DataVera

DataVera EKG Platform

Data Virtualization Platform

User Guide

v 1.17

Content

1. EKG Platform basics

DataVera EKG Platform is a data virtualization platform designed to represent large amounts of information in the form of an **Enterprise Knowledge Graph** (**EKG**). An enterprise knowledge graph is a coherent, structured representation of information in which each information object is represented by the vertex of the graph, and each property value is represented by the edge of the graph.

The structure of information in EKG is defined in an ontological model, or **ontology**. Ontological models are built in accordance with the specifications of the W3C Consortium: RDF, RDFS, OWL. Ontology allows you to define the entity types (classes) and the properties whose values the objects of these classes can have. The class and property definitions level in an ontology is known as a TBox (Terminology Box). The information about specific objects is expressed using a given set of classes and properties at the ABox (Assertions Box) level.

Schematically, the structure of the ontological enterprise knowledge graph content is shown in Fig. 1.

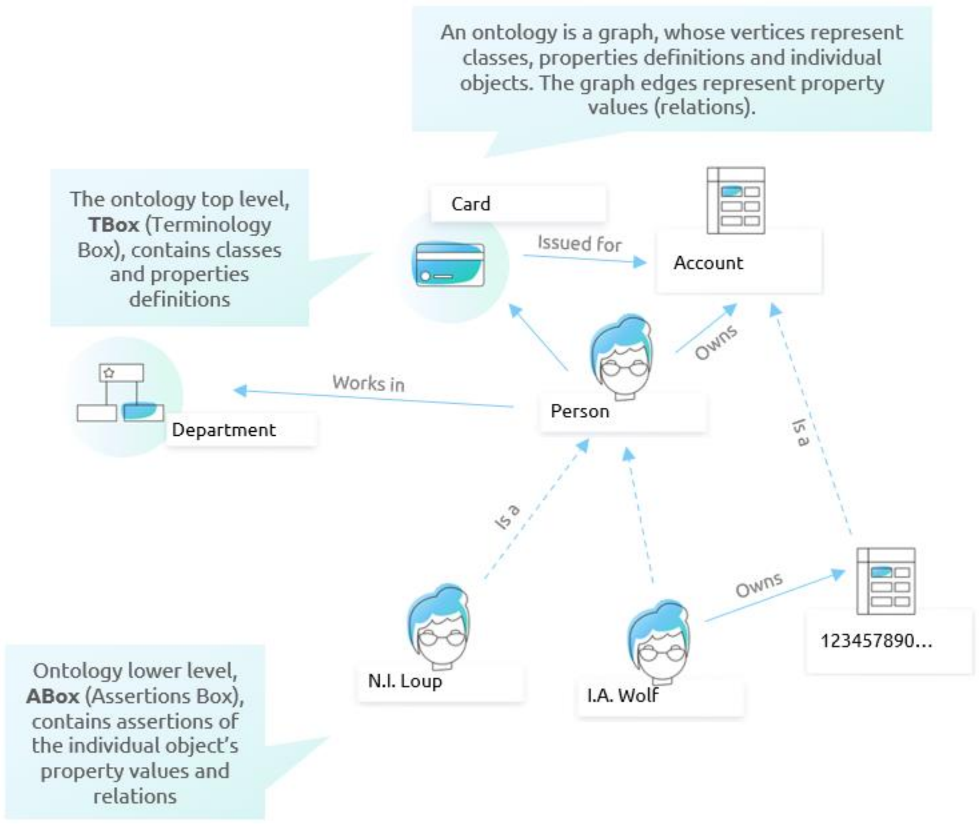

An ontology is a graph, whose vertices represent classes, properties definitions and individual objects. The graph edges represent property values (relations).
The ontology top level, **TBox** (Terminology Box), contains classes and properties definitions
Card
Issued for
Account
Owns
Is a
Person
Works in
Department
Is a
Owns
123457890...
N.I. Loup
I.A. Wolf
Ontology lower level, **ABox** (Assertions Box), contains assertions of the individual object's property values and relations

Fig. 1. Example of the logical structure of the enterprise knowledge graph

A knowledge graph can be built without using an ontology – there are other ways to do this. You can create an ontology to describe a domain, but not use it to create an enterprise knowledge graph. However, by combining the ontological modeling methods and specifications with the enterprise knowledge graph paradigm, you can achieve the greatest flexibility in managing your organization's data. This guide describes how to use DataVera tools to implement solutions in this area.

The natural way to store information represented in the form of a graph is a Graph Database. There is a Graph DBMSs class specifically designed to work with ontological models – **RDF triple stores**. The **SPARQL** query language (also a W3C specification) is developed to work with the content of such DBMSs. Triple stores with SPARQL support are convenient for working with data represented in an ontology, but they are significantly slower than other DBMS types in the large data amounts processing. Graph DBMS can be used to create analytical repositories of information but are not very suitable for storing operational or transactional data. At the same time, in corporate data management, the processing of operational data is a priority. To take full advantage of the flexibility of ontological data models and Graph DBMS analytical capabilities, without losing the reliability and speed of relational and document-oriented databases, it is necessary to use **data virtualization** technology. Its principle in the context of the tasks described in this guide is that **data must be physically distributed among a variety of "traditional" (primarily relational) DBMSs** to ensure reliability and performance, but **logically they should be represented as a graph**, which structure is determined by the ontological model. This is the way DataVera EKG Provider, one of the EKG Platform components, works with data.

Data Virtualization

Information is physically distributed in several relational DBMSs,
but logically it is represented as a graph. The virtual graph structure is described by an ontology.

EKG Provider offers an application programming interface (API) through which information providers and consumers can interact with data without worrying about where and in what structure it is physically stored. The API allows you to work with data as if it were in a graph and corresponds to the structure of the ontological model – including the use of the SPARQL language to access it. **TBox, the model structure**, is indeed stored in a graph DBMS (we recommend using Apache Fuseki, but it is possible to use other triple stores). **The actual data, ABox**, can be partially located in a graph DBMS (usually the frequently used enumerations contents remain in it) or in a variety of relational DBMSs, of which Postgresql is a best choice. Multiple Postgresql DBMSs or clusters can be plugged in to a single EKG Provider instance as data storages. The large object sets, transactional information, time series, and other types of data that are significant in volume but relatively uniform in structure can be effectively projected into relational DBMS.

A graph that reflects an ontological model can include not only the data itself and a description of its structure, but **the data processing rules** also. The most modern W3C specification that describes the rules are representation mode is SHACL. The EKG Provider supports the SPARQL Rules described in this specification. Rules are represented as objects of the ontology classes defined in the specification and residing in a triple store.

The EKG Provider settings determine in which DBMS instances the various classes objects are stored. The relationship of the EKG Provider with other components necessary for its operation is shown in Fig. 2.

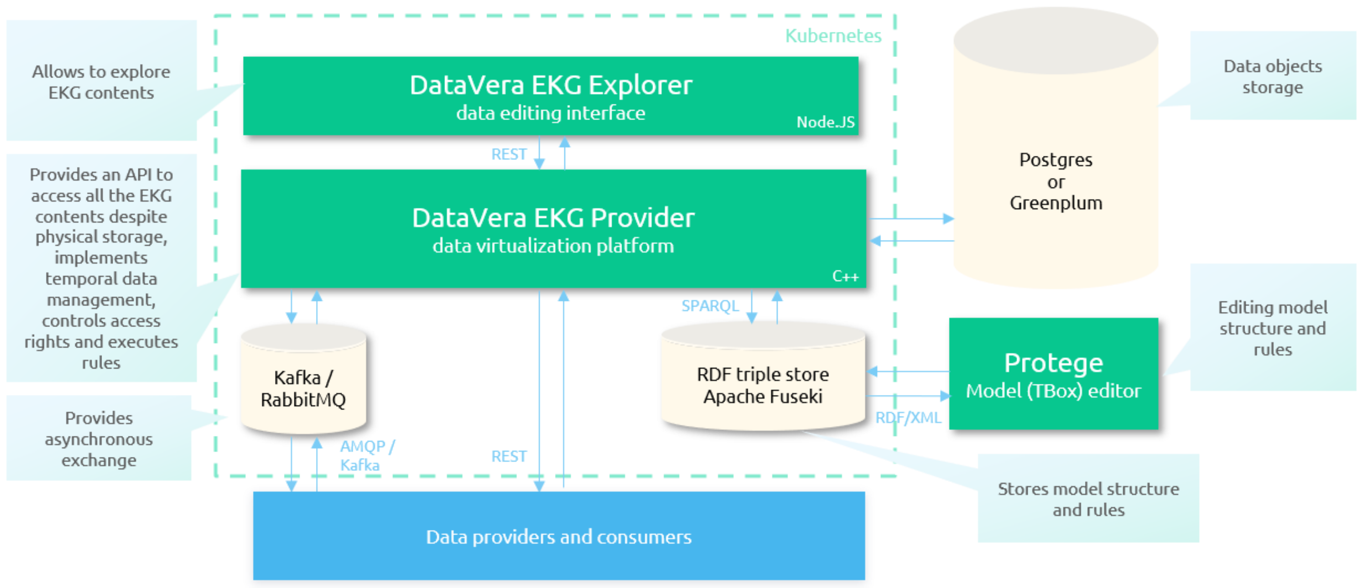

Allows to explore EKG contents
Provides an API to access all the EKG contents despite physical storage, implements temporal data management, controls access rights and executes rules
Provides asynchronous exchange
Kubernetes
DataVera EKG Explorer
data editing interface
Node.JS
REST
DataVera EKG Provider
data virtualization platform
C++
SPARQL
Kafka / RabbitMQ
RDF triple store Apache Fuseki
AMQP / Kafka
REST
Data providers and consumers
Postgres or Greenplum
Data objects storage
Protege
Model (TBox) editor
Editing model structure and rules
RDF/XML
Stores model structure and rules

Fig. 2. Interaction of EKG Provider with other components of the software environment

DataVera EKG Platform can perform the following **Data Governance** tasks in an organization's IT infrastructure:

1. MDM system functions: storing reference information and master data. Consolidation of information about the same business process objects from different information sources, cleaning, normalization and deduplication, distribution of reference data to consumer systems using a subscription mechanism. Solving data integration tasks. An example of how to do this is shown below in the "Subscriptions" section.
2. The logical data control according to customizable rules, data integrity control, rules-based data augmentation.
3. Storing of the history of all data objects transformations (Data provenance).
4. Storing a history of all states of each data object (data temporality), using it to retrieve data objects that meet certain conditions as of any given time.
5. