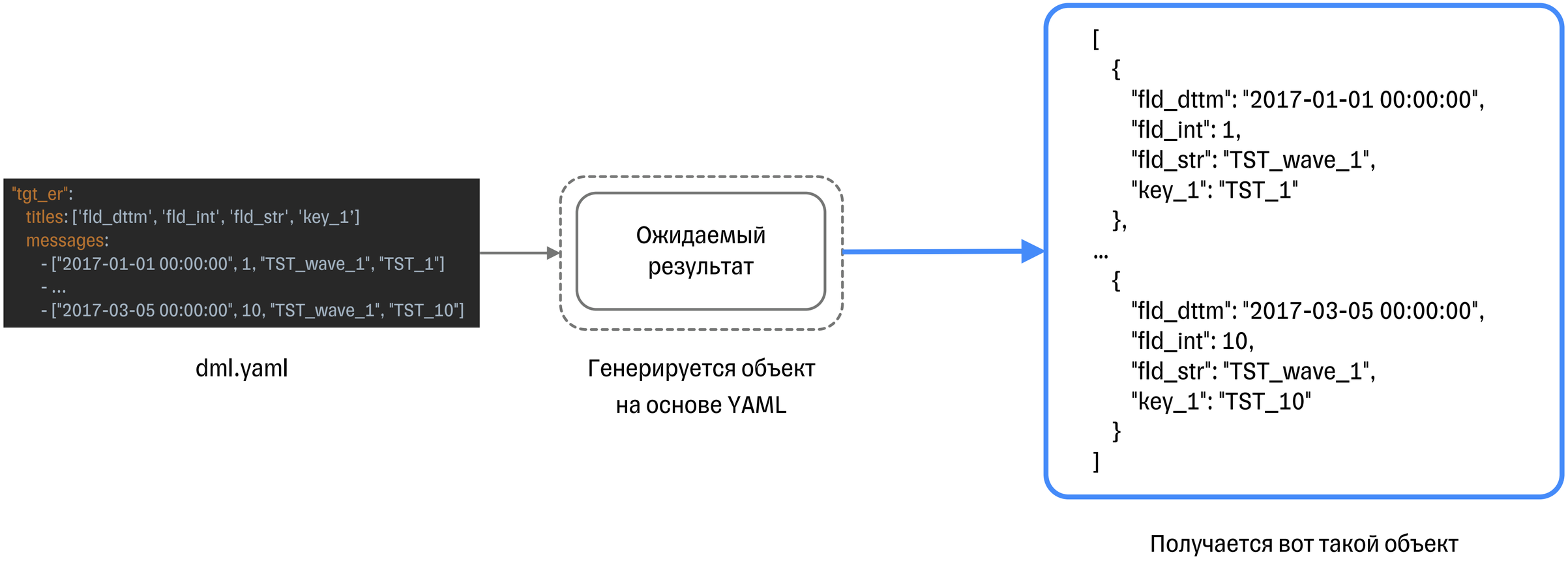


Ожидаемый результат

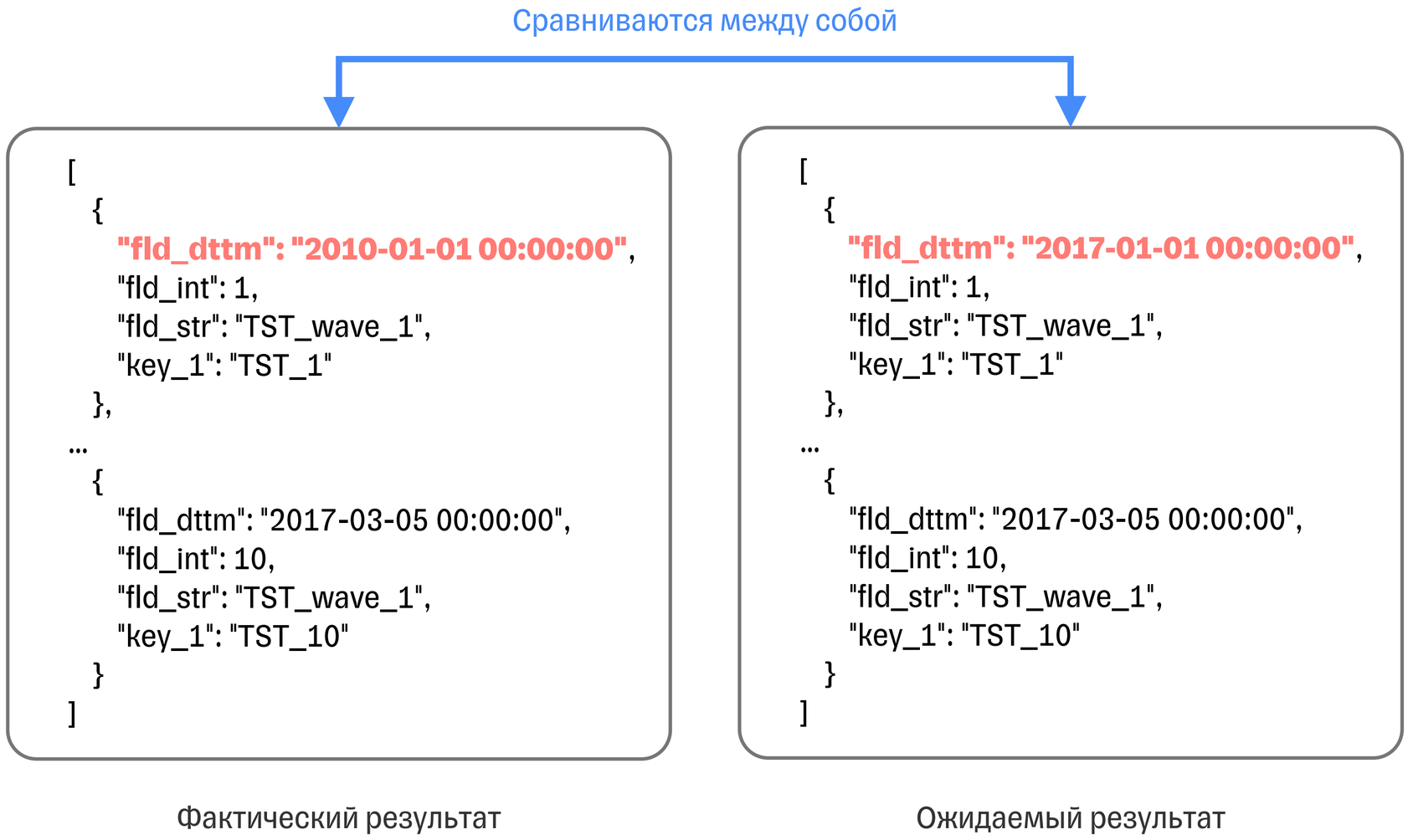
dml.yaml

Генерируется объект на основе YAML

Генерация ожидаемого результата (вариант 1)



Проверка результата (вариант 1)



Отображение расхождений

Строки с расхождениями

```
...
2024-09-13 14:29:42 - PrepareKafkaLoader - INFO    rows_only_in_actual = [{'key_1': 'TST_1', 'fld_str': 'TST_wave_1', 'fld_int': 1, 'fld_dttm': '2010-01-01 00:00:00'}]
2024-09-13 14:29:42 - PrepareKafkaLoader - INFO    rows_only_in_expected = [{'key_1': 'TST_1', 'fld_str': 'TST_wave_1', 'fld_int': 1, 'fld_dttm': '2017-01-01 00:00:00'}]
2024-09-13 14:29:42 - TestKafkaLoader - INFO    ======================== Check results ================================
2024-09-13 14:29:42 - TestKafkaLoader - INFO    count_rows_only_in_actual : 1
2024-09-13 14:29:42 - TestKafkaLoader - INFO    count_rows_only_in_expected : 1
2024-09-13 14:29:42 - TestKafkaLoader - INFO    count_rows_in_actual : 10
2024-09-13 14:29:42 - TestKafkaLoader - INFO    count_rows_in_expected : 10
2024-09-13 14:29:42 - TestKafkaLoader - INFO    field_in_actual : dict_keys(['key_1', 'fld_str', 'fld_int', 'fld_dttm'])
2024-09-13 14:29:42 - TestKafkaLoader - INFO    field_in_expected : dict_keys(['key_1', 'fld_str', 'fld_int', 'fld_dttm'])
...
2024-09-13 14:29:42 - TestKafkaLoader - INFO    ====================== Finish check results ============================
```

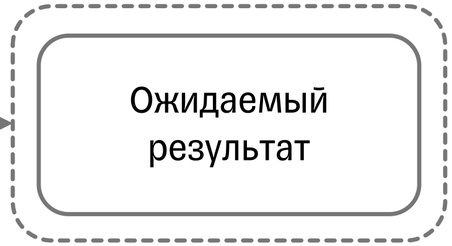
Отображение расхождений

...
2024-09-13 14:29:42 - PrepareKafkaLoader - INFO rows_only_in_actual = [{'key_1': 'TST_1', 'fld_str': 'TST_wave_1', 'fld_int': 1, 'fld_dttm': '2010-01-01 00:00:00'}]
2024-09-13 14:29:42 - PrepareKafkaLoader - INFO rows_only_in_expected = [{'key_1': 'TST_1', 'fld_str': 'TST_wave_1', 'fld_int': 1, 'fld_dttm': '2017-01-01 00:00:00'}]
2024-09-13 14:29:42 - TestKafkaLoader - INFO ========== Check results ==========
2024-09-13 14:29:42 - TestKafkaLoader - INFO count_rows_only_in_actual : 1
2024-09-13 14:29:42 - TestKafkaLoader - INFO count_rows_only_in_expected : 1
2024-09-13 14:29:42 - TestKafkaLoader - INFO count_rows_in_actual : 10
2024-09-13 14:29:42 - TestKafkaLoader - INFO count_rows_in_expected : 10
2024-09-13 14:29:42 - TestKafkaLoader - INFO field_in_actual : dict_keys(['key_1', 'fld_str', 'fld_int', 'fld_dttm'])
2024-09-13 14:29:42 - TestKafkaLoader - INFO field_in_expected : dict_keys(['key_1', 'fld_str', 'fld_int', 'fld_dttm'])
...
2024-09-13 14:29:42 - TestKafkaLoader - INFO ========== Finish check results ==========

Количество отличающихся строк!!!

Генерация ожидаемого результата (вариант 2)

```
"tgt_er":
  - [1, 1, "Value 1", ""]
  - [2, 2, "Value 2_1", “Value 2_2"]
  - [3, 3, "Value 3_1", “Full copy"]
```



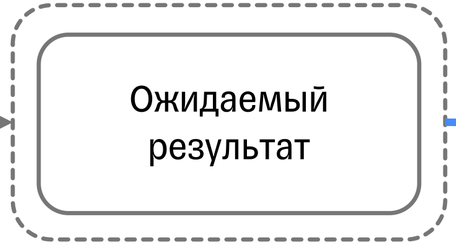
dml.yaml

Генерируется объект
на основе YAML

Генерация ожидаемого результата (вариант 2)

```
"tgt_er":
  - [1, 1, "Value 1", ""]
  - [2, 2, "Value 2_1", “Value 2_2"]
  - [3, 3, "Value 3_1", “Full copy"]
```

dml.yaml



Генерируется объект
на основе YAML

	key_1	fld_str_txt_1	fld_str_txt_2	fld_str_txt_3
0	1	1	Value 1	
1	2	2	Value 2_1	Value 2_2
2	3	3	Value 3_1	Full copy

Получается вот такой Dataframe

Проверка результата (вариант 2)

Сравниваются между собой

	key_1	fld_str_txt_1	fld_str_txt_2	fld_str_txt_3
0	1	1	Value 1	
1	3	3	Value3_1	Full copy
2	3	3	Value3_1	Full copy
3	2	2	Value 2_1	Value 2_2

Фактический результат

	key_1	fld_str_txt_1	fld_str_txt_2	fld_str_txt_3
0	1	1	Value 1	
1	2	2	Value 2_1	Value 2_2
2	3	3	Value 3_1	Full copy

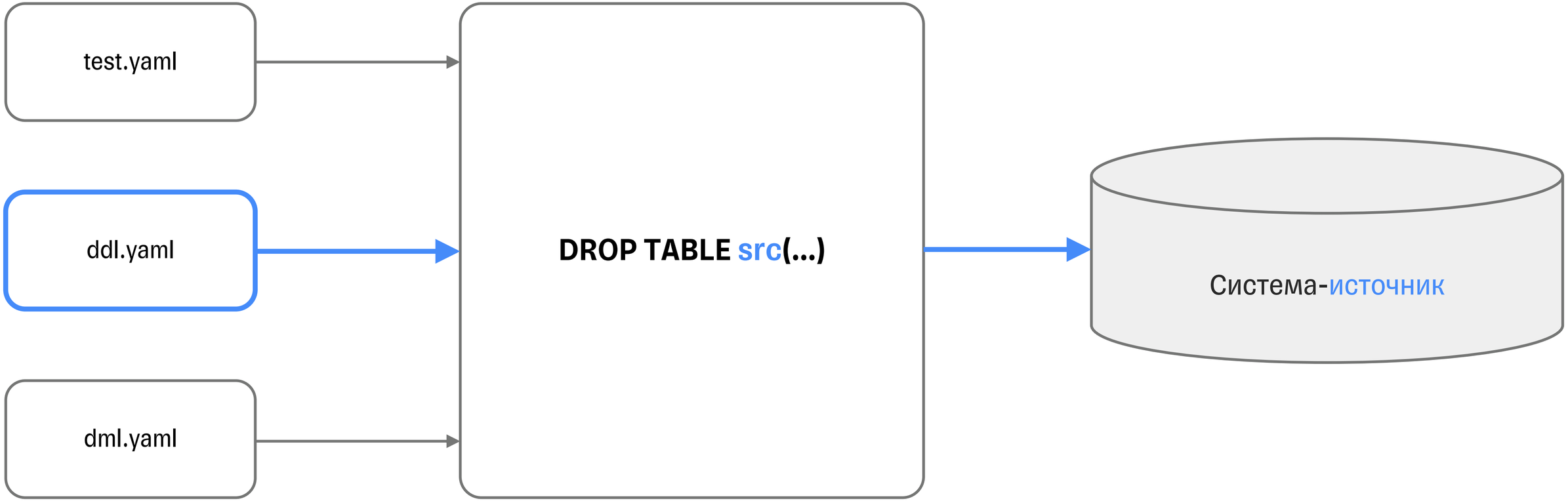
Ожидаемый результат

Отображение расхождений

```
2024-09-13 14:03:44 - PrepareS3Loader - INFO    ========================= Check results ================================
...
2024-09-13 14:03:44 - PrepareS3Loader - INFO    dataframe_diff :    key_1  fld_str_txt_1 fld_str_txt_2 fld_str_txt_3  cnt_1  cnt_2
2     3          3     Value3_1    Full copy    2     0
3     3          3    Value 3_1    Full copy    0     1
...
2024-09-13 14:03:44 - PrepareS3Loader - INFO    ==================== Finish check results ============================
```

