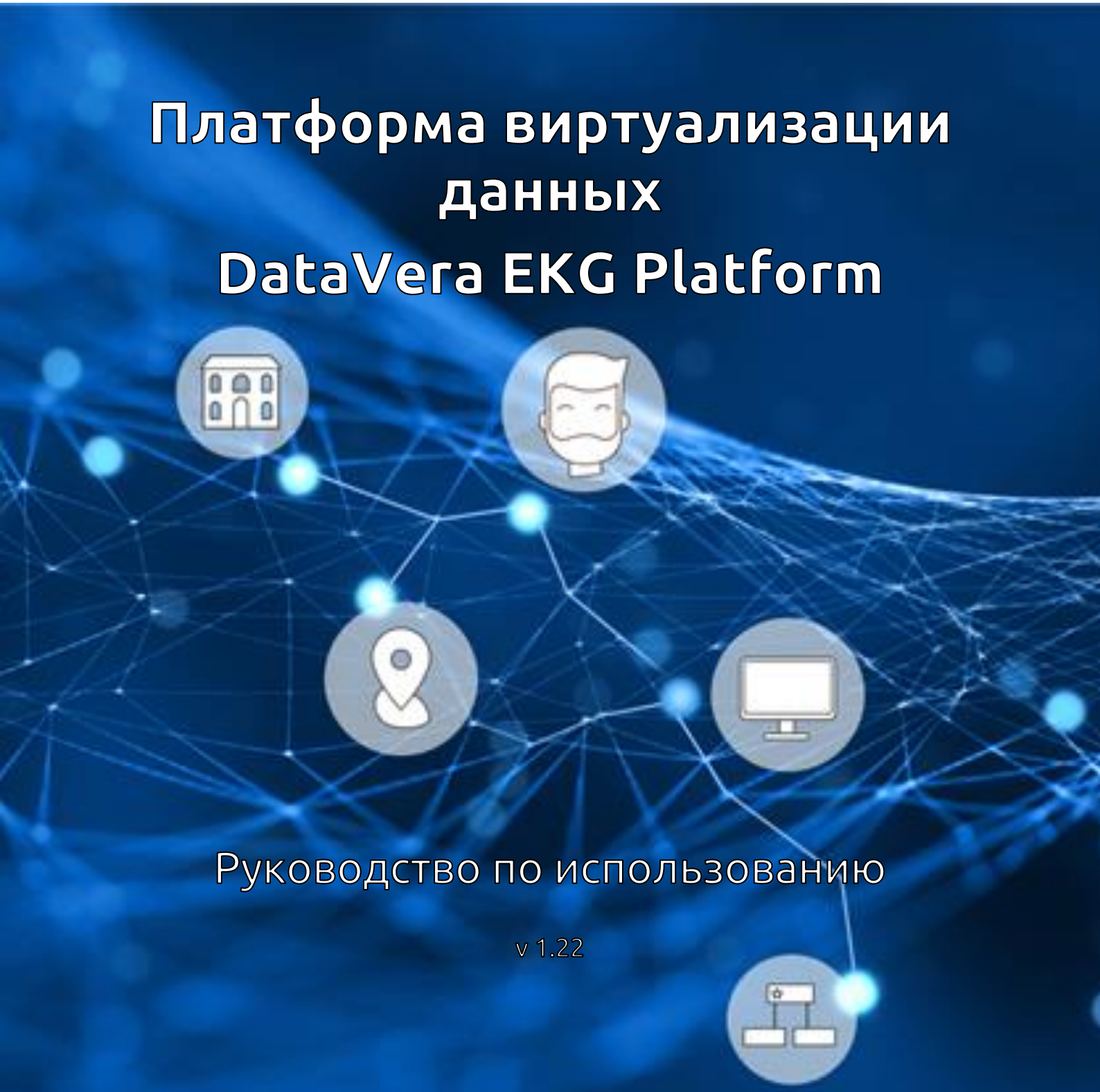
DataVera

# Платформа виртуализации данных

# DataVera EKG Platform



Руководство по использованию

v 1.22

# Содержание

# 1. Назначение и принцип работы

**DataVera EKG Platform** – платформа виртуализации данных, предназначенная для представления больших массивов информации в виде **корпоративного графа знаний** (**EKG, Enterprise Knowledge Graph**). Корпоративный граф знаний представляет собой связное, структурированное представление информации, в котором каждый информационный объект представлен вершиной графа, а каждое значение свойства – ребром графа.

Структура информации в EKG определяется онтологической моделью, или **онтологией**. Онтологические модели строятся в соответствии со спецификациями консорциума W3C: RDF, RDFS, OWL. Онтология позволяет определить типы сущностей (классы) и определить свойства, значениями которых могут обладать объекты этих классов. Уровень определений классов и свойств в онтологии называется TBox (Terminology Box). Фактические сведения о конкретных объектах выражаются при помощи заданного набора классов и свойств на уровне ABox (Assertions Box, утверждения о конкретных объектах).

Схематически структура содержимого онтологического корпоративного графа знаний показана на рис. 1.

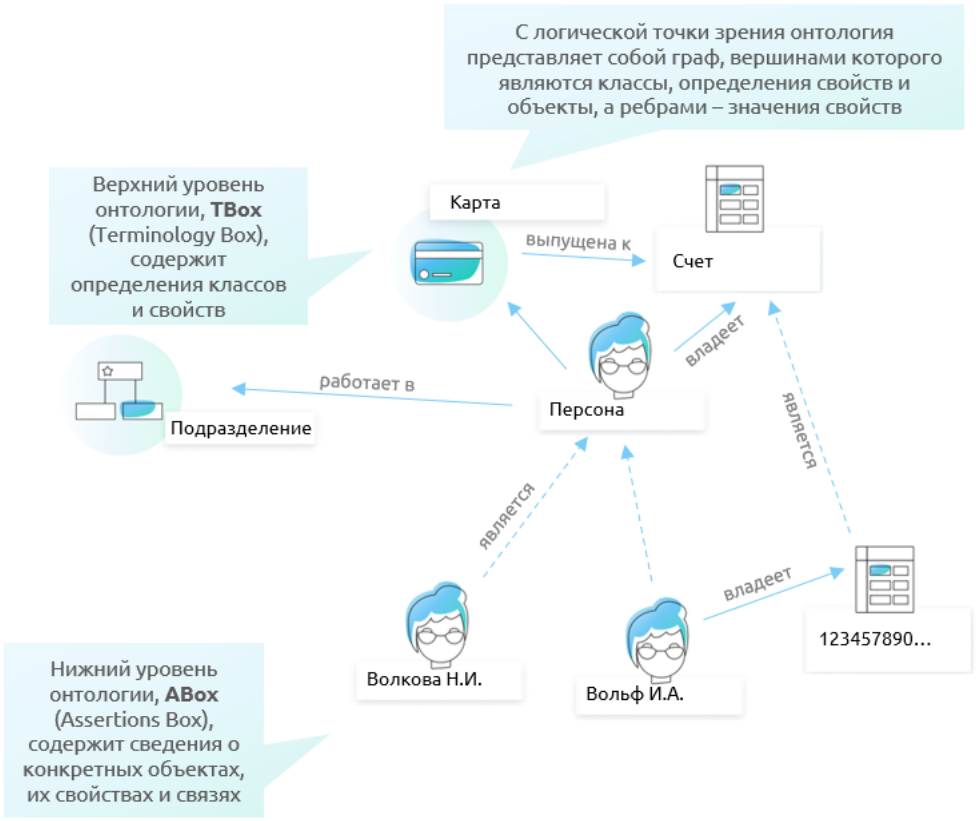



Рис. 1. Пример логической структуры корпоративного графа знаний

Граф знаний можно построить и без использования онтологической модели – для этого существуют и другие способы. Можно создать онтологию для описания предметной области, но не использовать ее для создания корпоративного графа знаний. Однако совместное использование методов и спецификаций онтологического моделирование с парадигмой корпоративного графа знаний позволяет достичь наибольшей гибкости в управлении данными организации. Настоящее руководство рассказывает о том, как использовать инструменты компании DataVera для решения задач в этой области.

Естественным способом хранения информации, представленной в виде графа, являются графовые базы данных. Существует класс графовых СУБД, специально предназначенных для работы с онтологическими моделями – **хранилища триплетов, RDF triple stores**. Для работы с содержимым таких СУБД существует язык запросов **SPARQL**, синтаксис которого определен спецификацией W3C. Хранилища триплетов с поддержкой SPARQL удобны для работы с данными, представленными в виде онтологии, но они существенно проигрывают другим видам СУБД в скорости обработки больших объемов данных. Графовые СУБД могут использоваться для создания аналитических хранилищ информации, но мало пригодны для хранения операционных, или транзакционных данных. В то же время, в управлении корпоративными данными обработка именно операционных данных является первоочередной задачей. Чтобы использовать все преимущества гибкости онтологических моделей данных, аналитические возможности графовых СУБД, не теряя надежности и скорости обработки информации, которыми обладают реляционные и документ-ориентированные хранилища, необходимо использовать технологию **виртуализации данных**. Ее принцип в контексте описываемых в этом руководстве задач состоит в том, что **физически данные должны быть распределены между множеством «традиционных» (в первую очередь реляционных) СУБД** для обеспечения надежности и производительности, но **логически они должны быть представлены в виде графа**, структура которого определяется онтологической моделью. Именно такой способ работы с данными реализует один из компонентов платформы – DataVera EKG Provider.

**Виртуализация данных**

Физически информация находится во множестве реляционных СУБД, но логически представлена в виде графа. Структура виртуального графа соответствует онтологической модели.

EKG Provider предоставляет поставщикам и потребителям информации программный интерфейс (API), с помощью которого они могут взаимодействовать с данными, не заботясь о том, где и в какой структуре они физически хранятся. API позволяет работать с данными так, как будто они находятся в графе и соответствуют структуре онтологической модели – в том числе можно использовать язык SPARQL для доступа к ним. **Структура модели, TBox**, действительно хранится в графовой СУБД (в базовом варианте развертывания используется свободно распространяемая Apache Fuseki, но возможно использование и других хранилищ триплетов). **Фактические**

**данные, ABox**, могут частично находиться в графовой СУБД (обычно в ней остается содержимое часто используемых справочников и перечислений) или во множестве реляционных СУБД. В качестве реляционной СУБД в базовом варианте развертывания используется Postgresql. К одному экземпляру EKG Provider в качестве хранилищ данных можно подключить несколько СУБД Postgresql, или несколько кластеров этой СУБД. В реляционную СУБД проецируются наборы значений крупных справочников, транзакционная информация, временные ряды и другие виды данных, значительных по объему, но относительно однообразных по структуре.

Граф, отражающий онтологическую модель, может включать не только сами данные и описание их структуры, но и **правила обработки данных**. Наиболее современная спецификация W3C, описывающая способ представления правил – SHACL. EKG Provider поддерживает обработку правил SPARQL Rules, описанных в этой спецификации. Правила представляются как объекты классов онтологии, определенных в спецификации, и находятся в хранилище триплетов.

На уровне настроек EKG Provider определяется, в каком экземпляре СУБД хранятся объекты тех или иных классов. Взаимосвязь EKG Provider с другими компонентами, необходимыми для его работы, показана на рис. 2.

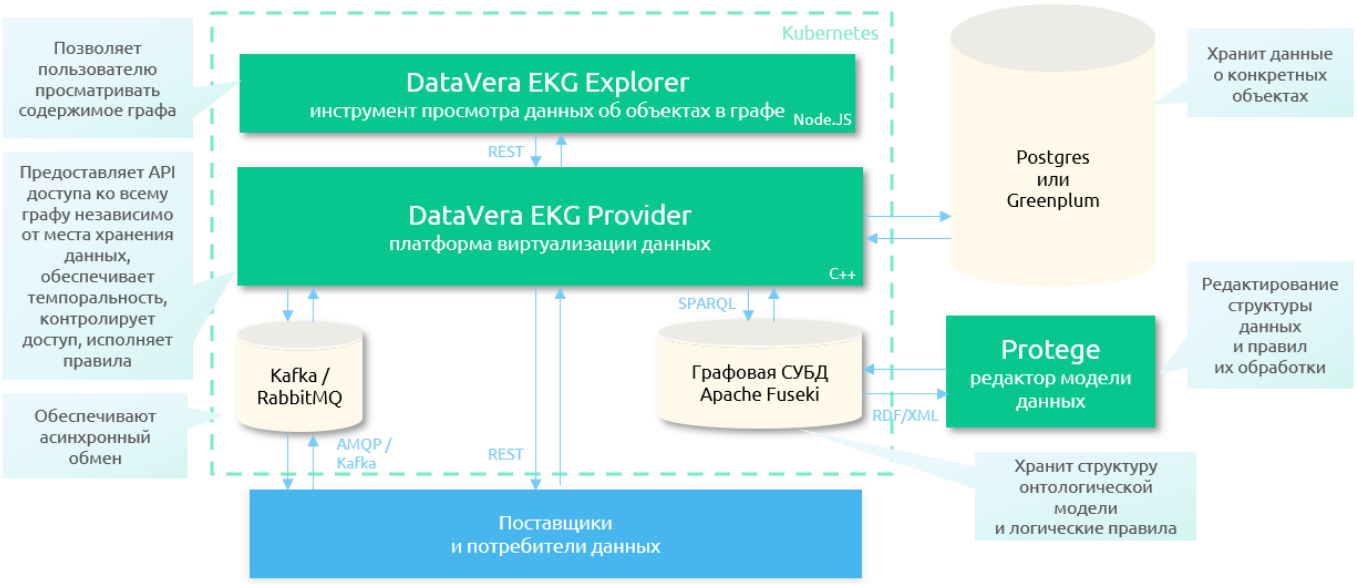



Рис. 2. Взаимодействие EKG Provider с другими компонентами программной среды

DataVera EKG Platform в ИТ-инфраструктуре организации может выполнять следующие задачи **управления данными (Data Governance)**:

1. Выполнение функций MDM-системы: хранение нормативно-справочной информации (НСИ) и мастер-данных. Консолидация сведений об одних и тех же объектах бизнес-процессов из разных источников информации, очистка, нормализация и дедупликация, распространение эталонных данных системам-потребителям с помощью механизма

подписки. Решение задач интеграции данных. Пример решения подобной задачи приведен ниже в разделе «Работа с подписками».

2. Форматно-логический контроль данных в соответствии с настраиваемыми правилами, контроль целостности данных, автоматическое дополнение данных на основе правил.
3. Консолидация данных – формирование эталонных объектов на основе объектов из источников данных.
4. Хранение истории появления и преобразования всех объектов данных (Data provenance).
5. Хранение истории всех состояний каждого объекта данных (темпоральность данных) с возможностью получить объекты данных, соответствующие определенным условиям, по состоянию на любой момент времени.
6. Хранение аналитической информации, консолидированной из множества прикладных автоматизированных систем и хранилищ организации. Создание Системы управления знаниями – единого окна доступа к любой корпоративной информации. Пример решения подобной задачи приведен ниже в разделе «Точка доступа SPARQL».
7. EKG Provider может стать ядром дата-центричной ИТ-архитектуры предприятия, выполняя роль единого «облачного» хранилища данных для приложений, не имеющих собственной СУБД.

Представление данных с помощью онтологической модели имеет несколько важных особенностей и преимуществ:

- Каждый объект может одновременно входить в несколько классов (относиться к нескольким типам). Например, объект «ИП Вольф» может одновременно входить в класс «Персона» и в класс «Индивидуальный предприниматель».
- Классы образуют одну или несколько иерархий по принципу «надкласс»-«подкласс» (надмножество-подмножество). Например, класс «ТОО» может являться подклассом класса «Юридическое лицо».
- Свойства также образуют иерархии. Например, свойство «Адрес регистрации» является под-свойством для свойства «Адрес».
- Каждое свойство может быть применимо к объектам разных классов. Например, значением свойства «Размер капитала» могут обладать как персоны, так и организации – это позволяет сравнивать, фильтровать и сортировать объекты разных типов по одним и тем же свойствам.
- Каждое свойство каждого объекта может иметь одновременно несколько значений. Например, организация может иметь несколько адресов или телефонов.
- Каждое строковое свойство может иметь несколько значений, заданных на разных языках. Например, для объекта «Казахстан» может быть задано наименование на казахском, английском и русском языках.

Все эти и другие преимущества доступны при работе с данными и моделью с помощью API EKG Provider несмотря на то, что физически часть информации находится в реляционных СУБД.

# 2. Архитектура приложения DataVera EKG Provider

EKG Provider – Linux-приложение, написанное на языке C++. Оно поставляется в виде контейнера и предназначено для запуска в standalone-режиме или в кластере Kubernetes. На рис. 3 показана архитектура компонентов EKG Provider.

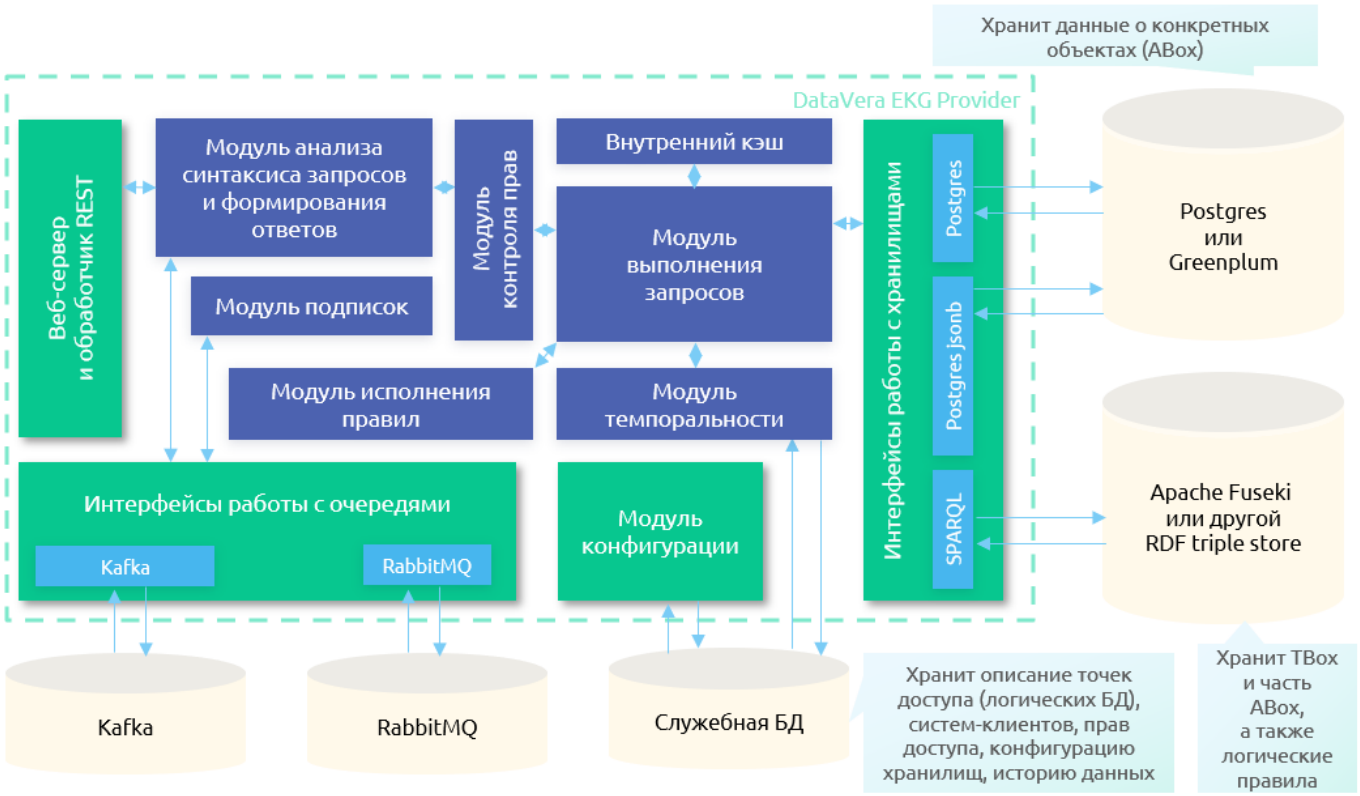



Рис. 3. Архитектура компонентов EKG Provider

При старте исполняемый модуль ekg-provider запускает несколько процессов:

- Веб-сервер, обслуживающий REST API и точку доступа SPARQL. Для обработки каждого входящего запроса он порождает новый процесс (или несколько, если необходимые данные находятся сразу в нескольких хранилищах)
- Обслуживающий процесс для каждой прослушиваемой входящей очереди сообщений Kafka или RabbitMQ
- Внутренний сервис рассылки уведомлений об изменении объектов данных по подписке
- Родительский процесс, отслеживающий изменение глобальных настроек приложения и перезапускающий при необходимости дочерние процессы.

# 3. Некоторые технические аспекты онтологических моделей данных

Для понимания структуры API необходимо привести некоторые сведения о том, как выражаются знания в онтологических моделях. Мы рекомендуем более детально ознакомиться с методами и технологиями онтологического моделирования при помощи одного из доступных в Интернете учебников или методических пособий.

В качестве идентификаторов сущностей в онтологии используются IRI, т.е. идентификатор объекта представляет собой выражение типа {префикс}#{уникальная часть идентификатора}. Например, идентификатор класса «Персона» может иметь вид http://xmlns.com/foaf/0.1/Person, а идентификатор конкретной персоны - http://xmlns.com/foaf/0.1/Person_Jane_Doe. Каждый идентификатор относится к определенному пространству имен, которое называется «префиксом». Например, распространенная онтология FOAF (Friend of a friend), пример идентификатора из которой приведен выше, использует префикс http://xmlns.com/foaf/0.1/ - все объекты этой онтологии имеют IRI, начинающийся с этого выражения. Существует сокращенная запись префиксов, например запись foaf:Person эквивалентна http://xmlns.com/foaf/0.1/Person.

Каждая онтология может иметь префикс по умолчанию – в EKG Provider его можно указать для каждой точки доступа, т.е. отдельного графа. В сокращенной записи этот префикс можно не указывать. Например, если онтология имеет префикс по умолчанию http://datavera.kz/model, идентификатор объекта http://datavera.kz/model/Asset будет эквивалентен записи Asset.

Автор онтологии может выбирать пространства имен и присваивать идентификаторы сущностям по своему усмотрению. Однако для обеспечения интероперабельности рекомендуется использовать заданные в спецификациях и рекомендациях W3C идентификаторы распространенных сущностей, такие как org:Organization или sosa:Sensor. В одной онтологии можно использовать вместе идентификаторы из разных пространств имен, это не несет никаких последствий – для СУБД любой IRI представляет собой просто строку. Для самостоятельно вводимых идентификаторов можно использовать любые пространства имен. Идентификаторы индивидуальных объектов можно присваивать произвольно. Мы рекомендуем использовать для них абстрактные значения, не связанные со свойствами объекта, которые могут измениться со временем – например guid или любой хэш.

Спецификации W3C, определяющие способы моделирования онтологий (RDF, RDFS, OWL), также имеют свои пространства имен. В этих пространствах заданы идентификаторы основных типов сущностей и некоторых свойств. Например, определены четыре основных типа сущностей:

- owl:Class – класс

- owl:ObjectProperty – свойство, представляющее собой ссылку с одного объекта на другой
- owl:DatatypeProperty – свойство, значением которого является литерал (строка, число и др.). Используются типы XSD, например xsd:string, xsd:integer и т.д.
- owl:NamedIndividual – индивидуальный объект, который обычно относится к одному или нескольким классам и обладает значениями свойств.

Среди стандартных свойств (предикатов) важно упомянуть:

- rdf:type – отношение сущности к типу. Используется для указания «основного» типа каждой сущности из приведенного выше списка, а также для указания принадлежности индивидуального объекта определенным классам.
- rdfs:label – читаемое название сущности
- rdfs:subClassOf – отношение между подклассом и надклассом
- rdfs:domain – набор классов, объекты которых могут обладать значением некоторого свойства (область определения свойства)
- rdfs:range – диапазон значений свойства

Во всех запросах к API EKG Provider URI могут указываться как в полном, так и в сокращенном формате. В ответах EKG Provider для элементов, префикс URI которых совпадает с префиксом по умолчанию, возвращается только уникальная часть URI. Для остальных элементов URI приводится полностью.

# 4. Обзор функций EKG Provider

EKG Provider предоставляет через API следующие группы функций:

- Извлечение объектов данных
  - Конкретного объекта по известному идентификатору
  - Группы объектов по набору условий
- Извлечение структуры онтологической модели TBox – набора классов и свойств
- Работа с темпоральной информацией
  - Получение объекта данных по состоянию на определенный момент времени
  - Извлечение всей истории изменений конкретного объекта данных
  - Получение набора объектов, которые отвечали условиям заданных фильтров на определенный момент времени
- Создание объекта данных или изменение значений его свойств
- Массовая загрузка объектов данных
- Консолидация дубликатов и формирование эталонных объектов данных
- Получение метрик качества набора данных
- Удаление объекта данных

- Применение к объекту правил контроля целостности и правил логического вывода
- Подписка на получение сообщений об изменении/удалении объектов определенных классов, изменение свойств подписки
- Получение списка активных подписок
- Удаление подписки

Доступен также SPARQL-интерфейс (с рядом ограничений).

Функции основного API предназначены прежде всего для выполнения операций с данными, типичных для традиционных СУБД: CRUD-операций (Create, Update, Delete), а также работы с данными в режиме подписки и работы с темпоральными данными. Во всех случаях речь идет о работе с отдельными объектами данных или с группами однородных объектов.

SPARQL-интерфейс предназначен для выполнения операций, более типичных для графовых СУБД: поиска паттернов в графе, которые представляют собой наборы связанных определенным образом элементов. С помощью SPARQL-интерфейса удобно обнаруживать связи между разнородными объектами, выстраивать длинные цепочки связей. Примеры таких запросов приводятся ниже в подразделе «Точка доступа SPARQL».

С некоторой долей условности можно сказать, что основной API предназначен прежде всего для задач управления основными и операционными данными, а SPARQL-интерфейс – для аналитической обработки сложных связанных данных.

Запросы на извлечение данных способны объединять информацию из объектов из разных физических хранилищ. Запросы на изменение данных могут обрабатывать ситуацию перемещения объекта из одного хранилища в другое из-за изменения его принадлежности классам, или хранения объекта сразу в нескольких хранилищах.

На уровне конфигурации EKG Provider настраивается:

- Набор «точек доступа» – отдельных наборов данных, доступ к которым обеспечивает EKG Provider
- Набор хранилищ и правил распределения объектов данных между ними, правила сопоставления элементов структуры онтологии элементам структуры хранилищ
- Набор систем-клиентов
- Права доступа систем-клиентов к объектам различных классов
- Прослушиваемые очереди Kafka/RabbitMQ, через которые клиенты могут отправлять асинхронные запросы к API.

Управление структурой онтологической модели в текущей версии платформы может выполняться с помощью свободно распространяемого редактора Protégé.

Для управления конкретными данными (поиска, редактирования) предназначен другой компонент нашей платформы – DataVera EKG Explorer.

# 5. REST API EKG Provider и работа через очереди Kafka/RabbitMQ

EKG Provider предоставляет два способа взаимодействия с его программным интерфейсом:

- Синхронный, с помощью REST API или SPARQL точки доступа
- Асинхронный, через очереди Kafka (рекомендуется) или RabbitMQ.

Функции, предоставляемые через REST API и через очереди, совершенно одинаковы. Swagger-описание REST API доступно по адресу http://api.k8s.datavera.kz/. Любой метод API имеет следующие общие параметры:

- URL – тип метода. Может принимать следующие значения (полный список см. далее):
    - Entity – работа с одной сущностью, с одним информационным объектом
    - Entities – работа с группой объектов, отвечающих определенным условиям
    - Bulk – массовая загрузка объектов
    - Validate – применение правил контроля целостности и правил логического вывода к объекту
    - Model – получение TBox, структуры онтологической модели
    - History – работа с историческими, темпоральными данными
    - Subscription – работа с подписками
    - Endpoints – получение списка точек доступа
    - Languages – получение списка доступных языков

  При работе с REST API указывается в составе URL, при работе через очереди передается в теле запроса как параметр "URL", например "URL": "entity".
- Метод. При обращении к REST API указывается как HTTP-метод, при работе через очереди передается в теле запроса как параметр "Method", например "Method": "POST". Поддерживаются следующие методы: POST (заменяет GET, т.к. любой запрос имеет тело), PUT/PATCH (эквивалентны), DELETE.
- Endpoint – идентификатор точки доступа. Точка доступа – это отдельный набор данных внутри EKG Provider, можно рассматривать его как «отдельную базу данных». Внутри каждой точки доступа существует своя онтологическая модель. Данные в разных точках доступа не могут ссылаться друг на друга. При работе с REST API указывается в составе URL, при работе через очереди передается в теле запроса как параметр "Endpoint", например "Endpoint": "demo".

- Client – идентификатор системы-клиента EKG Provider. Это имя учетной записи клиента, которая используется для определения уровня прав доступа клиента к данным. Всегда передается в теле запроса, например "Client": "test".
- Rid (не обязательно) – идентификатор конкретного запроса. Этот же идентификатор будет содержаться в ответе на запрос и позволит сопоставить запрос с ответом при работе через очереди. Всегда указывается в JSON-пакете. Всегда передается в теле запроса.
- Версия API (только при работе с REST API), всегда передается в составе URL.

URL REST-сервиса имеет следующую структуру: /v{Version}/{Endpoint}/{URL}, например: /v1/demo/entity

Тело запроса представляет собой JSON-коллекцию, в которой содержится «полезная нагрузка» вызова. Состав этой коллекции отличается для каждого из методов.

# 6. Работа с конфигурацией платформы

## 6.1. Получение списка точек доступа

Запрос POST /v1/endpoints позволяет получить информацию о точках доступа, к которым может обращаться система-клиент. Тело запроса должно содержать единственный параметр, "Client":

```
{
  "Client": "demo"
}
```

Ответ на этот запрос имеет такой вид:

```
{
  "Client": "demo",
  "Endpoints": [
    {
      "Code": "Demo",
      "Label": "Demo"
    }
  ]
}
```

Массив Endpoints в ответе содержит список точек доступа. Для каждой из них приводится код, который можно использовать как часть URL при обращениях к другим методам REST-сервиса, и читаемое название.

## 6.2. Получение описания структуры онтологической модели (TBox)

В EKG Provider структура данных определяется структурой онтологической модели. Запрос POST /v1/{endpoint}/model возвращает верхний уровень онтологической модели (Tbox) в виде списка классов и определений свойств. Запрос имеет необязательный параметр Lang, определяющий язык, на котором будут возвращены читаемые наименования классов и свойств.

Пример ответа на запрос:

```
{
  "Client": "test",
  "Endpoint": "demo",
  "DefaultPrefix": "http://datavera.kz/demo/",
  "Tbox": {
    "Classes": [
      {
        "URI": "Company",
        "Label": "Company",
        "SubClassesOf": [
          {
            "URI": "Organization"
          }
        ],
        "DomainsOf": [
          {
            "Type": "ObjectProperty",
            "URI": "HasOwner",
            "minCardinality": "1",
            "Ranges": [
              {
                "URI": "Person",
                "Label": "Person"
              }
            ]
          },
          {
            "Type": "DatatypeProperty",
            "URI": "Address",
            "maxCardinality": "1",
            "DataType": "http://www.w3.org/2001/XMLSchema#string"
          }
        ]
      }
    ]
  }
}
```


Ключ DefaultPrefix в корне коллекции содержит префикс онтологии по умолчанию. Все остальные IRI, как было сказано выше, приводятся без префикса в случае, если он равен префиксу по умолчанию.

Далее следует коллекция Tbox и вложенный в нее массив Classes, который содержит определения всех классов онтологии. Описание каждого класса содержит следующие ключи:

URI – идентификатор класса

Label – читаемое название класса

Archive – принимает значение true, если класс помечен на удаление.

Массив SubClassesOf – перечень URI классов, являющихся непосредственным подклассом для данного класса

Массив DomainsOf – перечисление свойств, значениями которых могут обладать объекты данного класса. Для каждого свойства приводятся:

- Type – тип свойства, может принимать значение DatatypeProperty (свойство-литерал) или ObjectProperty (свойство-связь)

- URI – идентификатор свойства
- minCardinality – минимальное число значений, которое может принимать свойство. Обычно используется для пометки обязательных свойств: значение minCardinality = 1 указывает, что свойству должно быть присвоено как минимум одно значение
- maxCardinality – максимальное число значений, которое может принимать свойство. Указание maxCardinality = 1 сообщает системе, что свойству нельзя присвоить более одного значения (по умолчанию любое свойство может иметь любое число значений)
- DataType – XSD-тип значения для литеральных свойств
- Ranges – перечень классов из диапазона значений для свойств-связей (объекты этих классов могут быть значениями свойства).

Каждое свойство может быть применимо к объектам нескольких классов, в этом случае его описание дублируется в описаниях этих классов.

### 6.3. Работа с подписками

EKG Provider предоставляет клиентам возможность подписаться на получение уведомлений о создании/изменении/удалении объектов данных через очереди Kafka (рекомендуется) или RabbitMQ. Этот механизм позволяет синхронизировать состояние выбранных справочников (классов) между EKG Provider и системой-клиентом. При любом изменении содержимого справочника система-клиент получит информацию об этом и сможет синхронно внести изменения в хранящуюся на ее стороне копию объекта.

Для создания и изменения свойств подписок служит запрос PUT (PATCH) /v1/{endpoint}/subscription. Пример тела запроса:

```
{
  "Code": "1",
  "Classes": [
    {
      "URI": "foaf:Person"
    }
  ],
  "Self": 0,
  "Broker": "Kafka",
  "Host": "127.0.0.1",
  "Port": "9092",
  "Username": "string",
  "Password": "string",
  "Queue": "system_sync"
}
```

Ключи тела запроса имеют следующий смысл:

Code – код существующей подписки, используется в запросах на изменение. При создании подписки не указывается.

Classes – массив URI классов, изменения объектов которых интересуют подписчика.

Self – флаг, принимающий значения true или false. Если указано значение true, то системе-клиенту будут отправляться в том числе уведомления об изменении свойств объектов, сделанных от имени этой системы. По умолчанию используется значение false, которое означает, что уведомления приходят только об изменениях, сделанных другими системами-клиентами.

Broker – тип брокера, может принимать значения Kafka или RabbitMQ.

Host и Port – IP-адрес и порт для подключения к брокеру

Username и Password – имя пользователя и пароль, которые EKG Provider будет использовать для авторизации на брокере (в текущей версии авторизация поддерживается только для RabbitMQ, доступ к брокеру Kafka должен быть без авторизации)

Queue – имя очереди RabbitMQ или топика Kafka, куда EKG Provider будет помещать изменения согласно этой подписке.

Система-клиент должна обеспечить считывание сообщений из указанной очереди, иначе очередь может переполниться.

Получить список подписок, существующих для данной системы-клиента, можно запросом POST /v1/{endpoint}/subscription. Запрос существует в двух вариантах. Если известен код (идентификатор) подписки, можно указать его в параметре Code в теле запроса. Если код не известен, в запросе можно указать массив Classes, перечислив в нем URI интересующих классов. Ответ на этот запрос в любом случае имеет такой вид:

```
{
  "Rid": "12345",
  "Client": "demo",
  "Endpoint": "demo",
  "Items": [
    {
      "Code": "123",
      "Classes": [
        {
          "URI": "foaf:Person"
        }
      "Self": 0,
      "Broker": "Kafka",
      "Host": "127.0.0.1",
      "Port": "9092",
      "Queue": "system_sync"
      ]
    }
  ]
}
```

Ответ содержит массив Items, каждый элемент которого соответствует одной подписке. Для каждой подписки указывается код, перечень URI классов, значение флага Self (см. выше описание запроса на установку подписки), а также реквизиты подключения – тип брокера, хост, порт и имя очереди/топика. Логин и пароль в ответе не возвращаются.

Запрос DELETE /v1/{endpoint}/subscription позволяет удалить подписку с заданным кодом.

Приведем пример сценария синхронизации данных между несколькими приложениями при помощи механизма подписки. Пусть существуют прикладные автоматизированные системы A и B, которые хранят и обрабатывают информацию о клиентах организации. В онтологической модели EKG Provider клиенты представлены классами «Организация» и «Персона». Также в модели создан набор свойств, которые описывают все характеристики персон и организаций, существенные с точки зрения бизнес-процессов заказчика.

В каждой из систем A и B хранится собственный набор свойств клиентов, которые описывают их с определенной точки зрения. Пусть система A является CRM-системой и описывает свойства клиентов, важные для взаимодействия с ними (сегментация, контактные данные и др.), а система B автоматизирует ведение бухгалтерского учета и описывает свойства клиентов, важные с этой точки зрения (регистрационные данные, банковские реквизиты и др.). Наборы клиентов в системах также отличаются: в системе A представлены как потенциальные клиенты, так и те, с которыми уже ведется работа, в системе B – только те клиенты, с которыми заключены договоры.

Обычно новые клиенты возникают в системе A. В этой системе должна быть реализована функция интеграции с EKG Provider, которая при создании, изменении или удалении клиента отправляет информацию об этом в EKG Provider путем выполнения запроса PUT /v1/{endpoint}/entity, или отправки аналогичного JSON-сообщения через очереди. Это позволит добиться синхронности данных между системой A и EKG Provider, который в данном случае выполняет функции MDM-системы.

Разумеется, интеграционная процедура на стороне системы A должна обрабатывать ответы от EKG Provider. В случае успешного создания нового объекта записывать в хранилище системы A уникальный код (URI), присвоенный ему EKG Provider, чтобы использовать его в последующих запросах на изменение и удаление объекта. В случае возникновения ошибок валидации или срабатывании правил форматно-логического контроля необходимо транслировать эти ошибки пользователю системы и требовать корректировки данных. При срабатывании правил поиска дубликатов – также требовать от пользователя подтверждения операций слияния или разъединения дубликатов.

На стороне системы B должна быть реализована интеграционная процедура, которая при запуске системы (или с определенной периодичностью) с помощью вызова метода POST /v1/{endpoint}/subscription проверит существование подписки на сообщения об изменении объектов классов «Организация» и «Персона». При отсутствии такой подписки она должна быть установлена путем вызова метода PUT /v1/{endpoint}/subscription. Интеграционная процедура со стороны системы B должна прослушивать очередь(-и), через которую установлена подписка, и при поступлении от EKG Provider сообщений о создании, изменении, удалении объектов указанных классов будет применять соответствующие изменения в хранилище системы B. При этом процедура может фильтровать объекты – например, сохранять только те из них, которые

описывают клиентов со статусом «Договор заключен» или последующим по ходу бизнес-процесса, чтобы исключить попадание в систему бухучета потенциальных клиентов.

Если со стороны системы B внесены какие-либо изменения в свойства клиента, например заполнены его бухгалтерские реквизиты или актуализирована контактная информация, система B должна отправлять эту информацию в EKG Provider так же, как это делает система A. Система A также может подписаться на получение обновлений информации о клиентах.

Можно построить бизнес-процесс управления данными и таким образом, что часть изменений в свойства клиентов вносится централизованно через EKG Explorer – пользовательский интерфейс, предназначенный для работы с данными, содержащимися в EKG Provider. Также может быть реализован компонент, проверяющий актуальность данных о клиентах в EKG Provider путем обращения к государственным базам данных. В обоих случаях это приведет к операциям обновления данных, выполняемым со стороны EKG Provider. Это означает, что все системы-клиенты должны подписаться на получение обновлений данных об объектах с тем, чтобы оперативно актуализировать содержимое своих собственных баз данных.

Каждая система передает и считывает в/из EKG Provider значения только тех свойств клиента, которые имеют соответствие среди свойств, обрабатываемых в этой системе. Для этого на стороне каждой интеграционной процедуры должна храниться таблица соответствия идентификаторов атрибутов онтологической модели EKG Provider идентификаторам атрибутов в интегрируемой системе.

Реализация такой схемы обмена позволит решить две основных задачи:

- Синхронизация данных между системами A и B с целью достижения актуальности данных, исключения расхождений данных между ними
- Легкость подключения новых систем: если появится система C, в которой также необходимо обрабатывать информацию о клиентах, эта система легко встроится в обмен и использует уже существующий набор данных о клиентах.

Кроме этого, достигаются дополнительные задачи:

- Централизация контроля качества данных путем применения одинаковых правил форматно-логического контроля независимо от источника и места хранения данных
- Очистка данных от дубликатов
- Хранение информации обо всей истории изменения объектов данных, которая может пригодиться в аналитических системах
- Снижение затрат на создание и поддержку интеграционных процедур за счет замены множества интеграций точка-точка между прикладными системами на централизованную схему интеграции.

### 6.4. Экспорт и импорт модели данных

Редактирование модели данных (TBox) может выполняться во внешнем приложении, например в редакторе Protégé. Платформа предоставляет методы для экспорта и импорта модели. Чтобы получить модель определенной точки доступа в формате N-Triples (.nq), нужно сделать запрос POST /v1/{endpoint}/model/export, передав в теле запроса идентификатор клиента:

```
{ "Client": "demo" }
```

Ответом будет текстовый файл в формате N-Triples. Этот файл можно изменить в редакторе Protégé или в любом другом редакторе онтологий, и загрузить в точку доступа EKG Platform, вызвав метод POST /v1/{endpoint}/model/import, передав в теле запроса идентификатор клиента и содержимое файла .nq в виде строки:

```
{ "Client": "demo", "TBox": "[ модель в формате .nq ]" }
```

Ответ на этот запрос имеет такой формат:

```
{ "Client": "demo", "Status": "[ результат выполнения операции ]" }
```

После того, как новая модель будет загружена в графовую БД, сервер платформы автоматически применит ее, обновив свой кэш модели.

### 6.5. Сброс кэша модели платформы

После изменения настроек платформы, правил, хранящихся в модели, или редактирования структуры TBox непосредственно в Fuseki может потребоваться инициировать обновление кэша модели EKG Provider. Для этого нужно вызвать метод POST /v1/reload, передав в теле запроса идентификатор клиента:

```
{ "Client": "demo" }
```

Ответ на этот запрос имеет такой формат:

```
{ "Client": "demo", "Status": "[ результат выполнения операции ]" }
```

## 7. CRUD-операции

### 7.1. Запрос одного объекта

Запрос POST /v1/{endpoint}/entity позволяет получить информацию об одном объекте с известным идентификатором. Пример тела запроса:

```
{
  "URI": "Jane_Doe",
  "Client": "demo",
```

```
  "Rid": "12345",
  "Return": {
    "Exclude": "false",
    "Properties": [
      "foaf:firstName",
      "foaf:surname"
    ]
  }
}
```

Идентификатор объекта задается в параметре URI и может быть приведен как в полной, так и сокращенной записи. Кроме этого, могут быть заданы следующие дополнительные параметры:

Lang – двухбуквенный ISO-код языка, на котором должны быть возвращены значения строковых свойств. Если значение какого-либо свойства не задано на указанном языке, будет возвращено значение на языке по умолчанию.

Date – получить информацию об объекте по состоянию на указанную дату (темпоральная, историческая информация).

NoLabels – не извлекать читаемые наименования объектов, на которые возвращаемый объект ссылается значениями свойств-ссылок. Указание true в качестве значения этого признака существенно ускоряет выполнение запроса.

Return – вложенная коллекция, определяющая, значения каких свойств объекта должны быть включены в ответ или исключены из него. На первом уровне коллекции содержится строковый элемент Exclude, который может принимать значения true (перечисленные далее свойства необходимо исключить, вернуть все остальные) или false (необходимо вернуть значения только перечисленных свойств). Элемент Properties представляет собой массив с идентификаторами свойств.

Ответ на этот запрос имеет такой вид:

```
{
  "Rid": "12345",
  "Client": "demo",
  "Endpoint": "demo",
  "Items": [
    {
      "Classes": [
        {
          "URI": "foaf:Person"
        }
      ],
      "DatatypeProperties": [
        {
          "URI": "foaf:firstName",
          "Value": "John"
        },
        {
          "URI": "foaf:surname",
          "Value": "Doe"
        }
      ]
    }
  ]
```

```
}
```

Параметры Rid, Client и Endpoint имеют те же значения, что были переданы в запросе. Коллекция Items является массивом из одного элемента, который содержит следующие ключи:

Classes – массив коллекций, каждая из которых может иметь два элемента: URI – идентификатор класса, которому принадлежит объект, Label – читаемое наименование этого класса.

DatatypeProperties – массив значений литеральных свойств объекта. Каждый элемент представляет собой коллекцию со следующими ключами: URI – идентификатор свойства (или Variable – имя вычисляемой переменной), Value – значение свойства, Lang – язык, на котором задано значение строкового свойства.

ObjectProperties – массив значений ссылочных свойств объекта. Каждый элемент представляет собой коллекцию со следующими ключами: URI – идентификатор свойства, Value – значение свойства (идентификатор другого объекта, на которое ссылается свойство), Inferred – идентификатор правила SHACL, в результате работы которого получено значение свойства.

NestedItems – массив связанных объектов (см. раздел «Подзапросы и агрегирующие функции»).

### 7.2. Запрос набора объектов

Запрос POST /v1/{endpoint}/entities возвращает информацию о наборе объектов, отвечающих заданным условиям. Пример запроса:

```
{
  "URI": "foaf:Person",
  "Client": "demo",
  "Rid": "12345",
  "Filters": [
    {
      "Combine": "or",
      "Filters": [
        {
          "URI": "foaf:firstName",
          "Comparison": "contains",
          "Value": "John"
        },
        {
          "URI": "foaf:firstName",
          "Comparison": "contains",
          "Value": "Jane"
        }
      ]
    }
  ],
  "Sorts": [
    {
      "URI": "foaf:firstName",
      "Direction": "asc"
    },
    {
      "URI": "foaf:surname"
    }
  ],
  "Return": {
    "Exclude": "false",
```

```
    "Properties": [
      "foaf:firstName",
      "foaf:surname"
    ]
  }
}
```

В запросе необходимо указать один или несколько классов, которым должны принадлежать возвращаемые объекты. Если класс один, он указывается в параметре URI, если несколько – то в массиве Classes, каждый элемент которого представляет собой коллекцию с единственным ключом URI, например: "Classes": [ { "URI": "foaf:Person" } ].

Кроме этого, могут быть заданы следующие дополнительные параметры:

Items – массив URI конкретных объектов, информацию о которых нужно вернуть.

Lang – двухбуквенный ISO-код языка, на котором должны быть возвращены значения строковых свойств. Если значение какого-либо свойства не задано на указанном языке, будет возвращено значение на языке по умолчанию.

Date – получить информацию об объекте по состоянию на указанную дату (темпоральная, историческая информация). Фильтры в этом случае также применяются не к текущим значениям свойств, а к тем, которые были актуальны на указанную дату. Запросы с указанием параметра Date выполняются медленнее, чем запросы на получение текущего состояния объектов данных.

Приведем пример. Пусть в наборе данных имеется объект John_Doe, описывающий персону John Doe. В период с 01.01.2010 John Doe работал в компании Alpha, LLC, а с 01.07.2015 – в компании Delta, PLC. Соответственно, значение свойства worksIn объекта John_Doe с 01.01.2010 по 01.07.2015 ссылалось на объект Alpha_LLC, а с 01.07.2015 по настоящее время ссылается на объект Beta_PLC (в следующем подразделе рассказывается о том, как записать историческую информацию «задним числом», если о факте смены работы стало известно позже, чем он произошел).

При выполнении запроса на извлечение объектов класса «Персона», работающих в компании Alpha_LLC, в настоящем времени (без указания ключа Date или с указанием значения большего, чем 01.07.2015) объект John_Doe не попадет в выборку. Однако если в том же запросе указать значение ключа Date, равное, например, 01.01.2014 – объект John_Doe будет возвращен в результатах запроса, т.к. по состоянию на указанный момент он удовлетворял условиям фильтра.

Такой же механизм темпоральности действует по отношению ко всем свойствам всех объектов, по отношению к вхождению объектов в классы и по отношению к существованию объектов. Например, если объект John_Doe будет удален из базы 01.01.2020, в результатах запросов на более ранние даты он все равно будет появляться.

Return – вложенная коллекция, определяющая, значения каких свойств объекта должны быть включены в ответ или исключены из него. На первом уровне коллекции содержится строковый элемент Exclude, который может принимать значения true (перечисленные далее свойства необходимо исключить, вернуть все остальные) или false (необходимо вернуть значения только перечисленных свойств). Элемент Properties представляет собой массив с идентификаторами свойств.

Compute – вложенный массив, содержащий список вычисляемых выражений, значения которых необходимо рассчитать для возвращаемых объектов (см. раздел «Вычисляемые выражения»).

Join – вложенный массив, содержащий список запросов на извлечение связанных объектов (см. раздел «Подзапросы и агрегирующие функции»).

WithoutSubclasses – флаг, принимающий значение true или false (по умолчанию). Если передано значение true, будут возвращены только объекты указанных классов, но не их подклассов.

NoLabels – не извлекать читаемые наименования объектов, на которые возвращаемые объекты ссылаются значениями свойств-ссылок. Указание true в качестве значения этого признака существенно ускоряет выполнение запроса.

Violations – флаг, позволяющий получить только объекты, нарушающие правила контроля целостности (см. раздел «Качество данных»).

ValidateOnly – флаг, позволяющий применить правила логического вывода и правила контроля качества данных к объектам, удовлетворяющим условиям выборки. Если этот флаг установлен, ответ не будет содержать информации о самих объектах, а будет содержать лишь разделы Affected и Violations для каждого затронутого объекта. Такой вызов эквивалентен группе вызовов POST /validate для каждого объекта. Если указать одновременно флаг OnlyConstraints, то правила логического вывода применены не будут.

PreserveEmptyMandatory – флаг, указывающий, что если при консолидации не удалось определить значение какого-либо свойства, но значение такого свойства уже есть в ранее созданном эталонном объекте, то необходимо сохранить это значение.

OnlyFreshest – флаг, указывающий, что при выборе значения наиболее свежего свойства при консолидации необходимо оставлять значение пустым, если самое свежее значение не удалось определить. Если же флаг равен false, то будет выполнен поиск второго, третьего и т.д. по свежести значения свойства, а при отсутствии истории значений свойств будет использоваться дата актуализации самого объекта из источника данных.

URIonly – флаг, принимающий значение true или false (по умолчанию). Если передано значение true, будут возвращены только идентификаторы объектов, отвечающих условиям запроса.

SaveCsv – в этом параметре можно указать полный путь и имя csv-файла, куда будет возвращен результат запроса (json-данные при этом не возвращаются). Параметр предназначен для использования при выгрузке данных в компонент AutoML. Путь должен указывать на S3-хранилище, подмонтированное к подам ekg-provider. Запрос с этим параметром обязательно должен содержать элемент Return (перечень значений возвращаемых свойств) для основного запроса и каждого связанного запроса, для которого не указан способ агрегации. В файл также выгружаются значения всех вычисляемых переменных (Compute) и агрегированных значений, рассчитываемых связанными запросами.

Offset и Limit – смещение от начала выборки и число возвращаемых значений. Эти параметры используются для постраничной выборки из больших массивов значений. Если запрос может вернуть большое число объектов (более 1000), необходимо использовать эти параметры, а не пытаться получить весь набор объектов одним запросом.

Count – флаг, принимающий значение true или false (по умолчанию). Если передано значение true, вместо объектов данных будет возвращено количество объектов, отвечающих условиям выборки.

Filters – набор условий (фильтров), которым должны удовлетворять возвращаемые объекты. Элемент filters в простом варианте может являться массивом вложенных коллекций, каждая из которых определяет одно условие. В этом случае в выборку попадут только те объекты, которые отвечают всем перечисленным условиям (логическое И). Пример элемента массива Filters: { “URI”: “foaf:firstName”, “Value”: “Jane”, “Comparison”: “equal”, “Lang”: “en” }. Для фильтров по ссылочным свойствам можно также указать в фильтре опциональный параметр ByName="true", который указывает, что операцию сравнения нужно провести не с идентификатором объекта, на который ссылается свойство, а с его читаемым наименованием (в текущей версии EKG Provider такой вариант можно использовать только с операцией точного сравнения).

Каждый элементарный фильтр содержит элемент URI, в котором указано свойство (если условие накладывается на вычисляемое выражение – вместо URI передается параметр Variable, см. раздел «Вычисляемые выражения»), элемент Value, в котором указывается проверяемое значение свойства, и элемент Comparison, который содержит название одной из операций сравнения:

- Equal – равно
- Notequal – не равно
- Contains – содержит (только для строк)
- Notcontains – не содержит (только для строк)
- Exists – существует значение (в этом случае передавать Value не требуется)
- Notexists – ну существует значение (в этом случае передавать Value не требуется)

- More, less, moreorequal, lessorequal – операции сравнения >, <, >=, <= для числовых значений.

Для строковых свойств можно также указать код языка в свойстве Lang, в этом случае сравнение будет выполняться только для значений свойств на указанном языке (иначе – для значений свойств на языке по умолчанию).

Если требуется объединить несколько фильтров более сложным образом, чем соединением с помощью логических «И», элемент filters может быть задан как коллекция, на первом уровне которой содержится элемент «Combine», принимающий значения «and» или «or», а также вложенный элемент filters, который в свою очередь может быть набором простых фильтров или еще одной вложенной секцией (допустимо до 2 уровней вложенности). В этом случае вложенные коллекции filters интерпретируются как «скобки» в логическом выражении. Элемент Combine на верхнем уровне задает логическую операцию между скобками, на следующем уровне – внутри скобок. Приведем пример:

```
"Filters": [
  {
    "Combine": "or",
    "Filters": [
       {
      "Combine": "and",
       "Filter": [
      {
        "URI": "foaf:firstName",
        "Comparison": "contains",
        "Value": "John"
      },
      {
        "URI": "foaf:firstName",
        "Comparison": "contains",
        "Value": "Jane"
      }
      ]
       },
       {
      "Combine": "and",
       "Filters": [
      {
        "URI": "foaf:surname",
        "Comparison": "equal",
        "Value": "Doe"
      }
         ]
       }
    ]
  }
]
```

Этот пример представляет логическое выражение ( ( foaf:firstName – содержит – John ) ИЛИ ( foaf:firstName – содержит – Jane ) И ( foaf:surname – содержит – Doe ) ), которое вернет объекты с информацией о персонах John Doe и Jane Doe.

Sorts – элемент, описывающий порядок сортировки результатов. Он представляет собой массив из коллекций, каждая из которых содержит два элемента: URI определяет идентификатор

свойства, по значению которого происходит сортировка, Direction может принимать значения «asc» или «desc» и определяет направление сортировки.

Структура ответа на этот запрос такая же, как у ответа на запрос одного объекта, с несколькими отличиями:

- массив Items может содержать несколько элементов;
- если в запросе были переданы вычисляемые выражения, в ответе будут содержаться их значения (см. раздел «Вычисляемые выражения»);
- если в запросе был задан параметр Count = «true», то в ответе вместо массива Items будет возвращен параметр Count, значение которого содержит число объектов данных, отвечающих условиям запроса.

Для получения полной версии Руководства обратитесь в компанию DataVera

### 7.3. Вычисляемые выражения

### 7.4. Подзапросы и агрегирующие функции

### 7.5. Получение истории изменений объекта

### 7.6. Создание или изменение свойств объекта данных

Запрос PUT (или PATCH) /v1/{endpoint}/entity позволяет создать новый объект данных или изменить свойства существующего. Пример запроса:

```
{
  "URI": "John_Doe",
  "UpdateMode": "Append",
  "Classes": [
    {
      "URI": "foaf:Person"
    }
  ],
  "DatatypeProperties": [
    {
      "URI": "foaf:surname",
      "Value": "Doe",
      "Lang": "en"
    }
  ],
  "ObjectProperties": [
    {
      "URI": "worksIn",
      "Value": "Alpha_LLC",
      "Label": "Alpha, LLC"
    }
  ]
}
```

Ключ URI содержит идентификатор объекта. Если объект с таким идентификатором не существует в EKG Provider, он будет создан, иначе – отредактирован. При создании объекта можно не указывать значение URI – в этом случае EKG Provider присвоит его сам и вернет в ответе. Сопоставить ответ с запросом в случае обмена через очереди можно по значению ключа Rid.

Ключ Date – позволяет записать сведения об истории изменений указанным временем, а не текущим. В предыдущем подразделе мы рассматривали пример, в котором свойство worksIn объекта класса «Персона» изменило свое значение с 01.07.2015, когда человек, информацию о котором хранит объект, сменил работу. Очевидно, что информация о смене работы не могла моментально попасть в базу данных – предположим, что о факте смены работы стало известно только 15.07.2015. В эту дату и будет выполнен запрос на изменение значений. Но для корректной работы запросов к историческим данным необходимо установить дату изменений равной 01.07.2015, то есть той дате, когда фактически произошло событие, описанное в данных. Для этого и предназначен параметр Date в запросе на создание/изменение объекта.

Ключ UpdateMode может принимать одно из трех значений, определяющих политику изменения свойств объекта:

- append – объекту будут добавлены новые значения свойств, переданные в запросе. Существующие значения свойств не изменятся. То же самое относится к принадлежности к классам: объект будет включен в новые классы, но не будет исключен из тех, к которым уже относится.
- replace – значения свойств, переданных в запросе, будут изменены на те, что переданы. Значения свойств, не указанных в запросе, останутся неизменными. Объект будет принадлежать только тем классам, которые переданы в запросе.
- fullupdate – объект будет перезаписан полностью. Он получит только те значения свойств, которые переданы в запросе, а значения остальных свойств будут очищены.

Ключ NoRefCheck позволяет отключить проверку существования объектов, на которые ссылается редактируемый объект. Использование этого ключа существенно ускоряет выполнение запроса на изменение объекта, но исключение появления ссылок на несуществующие объекты является ответственностью приложения.

Ключ CheckStorages позволяет отключить проверку наличия объекта в других хранилищах, кроме тех, в которые должен быть записан объект. Этот ключ со значением false можно использовать для ускорения выполнения запроса, если есть гарантия, что набор классов объекта не изменился и конфигурация хранилищ не менялась между предыдущим и текущим редактированием объекта.

Ключ NoHistory позволяет подавить запись истории изменений для данного запроса.

Ключ ApplyRules позволяет управлять тем, должны ли применяться правила логического вывода, нормализации значений свойств, поиска дубликатов и формирования эталонных объектов для редактируемого объекта. Внимание: при использовании флага ApplyRules = true в режиме загрузки множества объектов (bulk) применяются только правила нормализации и ограничения (Constraints), но не правила логического вывода.

Ключ NoPropagate позволяет исключить каскадное применение правил логического вывода к другим объектам, которые могут быть изменены в результате применения правил к текущему объекту. **Очень важно всегда указывать этот ключ при выполнении запросов на изменение объектов, для которых определены правила поиска дубликатов.**

Ключ OnlyConstraints позволяет применить к объекту правила контроля целостности, но не применять правила логического вывода и поиска дубликатов.

Далее в теле запроса следуют три массива: Classes, DatatypeProperties и ObjectProperties. Массив Classes содержит URI классов, которым принадлежит объект. Массивы DatatypeProperties содержат соответственно значения литеральных свойств и свойств-связей, которые будут ему присвоены. Для каждого свойства указываются следующие ключи:

URI – идентификатор свойства

Value – значение свойства (литерал или идентификатор другого объекта, в зависимости от типа свойства). Если пропустить этот ключ, т.е. указать только URI свойства, то все существующие значения этого свойства будут очищены независимо от выбранного режима обновления (UpdateMode).

Lang – язык, на котором задано значение строкового свойства (если не указан, то считается, что значение задано на языке по умолчанию).

Ответ на этот запрос сообщает результат операции и имеет такой вид:

```
{
  "Rid": "123456",
  "URI": "Person_123",
  "Status": "Error",
  "Message": "Invalid parameter"
}
```

Ключ Status может принимать значения «Error», «Object has been created», «Object has been updated» или «Object has been left intact». Ключ Message содержит описание ошибки. Ключ URI содержит идентификатор объекта, который был создан или изменен – не важно, был ли он передан в запросе или присвоен самой системой. Также в ответе могут содержаться ключи Violations и Affected, значение которых описано в разделе 5.7.

При создании или изменении свойств объекта платформа выполняет ряд проверок, в том числе:

- проверку существования всех объектов, на которые ссылаются значения свойств
- проверку кардинальностей – условий на минимальное и максимальное число значений свойств объекта (см. подробности в разделе «Получение описания структуры онтологической модели (TBox)»)
- проверку правил контроля целостности (см. раздел «Правила контроля целостности и правила логического вывода»).

В случае, если какая-либо из этих проверок не пройдена, клиенту будет возвращено сообщение об ошибке с указанием ее причины.

Также в момент создания или изменения свойств объекта применяются функции нормализации значений свойств (см. раздел «Функции нормализации значений свойств»).

После успешного выполнения операции происходит запись изменений в хранилище исторических (темпоральных) данных, а также передача во внутреннюю очередь задачи на отправку уведомления подписчикам об изменении свойств объекта. Собственно отправка уведомления подписчикам происходит позже в асинхронном режиме.

### 7.7. Массовая загрузка объектов данных

### 7.8. Удаление объекта данных

### 7.9. Точка доступа SPARQL

EKG Provider поддерживает выполнение некоторых видов SPARQL SELECT-запросов. Формат сервиса аналогичен тому, что реализует Apache Fuseki, поэтому консоль Fuseki можно использовать для выполнения запросов и просмотра ответов EKG Provider – достаточно заменить в строке адреса сервиса адрес Fuseki на адрес SPARQL точки доступа EKG Provider.

Запрос POST /v1/{endpoint}/query с текстом запроса в параметре query возвращает ответ в виде JSON-коллекции, состоящей из двух секций. Секция head со вложенным массивом vars содержит перечень переменных, входящих в состав ответа на запрос. Секция results со вложенным массивом bindings содержит элементы результата (tuples) в таком виде: { "type": "uri" | "literal", "value": "variable value" }. SPARQL-запросы можно выполнять только через REST-интерфейс, но не через очереди.

Приведем несколько примеров SPARQL-запросов, на которые может отвечать EKG Provider:

```
SELECT * WHERE { ?a ?b <http://datavera.kz/demo/John_Doe> }
```

В этом запросе переменные ?a и ?b указаны в позициях субъекта и предиката. Запрос вернет все объекты, ссылающиеся любым свойством на объект с идентификатором http://datavera.kz/demo/John_Doe (в SPARQL-запросах идентификаторы объектов необходимо приводить полностью, даже для объектов с префиксом, совпадающим с префиксом по умолчанию).

```
SELECT * WHERE {
      { ?obj rdf:type <http://datavera.kz/demo/Company> }
      UNION { ?obj rdf:type <http://datavera.kz/demo/Organization> }

      ?obj <http://datavera.kz/demo/Capital> ?capital
      OPTIONAL { ?obj <http://datavera.kz/demo/ParticipatesInProject> ?project }
      ?obj rdfs:label ?name

      FILTER (?capital > 1000) }
ORDER BY ?name ASC
```

Этот запрос вернет объекты классов Company и Organization, для которых указано значение свойств Capital и rdfs:label, причем значение свойства Capital превышает 1000, а также возможно задано значение свойства ParticipatesInProject. Результаты будут отсортированы по убыванию значения переменной ?name, в которую помещено значение предиката rdfs:label (читаемое название объекта).

Как упоминалось выше, язык запросов SPARQL удобен для поиска сложных цепочек связей объектов. Приведем пример:

```
SELECT * WHERE {
?obj rdf:type <http://datavera.kz/demo/Company>.
?obj <http://datavera.kz/demo/hasOwner> ?owner.
?owner ?relation <http://datavera.kz/demo/John_Doe> }
```

Такой запрос найдет все цепочки объектов

```
?obj (hasOwner) -> ?owner (любая связь) -> John_Doe
```

где переменная ?obj представляет объект класса Company, ?owner описывает владельца этой компании (который может принадлежать классам юридических или физических лиц), с условием, что владелец имеет связь любого типа с объектом John_Doe, представляющим конкретную персону. Такой запрос вернет все компании, владельцами которых являются родственники John_Doe, учрежденные им организации и т.д. Усложняя условия поиска, легко построить алгоритм поиска связей между аффилированными физическими и юридическими лицами.

Список возможностей, поддерживаемых в SPARQL-запросах для операторов SELECT и ASK:

- Использование паттернов с переменными в любой позиции
- Использование UNION для объединения паттернов (без вложенности)
- Использование OPTIONAL для указания не обязательных паттернов (без вложенности)
- Использование фильтров FILTER. Фильтр может содержать несколько условий, объединенных операторами && и ||. Вложенность условий в фильтрах не поддерживается. Поддерживаются некоторые функции, которые могут быть использованы в фильтрах, такие как STR(), CONTAINS(), SUBSTR(), CONCAT(), STRLEN(), DATEADD(), ROUND() и REGEX().
- Использование выражений EXISTS и NOT EXISTS
- Указание списка возвращаемых значений после ключевого слова SELECT
- Использование LIMIT и OFFSET
- Оператор CONSTRUCT
- Правила SHACL поддерживаются только в варианте SPARQL-правил

Список возможностей, НЕ поддерживаемых в SPARQL-запросах:

- Вложенные конструкции

- Property paths
- Пустые узлы
- Арифметические вычисления (кроме вычислений в SPARQL-функциях)
- GROUP BY и агрегирующие функции
- Сортировка ORDER BY
- Функции, кроме перечисленных в предыдущем списке
- Ключевое слово DISTINCT
- Выражение BIND (кроме использования для привязки результата SPARQL-функции)
- Операторы INSERT и DELETE

В дополнение к стандартным функциям SPARQL, EKG Provider определяет некоторые специальные функции:

| **Функция** | **Аргументы** | **Назначение** |
|---|---|---|
| DV_REPEATING_SYMBOLS_LEN | ?str – строка для проверки | Возвращает длину максимальной последовательности повторяющихся символов |
| DV_REPEATING_DIGITS_LEN | ?str – строка для проверки | Возвращает длину максимальной последовательности повторяющихся цифр |
| DV_ALPHABET_MIX | ?str – строка для проверки | Возвращает true, если строка содержит смесь символов латинского и не-латинского алфавита |
| DV_IIN_7TH_DIGIT_MATCH | ?iin – ИИН для проверки<br>?year – год рождения<br>?gender – пол (Gender_Male или Gender_Female) | Возвращает true, если 7-й символ ИИН соответствует году рождения и полу |

Еще одно дополнение к стандарту, которое можно использовать при работе с точкой доступа SPARQL EKG Provider – оператор HINT. Его стоит использовать в запросах, которые ищут пары объектов с совпадающими значениями свойств, причем эти значения имеют разную семантику. Рассмотрим запрос, который ищет дубликаты объектов класса «Персона» с одинаковым ИИН, полученные из одной и той же системы-источника. Такой запрос будет содержать условия:

```
$this <http://datavera.kz/ekg/iin> ?iin.
$this <http://datavera.kz/ekg/hasOrigin> ?origin.
?dupl <http://datavera.kz/ekg/iin> ?iin.
?dupl <http://datavera.kz/ekg/hasOrigin> ?origin
```

При выполнении запроса очевидно более экономичным вариантом является сначала выполнить поиск по ?iin, а затем по ?origin, т.к. значения свойства hasOrigin будут совпадать у миллионов объектов, а свойства iin – у единиц. Чтобы EKG Provider использовал эту оптимизацию, нужно добавить в запрос выражение HINT:

```
HINT(?iin, "mandatory")
```

Опишем пример ИТ-архитектуры, в которой EKG Provider играет роль центральной аналитической системы. Пусть в компании существует несколько автоматизированных систем, обрабатывающих структурированные операционные данные бизнес-процессов, и несколько файловых хранилищ, содержащих документы, сопровождающие выполнение этих процессов. Задача – построить «единое окно» поиска информации, которое позволит быстро найти все сведения, связанные с определенным бизнес-объектом или событием, определить возможные цепочки взаимосвязей между разными объектами или событиями, или решить прикладные задачи, такие как поиск аффилированных лиц, определение цепочек отказа в технологических системах и др.

При таком способе использования EKG Provider будет только получать информацию со стороны различных источников, но не будет распространять ее (если только не потребуется подключить BI-систему для количественного анализа собранных данных). Необходимо реализовать процедуры сбора данных, которые могут извлекать информацию из структурированных источников удобным для них способом. Если какие-либо прикладные системы предоставляют веб-сервисы для извлечения данных, процедуры сбора могут периодически обращаться к этим сервисам, а затем актуализировать соответствующую информацию в EKG Provider. Если возможен прямой доступ к базам данных структурированных источников (или к копиям таких СУБД, автоматически формируемым специально для передачи данных в корпоративный граф знаний), процедура сбора данных может использовать триггеры в СУБД для обнаружения изменений, а затем передавать изменения в EKG Provider с помощью менеджеров очередей. Если структурированные данные можно получать из файловых выгрузок – необходимо регулярно по расписанию запускать процедуру обработки свежих выгрузок, которая будет актуализировать соответствующую информацию в EKG.

Для работы подобных процедур потребуются:

- Набор классов и свойств онтологической модели, а также правила их мэппинга на структуру данных источников информации. Правила можно хранить в машинно-читаемом

виде так, чтобы не было необходимости вносить изменения в код интеграционных процедур при изменении структуры данных в источнике или в онтологической модели. Можно реализовать это с помощью конфигурационных JSON-файлов, или хранить правила мэппинга в самой онтологии.

- Наборы эталонных справочников, которые представляют в EKG перечисления, используемые в качестве значений свойств объектов: типы клиентов, виды оборудования и др. Для элементов таких перечислений должны храниться таблицы соответствия кодов элементов перечислений в EKG и в каждом из источников.
- Правила консолидации данных, в том числе правила идентификации сущностей. На основе этих правил интеграционная процедура будет принимать решения о том, как создавать или обновлять информационные объекты в EKG. Например, информация о каком-либо клиенте может храниться в нескольких источниках. При получении очередного клиента из какого-нибудь источника нужно будет выполнить его поиск в EKG для того, чтобы обновить существующий объект или создать новый (каждый бизнес-объект должен существовать в EKG в единственном экземпляре). Для этого необходимо опираться на набор ключевых свойств объекта, который и должен быть представлен в правилах. Например, поиск физических лиц можно выполнять по ИИН, при его отсутствии – по ФИО и дате рождения, при неполноте этой информации – по номеру телефона.

Интеграционная процедура со стороны каждого источника информации при работе с EKG Provider должна использовать собственный идентификатор системы-клиента (Client). Это позволит при анализе сформированного EKG проследить, из какого источника получены те или иные элементы информации о каждом объекте. Например, телефон клиента может быть получен из системы A, а его адрес – из системы B. Если интеграционные процедуры со стороны этих систем использовали разные идентификаторы клиента при записи данных в EKG, информацию об этом можно будет увидеть в ответе на запрос POST /v1/{endpoint}/history.

Для индексации неструктурированных данных из файловых хранилищ нужно реализовать процедуру-краулер, которая будет периодически сканировать всю структуру этих хранилищ, чтобы обнаруживать новые или изменившиеся файлы. Для каждого файла необходимо извлекать метаданные, для чего можно использовать название файла и путь к нему (например, имя каталога или файла может содержать название клиента, дату или тип оказываемой услуги), или содержимое файла. В содержимом можно обнаруживать ключевые слова, позволяющие установить связь с бизнес-объектами, процессами и событиями, либо реализовать смысловую обработку содержащихся там фактов с помощью технологий NLU (Natural Language Understanding, понимание смысла естественного языка). В любом случае, результатом обработки каждого файла должен становиться набор фактов, которые должны быть записаны в корпоративный граф знаний.

В результате будет сформирован полный, актуальный и связный массив информации, извлеченной из всех доступных источников. Осталось построить пользовательский интерфейс, который позволит выполнять поиск в этом массиве. В общем случае пользовательский интерфейс может состоять из двух основных блоков. Первый блок реализует поиск структурированной информации (фасетный поиск), где пользователь выбирает тип (класс) искомых объектов, затем задает условия поиска конкретных объектов, и выполняет поиск. Базовые функции такого рода выполняет и компонент нашей платформы – EKG Explorer. Каждый поисковый запрос пользователя должен транслироваться в обращение к методу POST /v1/{endpoint}/entities, отражающее все заданные пользователем условия поиска. Результат запроса – таблица информационных объектов, где пользователь может применить дополнительную фильтрацию, сортировку, экспортировать объекты в Excel, или перейти к просмотру детальной информации о каждом объекте.

Второй блок интерфейса реализует поиск взаимосвязей между объектами. Для реализации этой функции фактически необходимо создать конструктор SPARQL-запросов. Один из вариантов реализации может состоять в последовательном выборе исходного объекта и типа искомых объектов, с последующим полуавтоматическим или автоматическим формированием возможной структуры связей между этими объектами с опорой на структуру онтологической модели, которую можно получить запросом POST /v1/{endpoint}/model. Результатом конструирования будет SPARQL-запрос, выполнение которого позволит обнаружить искомую цепочку связей между объектами.

Возможны и другие варианты взаимодействия пользователя с EKG, в том числе графическое отображение как структуры концептуального уровня модели (возможных взаимосвязей объектов разных классов), так и связей конкретных объектов.

Для получения полной версии Руководства обратитесь в компанию DataVera

# 8. Качество данных

**8.1. Правила контроля целостности и форматно-логического контроля**
**8.2. SPARQL функции и проверка контрольных соотношений**
**8.3. Правила логического вывода и поиск дубликатов**
**8.4. Режимы применения правил**
**8.5. Консолидация данных – формирование эталонных объектов**
**8.6. Функции нормализации значений свойств**
**8.7. Метрики качества данных**

# 9. Конфигурирование EKG Provider

В каталоге EKG Provider должен находиться файл config.json, в котором задаются значения следующих ключей:

- postgres: строка подключения к служебной базе данных Postgresql в виде (пример): «user=postgres password=1 dbname=ekg». На сервере Postgresql должен быть разрешен способ авторизации md5 путем задания соответствующей настройки в файле pg_hba.conf.
- internal_broker: тип брокера, используемого для внутренней очереди рассылки по подписке – Kafka или RabbitMQ
- broker_host: IP-адрес брокера
- broker_port: порт брокера
- broker_user и broker_password: логин и пароль подключения к брокеру (в текущей версии применимы только для RabbitMQ)
- broker_queue: имя очереди (топика) для постановки задач на рассылку по подписке.

Все остальные настройки EKG Provider задаются в конфигурационной базе данных. Управлять ими можно при помощи консольных команд ekg-provider [команда] [параметры]. Если EKG Provider запущен в контейнере, то и команду нужно выполнять в контейнере.

После выполнения любых команд, описанных ниже, контейнер EKG Provider необходимо перезапустить для того, чтобы изменения вступили в силу.

Во всех случаях для того, чтобы пропустить значение параметра или присвоить ему пустое значение, используйте символ _ в соответствующей позиции в наборе параметров команды. Если параметр команды состоит из нескольких слов (например, название точки доступа или хранилища), заключайте его в кавычки.

## 9.1. Точки доступа

Получить список точек доступа можно командой

```
ekg-provider show endpoints
```

В ответ в консоль будет выведена таблица, содержащая список точек доступа. Для каждой из них приводится читаемое название для отображения в интерфейсе, код, используемый как значение параметра Endpoint в запросах, и префикс по умолчанию.

Чтобы создать новую точку доступа или изменить свойства существующей, выполните команду:

```
ekg-provider add | change endpoint [name] [code] [prefix] [default storage]
```

Значения параметров:

- name – читаемое название точки доступа. Может состоять из нескольких слов, в этом случае должно быть заключено в кавычки
- code – код точки доступа, который будет использоваться как значение параметра Endpoint при обращении к ней через API
- prefix – префикс по умолчанию для онтологии в данной точке доступа
- default storage – имя хранилища по умолчанию для данной точки доступа. Должно представлять собой имя хранилища Fuseki.

Поскольку точка доступа содержит ссылку на хранилище по умолчанию, а каждое хранилище привязано к какой-либо точке доступа, последовательность действий по созданию точки доступа выглядит так:

1. Создать точку доступа, не указывая значение параметра default storage
2. Создать хранилище Fuseki для этой точки доступа
3. Выполнить команду изменения точки доступа, указав имя созданного хранилища в качестве хранилища по умолчанию.

Для удаления точки доступа выполните команду:

```
ekg-provider delete endpoint [name или code]
```

### 9.2. Хранилища

Получить список хранилищ можно командой

```
ekg-provider show storages
```

В ответ в консоль будет выведена таблица, содержащая список хранилищ. Для каждого из них приводится читаемое название для отображения в интерфейсе, а также значения параметров, перечисленных ниже.

Создание или изменение свойств хранилищ выполняется командой:

```
ekg-provider add | change storage [name] [endpoint name or code] fuseki | postgresql [host] [port] [dbname or _ for Fuseki] [table or dataset for Fuseki] [login] [password] [history (0/1)] [logic (0/1)] [query_path for Fuseki] [update_path for Fuseki]
```

Параметры имеют следующее значение:

- name – читаемое название хранилища
- endpoint – читаемое название или код точки доступа
- Тип хранилища, который может принимать значение fuseki для Apache Fuseki или postgresql для Postgresql
- host – IP-адрес для подключения к хранилищу
- port – порт для подключения к хранилищу

- dbname – имя базы данных для Postgres, для хранилищ Fuseki в этом параметре надо указать _
- table – имя таблицы для Postgres или имя датасета для Apache Fuseki
- login и password – логин и пароль для подключения к хранилищу (для Fuseki используется простая HTTP-авторизация, для Postgres – md5 авторизация). Если логин и пароль не требуются, вместо каждого из них нужно указать _
- history – флаг, который может принимать значения 0 или 1. Если установлен в 1, то для данного хранилища будет вестись история значений свойств объектов
- logic – флаг, который может принимать значения 0 или 1. Если установлен в 1, то к объектам в данном хранилище будут применяться правила логического вывода
- query_path и update_path для Fuseki – пути к веб-сервисам query и update для Apache Fuseki (для Postgres-хранилищ эти параметры не нужно указывать). Пути указываются относительно корня веб-сервера и содержат имя датасета, например /demo/query и /demo/update.

Для удаления хранилища выполните команду:

```
ekg-provider delete storage [name]
```

### 9.3. Распределение объектов классов онтологии по хранилищам

Получить карту распределения объектов классов онтологии по хранилищам можно командой

```
ekg-provider show binding
```

В ответ в консоль будет выведена таблица, содержащая сведения о настройках сопоставления классов и хранилищ. В каждой строке выводится имя точки доступа, имя хранилища и URI класса, привязанного к этому хранилищу.

Привязать к хранилищу новый класс или отвязать класс от хранилища можно командой

```
ekg-provider bind | unbind [endpoint name or code] [storage name] [class URI]
```

Параметры:

- Код или название точки доступа
- Имя хранилища
- URI класса

Важно отметить, что после изменения привязки перемещение объектов класса между хранилищами не происходит автоматически (если объекты уже существовали в каком-либо хранилище). Если нужно переместить объекты из одного хранилища в другое, необходимо выполнить такую последовательность действий:

1. Привязать объекты к новому хранилищу командой bind. Настроить привязки элементов структуры модели к элементам структуры хранилища (см. следующий подраздел). Перезапустить ekg-provider, чтобы изменения вступили в силу.
2. Пересохранить все объекты в хранилище путем вызова методов API. Можно запросить все объекты, а потом сохранить их, не изменяя значения свойств.
3. Отвязать класс от старого хранилища и перезапустить ekg-provider, чтобы изменения вступили в силу.

### 9.4. Привязка свойств онтологии к столбцам хранилищ

Для хранилищ Postgres необходимо привязать структуру элементы онтологической модели к столбцам хранилищ. Получить карту привязки свойств онтологии к элементам структуры хранилищ можно командой

```
ekg-provider show mapping
```

В ответ в консоль будет выведена таблица, содержащая сведения о настройках сопоставления свойств и столбцов хранилищ. В каждой строке выводится имя хранилища, URI свойства, название соответствующей ему колонки таблицы Postgres, а также признак «является массивом» (столбцы Postgres могут быть объявлены массивами, например integer[], чтобы хранить несколько значений свойства) и указание на язык, если столбец хранит значения строковых литералов на определенном языке.

Привязать или отвязать свойство модели к столбцу хранилища Postgres можно следующей командой (для хранилищ Fuseki выполнять такие настройки не требуется):

```
ekg-provider map | unmap [storage name] [field name] [property URI] [is array?] [language]
```

Аргументы команды имеют следующий смысл:

- storage – название хранилища
- field – имя столбца в таблице СУБД
- property – URI свойства онтологической модели
- is array – флаг, который принимает значение 1, если столбец СУБД является массивом
- language – 2-буквенный ISO-код языка, на котором хранит значения данный столбец.

Не обязательно создавать в хранилищах Postgres отдельный столбец для каждого атрибута модели – это лишило бы решение гибкости, т.к. при создании каждого нового атрибута в модели пришлось бы создавать новый столбец в Postgres. Чтобы избежать такой проблемы, EKG Provider хранит значения всех свойств объекта в поле типа jsonb, которое имеет название data. Это поле входит в набор служебных полей, которые обычно присутствуют в любой таблице Postgres, подключенной как хранилище к EKG Provider:

- uri – имеет тип character varying(1024) и хранит URI объекта, которому соответствует строка (должно существовать обязательно)
- name – имеет тип character varying(1024) или text и хранит читаемое название объекта, имеет мэппинг на атрибут rdfs:label (должно существовать обязательно)
- type – имеет тип character varying(1024)[] и хранит список классов, в которые входит объект (должно существовать обязательно).
- data – имеет тип jsonb и хранит значения всех свойств объекта. Это поле может и не существовать в таблице – тогда каждый атрибут модели, применимый к классу, должен иметь соответствие отдельному столбцу таблицы.

Для остальных атрибутов имеет смысл создавать отдельные столбцы или в случае, если свойств у класса мало, а объектов в нем много (тогда целесообразно не создавать столбец data), или по свойству необходимо построить индекс. Впрочем, не менее эффективны и GIN-индексы по столбцу data. Индексы на таблицах Postgres необходимо создавать вручную.

## 9.5. Системы-клиенты

Получить список автоматизированных систем – клиентов EKG Provider можно командой

```
ekg-provider show clients
```

В ответ в консоль будет выведена таблица, содержащая список клиентов. Для каждого из них выводится название, код и точка доступа.

Создание или изменение свойств клиентов выполняется командой:

```
ekg-provider add | change client [name] [code] [endpoint name or code]
```

Параметры имеют следующее значение:

- Name – читаемое название клиента
- Code – значение, которое система-клиент будет передавать в качестве значения параметра Client при вызовах API
- Endpoint – читаемое название или код точки доступа, к которой получит доступ клиент

Для удаления клиента выполните команду:

```
ekg-provider delete client [name]
```

## 9.6. Управление правами доступа систем-клиентов

Каждой системе-клиенту можно назначить права на чтение и изменение объектов определенных классов. Права применяются с учетом наследования, т.е. система получит такие же права к объектам подклассов, какие установлены на надкласс. Если объект входит одновременно в

несколько классов, к которым установлен разный уровень доступа – выбирается наиболее строгий вариант. Получить список настроек прав можно командой

```
ekg-provider show rights
```

В ответ в консоль будет выведена таблица, содержащая список сочетаний систем-клиентов и классов модели. Для каждого из них выводится информация о наличии прав на просмотр объектов (Read) и их запись (Write).

Чтобы предоставить какой-либо системе права на объекты определенного класса, либо отозвать их, необходимо выполнить команду:

```
ekg-provider grant | revoke [endpoint name or code] [client name or code] [class URI] [read (1/0)] [write (1/0)]
```

Параметры имеют следующий смысл:

- endpoint – название или код точки доступа
- client – название или код системы-клиента
- class URI – идентификатор класса, к объектам которого предоставляется доступ
- read – установите в 1, если система должна иметь права чтения объектов этого класса, и в 0, если чтение необходимо запретить
- write – установите в 1, если система должна иметь права создания/редактирования/удаления объектов этого класса, и в 0, если эти операции необходимо запретить.

По умолчанию все системы имеют права на чтение и запись объектов всех классов. Если необходимо ограничить права на видимость или доступность для редактирования определенных ветвей онтологии, необходимо устанавливать права при помощи данной команды. Глагол grant используется в синтаксисе команды независимо от того, разрешается доступ или запрещается. Глагол revoke используется для удаления ранее назначенных прав и возврата к правам по умолчанию.

## 9.7. Языки

Получить список языков, на которых могут быть представлены данные в EKG Provider, можно командой

```
ekg-provider show languages
```

В ответ в консоль будет выведена таблица, содержащая список языков. Для каждого из них выводится название и двухбуквенный код ISO.

Создание языков выполняется командой:

```
ekg-provider add language [name] [ISO two-letter code in lowercase]
```

Параметры имеют следующее значение:

- name – читаемое наименование языка, обычно задается на самом этом языке
- ISO code – двухбуквенный код языка ISO в нижнем регистре, например en для английского языка.

Для удаления языка выполните команду:

```
ekg-provider delete language [ISO two-letter code in lowercase]
```

### 9.8. Прослушиваемые очереди

EKG Provider может прослушивать очереди Kafka и RabbitMQ, через которые в асинхронном режиме поступают запросы от систем-клиентов. В отличие от механизма подписки, в этом случае инициатором соединения выступает система-клиент, которая отправляет сообщение во входящую очередь и ожидает получить ответ в исходящую. Каждую пару таких очередей необходимо настроить, чтобы EKG Provider «знал», какие очереди необходимо прослушивать, и в какие очереди помещать ответные сообщения.

Получить список пар очередей, настроенных для прослушивания EKG Provider, можно командой

```
ekg-provider show queues
```

В ответ в консоль будет выведена таблица, содержащая список очередей. Для каждого из них выводится название и реквизиты входящей и исходящей очереди, исключая логин-пароль.

Создание или изменение свойств прослушиваемых очередей выполняется командой:

```
ekg-provider add | change | delete queues [name] Kafka | RabbitMQ [input_host] [input_port] [input_login] [input_password] [input_queue] [output_host] [output_port] [output_login] [output_password] [output_queue]
```

Параметры имеют следующее значение:

- name – читаемое имя пары очередей, которое используется только при их конфигурировании
- Kafka или RabbitMQ – тип брокера сообщений
- input_host и input_port – IP-адрес и порт брокера, где размещена входящая очередь
- input_login и input_password – логин и пароль, которые EKG Provider будет использовать для авторизации на брокере (в настоящее время поддерживается авторизация только для RabbitMQ, доступ к брокеру Kafka должен быть без пароля). Если логин и пароль не требуются, вместо них в команде надо указать _
- input_queue – имя прослушиваемой очереди или топика
- output_host, output_port, output_login, output_password, output_queue – аналогичные параметры для очереди, в которую EKG Provider будет помещать ответные сообщения.

Для удаления настроек прослушиваемой очереди выполните команду:

```
ekg-provider delete queue [name]
```

# 10. Логирование и мониторинг

EKG Provider записывает лог-файл /log/ekg.log (путь относительно корневой папки контейнера). Подкаталог log файловой системы контейнера желательно смонтировать в общее файловое хранилище. Также рекомендуется реализовать сбор логов из этих файлов с выводом в ELK, особенно при развертывании в Kubernetes, когда процесс ekg-provider запускается в нескольких экземплярах, и каждый экземпляр ведет свой собственный лог-файл.

Каждая строка лога представляет собой JSON, имеющий следующие ключи:

- Time – время события в формате timestamp
- Readable_time – время события в читаемом формате
- Code – HTTP-код типа события: 200 для нормальной обработки запросов, 503 для ошибок, 403 для событий отказа в связи с отсутствием прав доступа, 404 для обращений на несуществующие URL.
- Message – текст сообщения
- Extra – дополнительная текстовая информация (например, пояснение текста ошибки)
- Value – дополнительная числовая информация (например, PID процесса, в котором произошла ошибка)

EKG Provider имеет HTTP-интерфейс для сбора метрик в Prometheus, доступный по адресу /metrics веб-сервера. Он предоставляет следующие метрики:

- average_time – шкала (gauge), среднее время выполнения запроса. Метрика собирается в разрезе url API (entity, entities, model и т.д.) и типа метода (get, put, delete)
- rps – шкала (gauge), среднее число обработанных запросов в секунду. Собирается в тех же разрезах
- errors_counter – счетчик (counter), общее число возникших ошибок
- errors_rps – шкала (gauge), среднее число возникших ошибок в секунду.

Существуют также URL /health/live и /health/ready, которые могут использоваться для настройки liveness probe и readiness probe в Kubernetes. Метод /health/live отвечает HTTP-кодом 200, если сервис запущен. Метод /health/ready отвечает HTTP-кодом 200, если сервис запущен и связь со служебной базой данных Postgres доступна, и кодом 503 в ином случае.

На рис. 4 показаны примеры дэшбордов в Grafana с показателями функционирования EKG Provider.

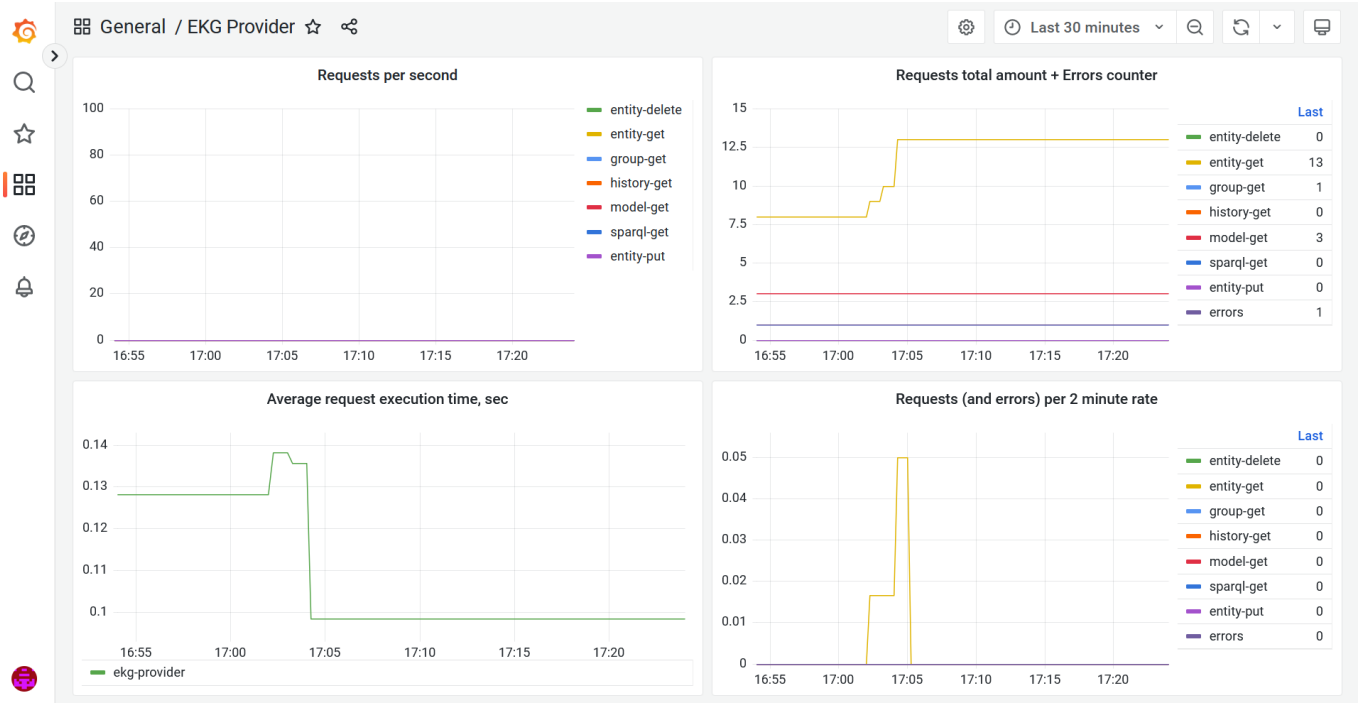



Рис. 4. Пример дэшборда в Grafana с показателями функционирования EKG Provider

# 11. Работа с данными с помощью EKG Explorer

DataVera EKG Explorer – компонент платформы, предназначенный для работы пользователя с содержимым корпоративного графа знаний. Для авторизации и аутентификации пользователей в этом веб-приложении используется KeyCloak. Перед началом работы необходимо создать группу(ы) пользователей KeyCloak, поставить каждой группе в соответствие систему-клиент EKG Provider. Для этого необходимо в панели администрирования KeyCloak выбрать группу, перейти на вкладку Attributes, создать атрибут с ключом datavera и значением: { "priority": 10, "client": "system" }. После этого пользователи KeyCloak, принадлежащие к этой группе, будут работать с данными EKG Provider с помощью интерфейса EKG Explorer с правами, соответствующими клиентской системе, чей идентификатор задан в ключе client.

## 11.1. Навигация и основные операции с данными

Общий вид главного окна интерфейса EKG Explorer показан на рис. 5.

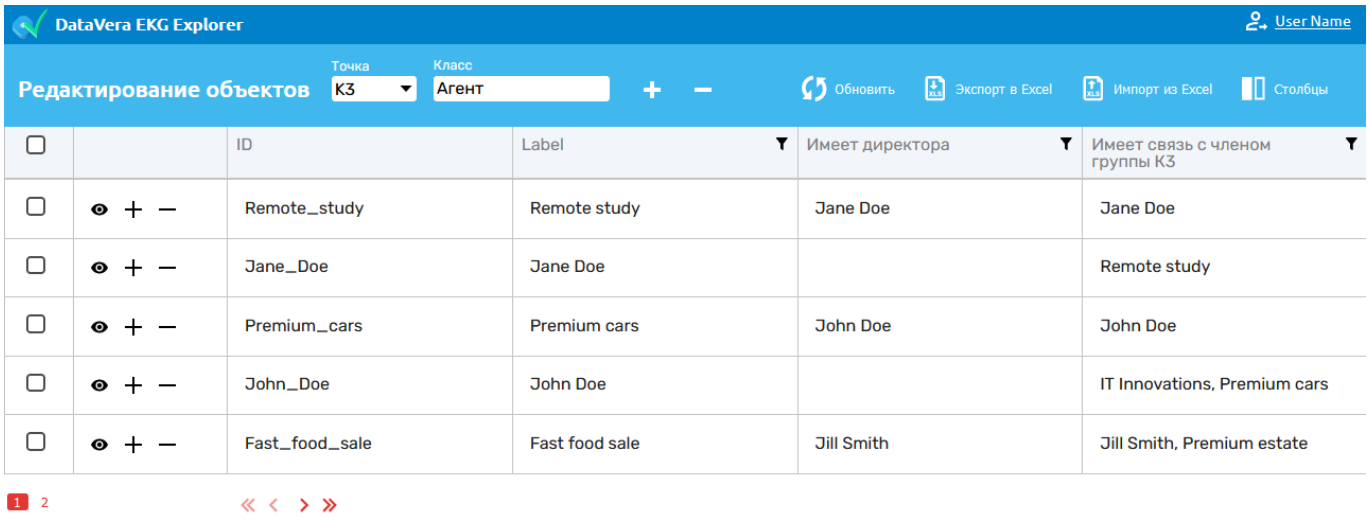



Рис. 5. Общий вид интерфейса DataVera EKG Explorer

Работа в интерфейсе начинается с выбора точки доступа и класса. Точка доступа выбирается из списка точек, доступных текущему пользователю в соответствии с его правами, из выпадающего меню «Точка». Класс выбирается во всплывающем диалоговом окне, вид которого показан на рис. 6:

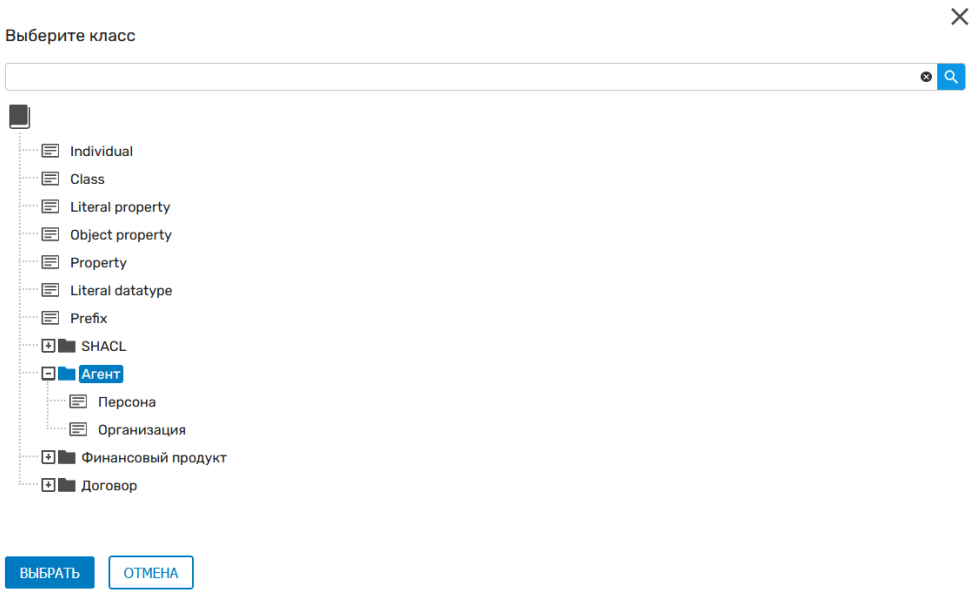



Рис. 6. Диалоговое окно выбора класса

После выбора класса в списке в основной части страницы отображается список индивидуальных объектов этого класса. Каждая строка этого списка соответствует одному объекту данных.

Справа от органа выбора класса над списком расположены кнопки «+» и «-». Нажатие на кнопку «+» добавляет в список новую строку, что позволяет создать новый объект данных. Нажатие на «-» позволяет удалить объекты, отмеченные переключателем в левой части строки.

В левой части каждой строки также расположены три кнопки. Кнопка с пиктограммой «глаз» открывает окно просмотра свойств объекта, которое описывается далее. Кнопка «+» позволяет добавить строку для создания нового объекта данных после этой строки. Кнопка «-» предназначена для удаления объекта данных, соответствующего этой строке. Далее в таблице следуют столбцы, каждый из которых соответствует какому-либо свойству объектов класса.

В правой части страницы над списком расположено меню, состоящее из нескольких кнопок:

- Обновить
- Экспорт в Excel
- Импорт из Excel
- Столбцы

Нажатие на кнопку «Обновить» позволяет заново запросить данные из EKG Provider – например, если они изменились в результате импорта туда данных через API. Работа остальных кнопок рассматривается далее.

В самой верхней строке интерфейса EKG Explorer отображаются органы управления для выбора языка интерфейса, языка данных и времени, на которое просматриваются данные:

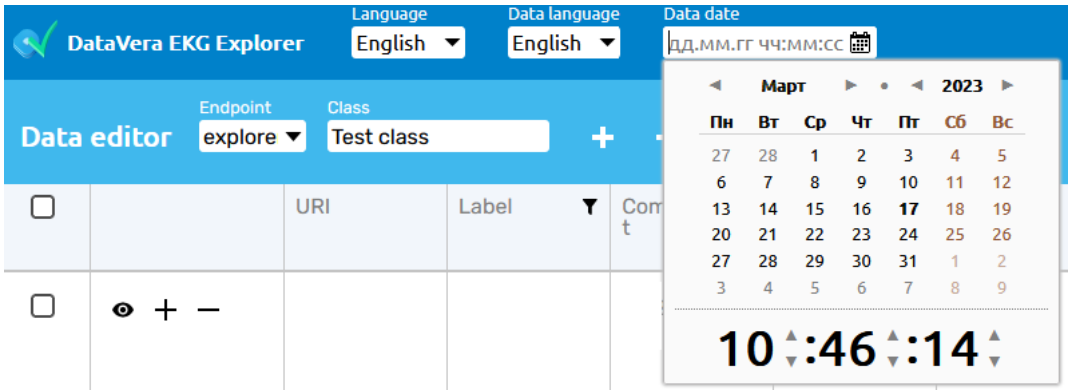



Рис. 7. Органы управления в верхней части страницы

Выбор языка интерфейса влияет только на то, на каком языке отображаются подписи элементов интерфейса.

Выбор языка данных влияет на отображение и редактирование данных. Будут отображены значения строковых свойств объектов и названия ссылочных объектов на выбранном языке, а также значения, для которых язык не указан. При редактировании все строковые значения, записываемые в хранилище, получат метку языка, соответствующую выбранному языку данных.

При выборе даты, на которую просматриваются и редактируются данные, в таблице будет отображаться не текущее состояние объектов данных, а то, которое было актуально на выбранную дату (работа системы при этом замедлится из-за необходимости извлечения исторической информации). При сохранении данных в режиме с установленной датой –

изменяется текущее состояние объектов в базе данных, но запись об истории изменений формируется на ту дату, которая выбрал пользователь.

### 11.2. Операции со списком объектов

Щелчок по заголовку любого столбца таблицы позволяет отсортировать объекты по значению свойства, содержащегося в этом столбце. После того, как сортировка установлена, в правой части заголовка столбца появится треугольник, показывающий направление сортировки. Повторный щелчок по заголовку того же столбца позволяет отсортировать список в обратном направлении.

В заголовке каждого столбца расположена иконка фильтра. При нажатии на нее отображается поле для ввода значения. Если ввести сочетание искомых символов и нажать кнопку Enter, список будет отфильтрован так, что в нем останутся только объекты, в значениях свойства которых присутствует введенное значение.

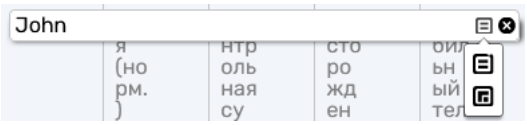



Рис. 7а. Выбор операции фильтра

В раскрывающемся поле для установки фильтра справа расположены пиктограммы, с помощью которых можно выбрать операцию сравнения. Для строковых свойств это точное равенство или поиск по подстроке.

Кнопка выбора столбцов в правой части страницы над списком позволяет вызвать диалоговое окно, в котором можно отметить, столбцы для каких свойств нужно вывести в списке (см. рис. 8). Перетаскиванием элементов в этом диалоговом окне можно изменить порядок столбцов. В списке отображаются все свойства, применимые к объектам выбранного класса согласно модели (TBox).

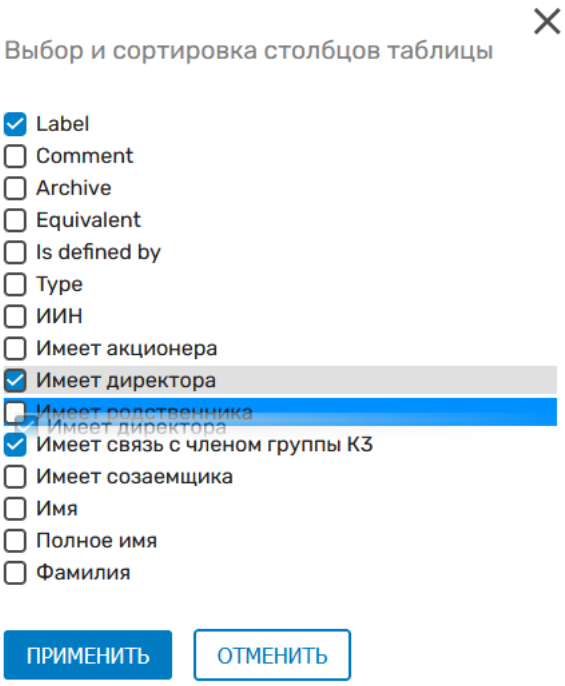



Рис. 8. Окно выбора столбцов списка

Над списком объектов расположено меню-сэндвич, пункты которого предназначены для выполнения операций со всеми объектами текущего класса.

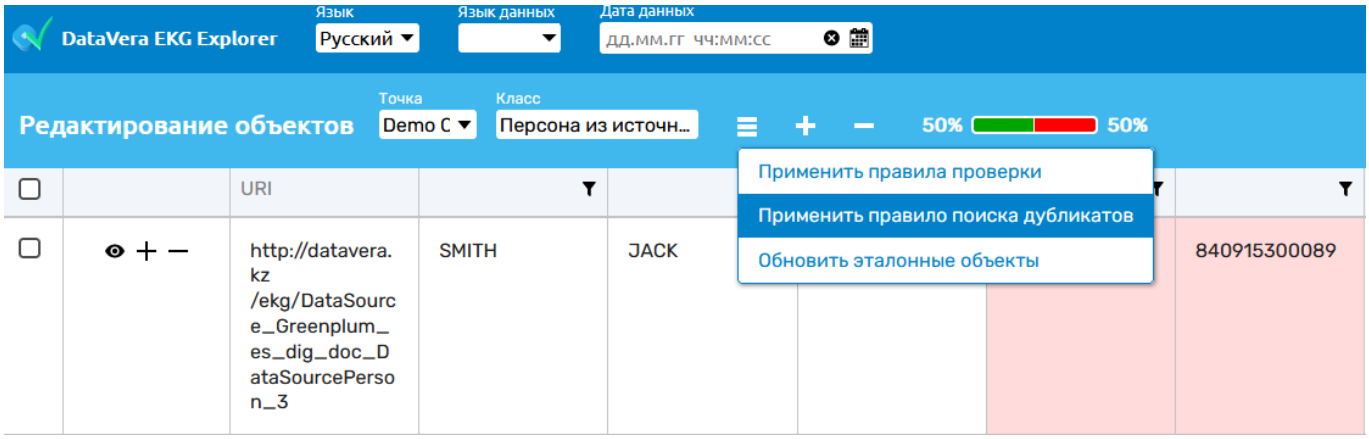



Рис. 9. Меню операций с объектами класса

Пункт «Применить правила проверки» позволяет применить правила контроля качества ко всем объектам класса. Это может потребоваться в случае, если объекты класса были загружены из Excel операцией массовой загрузки – в ходе такой операции правила не применяются.

Пункт «Применить правило поиска дубликатов» позволяет обновить связи типа «Является дубликатом» между объектами текущего класса. После нажатия на этот пункт меню система предлагает выбрать конкретное правило поиска дубликатов. Затем оно применяется ко всем объектам класса, если конфигурация хранилища данных позволяет это сделать (см. раздел «Режимы применения правил»).

Пункт «Обновить эталонные объекты» запускает создание или обновление эталонных объектов для всех групп объектов, связанных связью «Является дубликатом». Выполнение этой операции возможно, если для класса определено правило консолидации данных (см. раздел «Консолидация данных – формирование эталонных объектов»).

## 11.3. Просмотр качества данных

Если для объектов какого-либо класса настроены правила контроля качества данных, то при просмотре объектов этого класса над списком будет отображен индикатор качества.

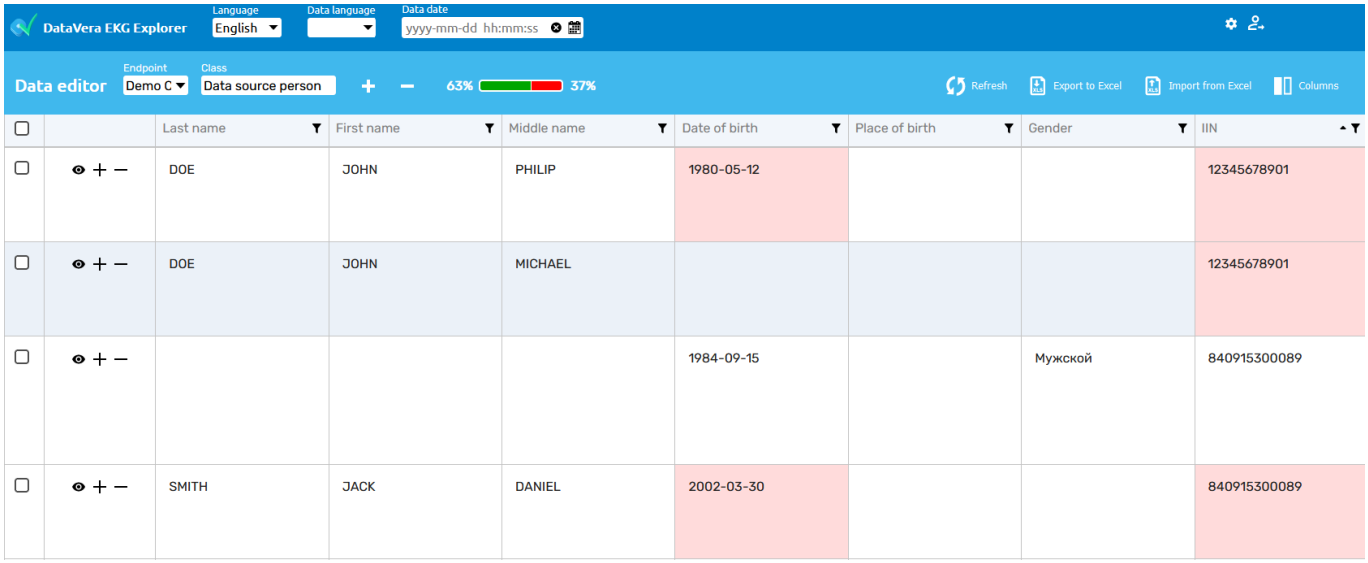



Рис. 10. Просмотр качества данных

Зеленая часть индикатора показывает процент объектов класса, не имеющих нарушений правил контроля качества, красная часть – процент объектов, для которых существует хотя бы одно нарушение. Нажатие на индикатор позволяет отфильтровать список, оставив в нем только объекты с нарушениями правил контроля – этот отбор дополнит уже установленные фильтры.

В списке ячейки, содержащие значения свойств объектов, для которых существуют нарушения контроля качества, подсвечены красным цветом. Также сведения о нарушении качества отображаются в диалоговом окне при просмотре свойств конкретного объекта.

## 11.4. Операции с записями списка. Редактирование значений

Добавить новую строку в список можно нажатием на кнопку «+» над списком или в любой существующей его строке. Появится новая строка, в которой отобразится идентификатор для сгенерированного объекта:

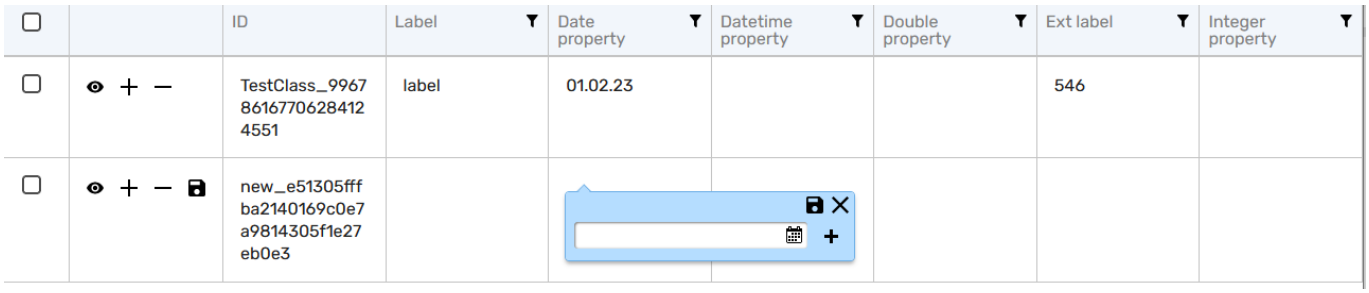



Рис. 11. Добавление нового объекта в список

Если среди свойств объекта есть обязательные для заполнения, т.е. имеющие minCardinality = 1 или более, то в списке сразу появятся всплывающие элементы для ввода или выбора значений этих свойств. Вызвать такое окно для любого другого свойства можно двойным щелчком по ячейке таблицы.

Всплывающее окно для ввода значений может иметь разный вид в зависимости от типа значений поля. Например, для полей типа «дата» оно содержит орган «календарь» для выбора даты. Если свойство является ссылкой на объекты другого класса, во всплывающем окне будет отображен подборщик для выбора объекта, и т.д. В любом случае важно отметить, что всплывающее окно позволяет ввести или выбрать сразу несколько значений для любого свойства каждого объекта, что является особенностью онтологической модели данных. Если какое-либо свойство может иметь только одно значение, это нужно указать явно на уровне модели данных, задав этому свойству maxCardinality = 1.

Нажатие на кнопку «+» справа от значения свойства позволяет добавить новое значение, как показано на рис. 12. Нажатие на кнопку «-» удаляет значение из списка.

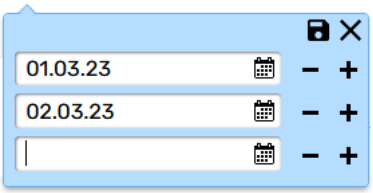



Рис. 12. Окно редактирования значений свойств объекта

По окончании редактирования значений свойств необходимо нажать на кнопку сохранения во всплывающем окне или в строке в целом.

Некоторые значения строковых свойств могут быть заданы на определенном языке. Метка языка выводится слева от значения:

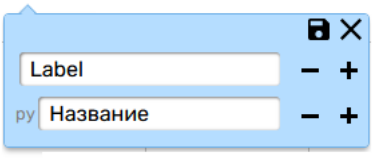


Рис. 13. Метка языка у значения строкового свойства

Чтобы сохранить значение строкового свойства на определенном языке, нужно выбрать язык данных в верхней части экрана EKG Explorer. Изменить язык уже созданных значений нельзя.

Свойства объектов могут иметь не только значения, присвоенные явно, но и получать значения в результате работы правил логического вывода. Такие значения отображаются в списке объектов серым цветом и не доступны для редактирования:

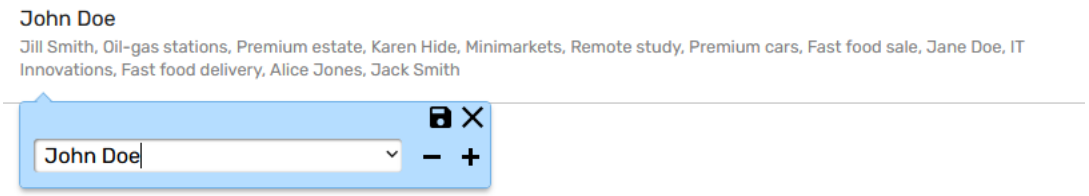



Рис. 14. Отображение значений, полученных правилами логического вывода

При сохранении объектов значения выведенных свойств могут пересчитываться автоматически.

### 11.5. Экспорт и импорт объектов в Excel

Кнопка «Экспорт в Excel» скачивает и открывает Excel-файл с объектами, список которых пользователь просматривал в приложении. Список в Excel имеет такой же формат и структуру, как и в приложении, т.е. на него распространяются настройки выбора столбцов, сортировки и фильтрации.

Можно отредактировать значения свойств объектов в файле Excel и загрузить их обратно в EKG Explorer. Для этого надо нажать кнопку «Импорт из Excel». Появится диалоговое окно, в котором нужно выбрать файл с измененными объектами (менять состав столбцов файла нельзя! Можно только редактировать в нем значения свойств объектов), и нажать кнопку «Импортировать». Переключатель «обновлять/замещать» определяет способ применения изменений к объектам в базе данных: при выборе варианта «обновлять» значения свойств объектов в системе будут дополнены значениями, содержащимися в файле, при этом существующие значения свойств, отличные от тех, что указаны в файле, будут сохранены. Вариант «замещать» позволяет полностью заменить свойства объектов теми, что содержатся в файле. В любом случае импорт не затронет значения тех свойств, для которых в файле нет столбцов.

Для свойств, имеющих ссылочный тип, при экспорте в качестве значения в ячейку подставляется идентификатор объекта, за которым в скобках идет название объекта. При импорте имеет значение только идентификатор, т.е. название объекта в скобках при импорте можно не указывать.

Если свойство имеет несколько значений, при выгрузке в Excel разные значения выделяются цветом, как показано на рис. 15:

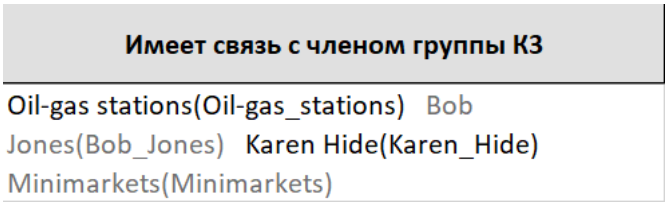



Рис. 15. Результат выгрузки в Excel свойства, имеющего несколько значений

Конкретный цвет, которым указаны значения свойств, не важен. При подготовке файла для импорта можно выделить цветом значения свойств так, как удобно пользователю.

Выведенные значения свойств не экспортируются в Excel.

### 11.6. Просмотр свойств объекта

Кнопка 👁 в строке с каждым объектом позволяет открыть диалог просмотра его свойств.

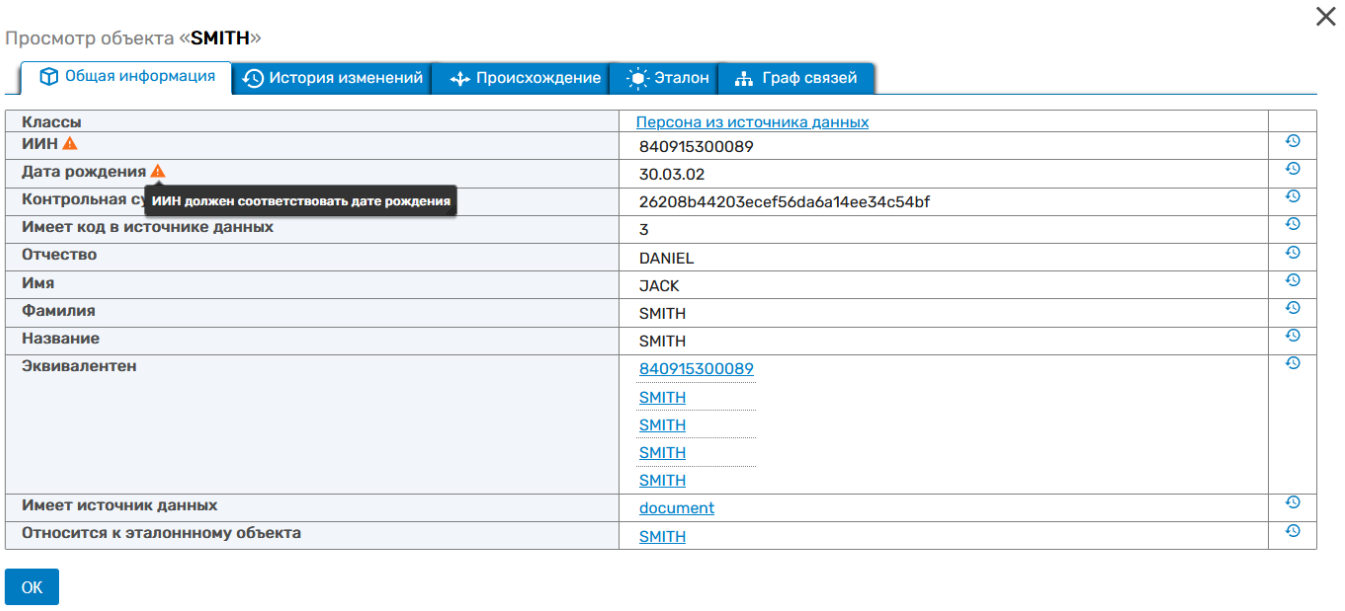



Рис. 16. Окно просмотра свойств объекта

На вкладке «**Общая информация**» в этом окне отображается принадлежность объекта классам, а также значения всех его свойств. Напротив каждого свойства отображается пиктограмма, нажатие на которую вызывает диалоговое окно просмотра истории значения свойства. Также отображаются нарушения ограничений, заданных правилами.

Выведенные свойства объекта отображаются в карточке серым цветом. Напротив каждого такого значения свойства отображается название правила, в результате работы которого получено значение:

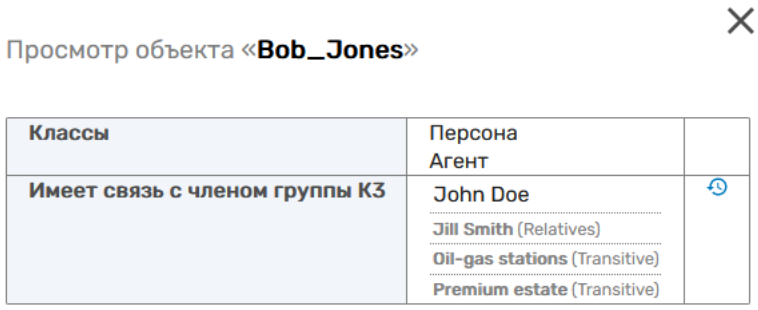



Рис. 17. Выведенные свойства объекта

На вкладке «**История изменений**» отображается полная история редактирования этого объекта данных.

Вкладка «**Происхождение**» отображает сведения о происхождении объекта данных, которые формируется для эталонных объектов. Здесь можно увидеть, из каких объектов исходных данных были получены значения тех или иных свойств эталонного объекта.

На вкладке «**Эталон**» для объектов из источников данных отображается ссылка на эталонный объект, если он существует, а также ссылки на объекты, связанные с текущим объектом связью «Является дубликатом». Для каждого такого объекта отображается его источник и идентификатор в нем. Внизу окна расположена кнопка «Консолидировать», нажатие на которую позволяет создать или обновить эталонный объект для группы дубликатов, в которую входит текущий объект.

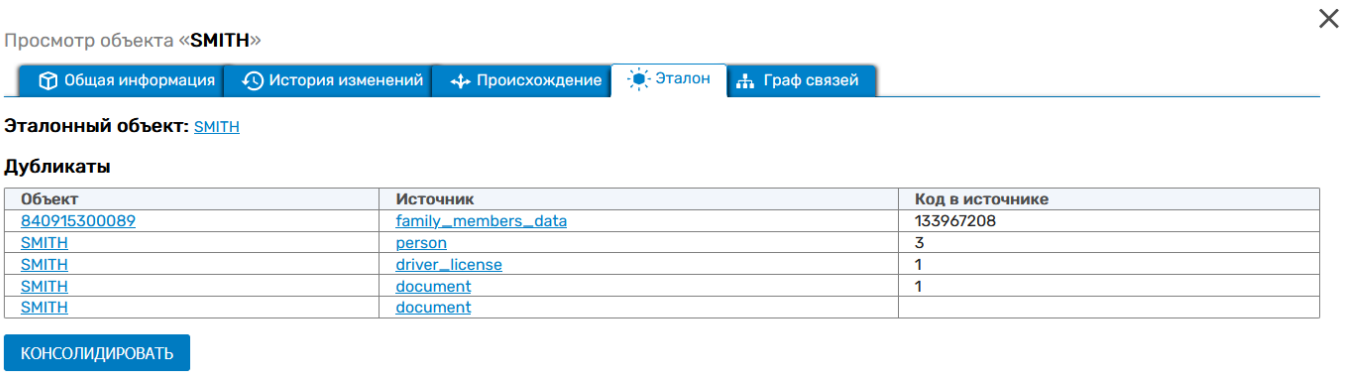



Рис. 18. Просмотр группы дубликатов и эталонного объекта

Вкладка «Граф связей» отображает связи текущего объекта с другими объектами. Пиктограммы вверху страницы позволяют выбрать глубину отображения – один, два или три уровня. Система старается оптимально расположить элементы на диаграмме, однако при необходимости можно изменить их положение на холсте путем перетаскивания мышью.

Обратим внимание, что диаграмма отображает связи конкретных объектов данных (ABox), а не уровень концептуальной модели (TBox). Алгоритм отображения графа выбирает все связи, ведущие к текущему объекту или от него, и отображает первый уровень связей. Затем при

необходимости отображение связей повторяется для каждого связанного объекта на втором или третьем уровне. При большом числе связанных объектов и уровней отображения диаграмма может выглядеть запутанной – тогда нужно уменьшить глубину отображаемых связей.

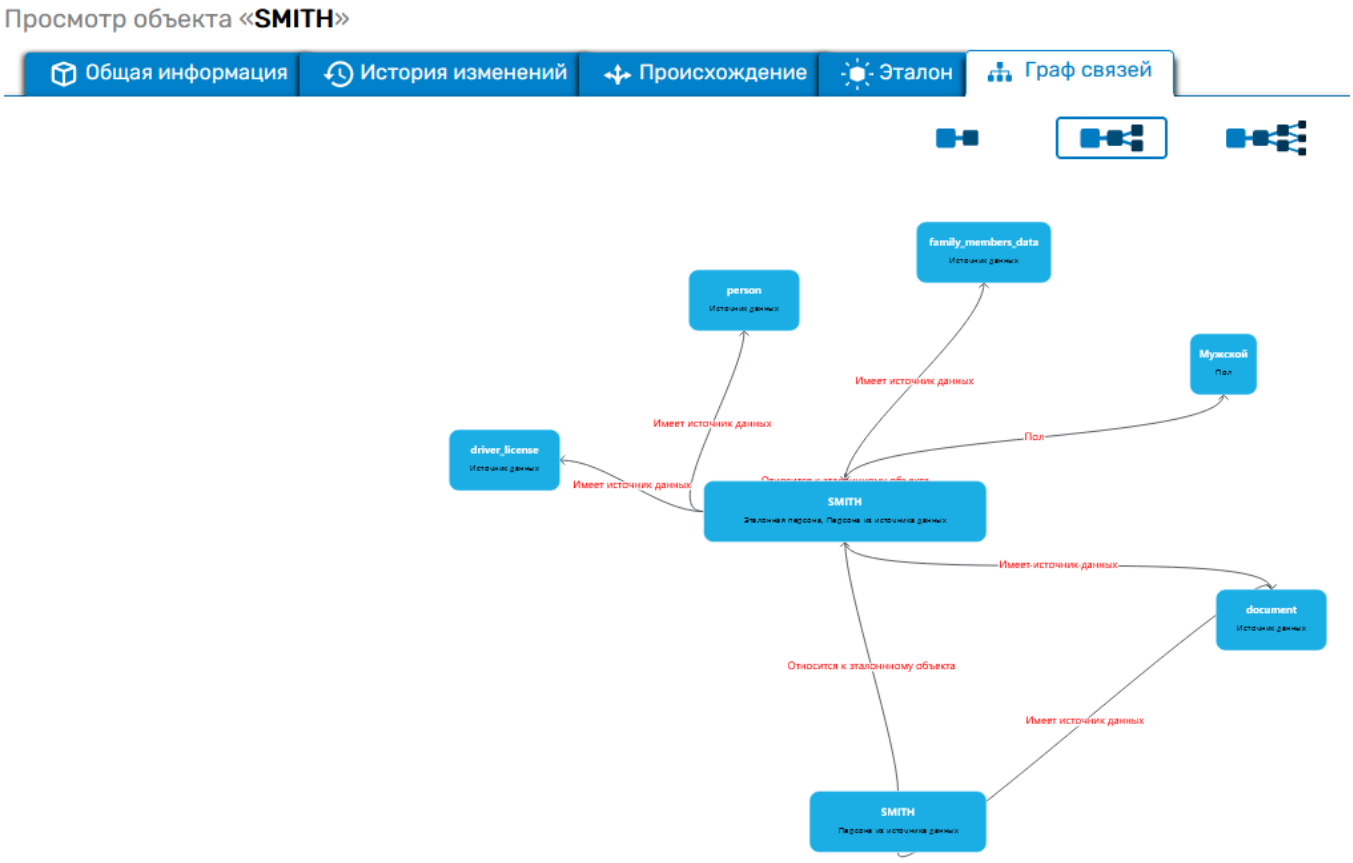



Рис. 19. Граф связей объекта