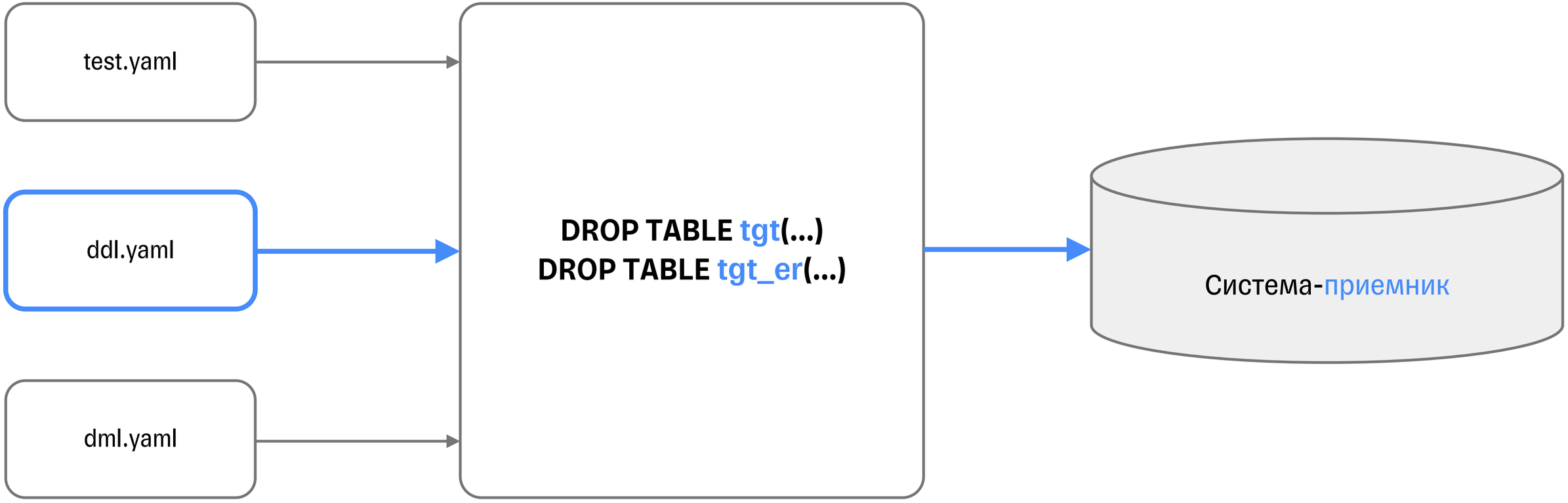
Пришло 2 строки, а ожидали 1!!!

Очистка окружения



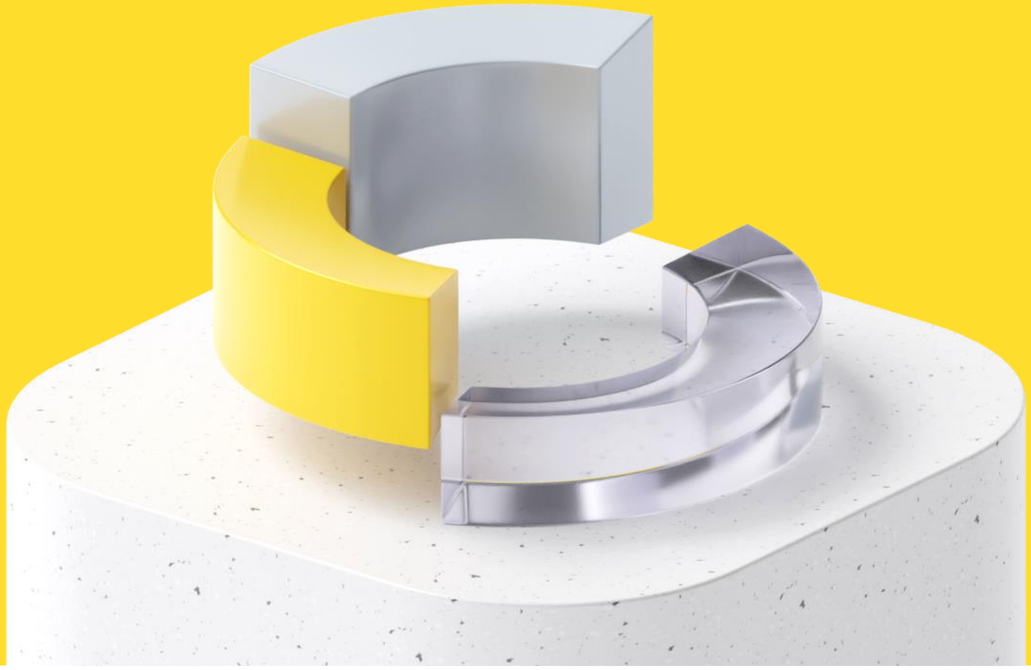
Prepare класс

Очистка окружения



Prepare класс

Подведём итоги



- DWH сложно,
  но можно тестировать
- Тестирование
  системных компонент –
  очень интересно
- QA Auto в DWH – это
  исследовательская
  работа

Т БАНК

Спасибо!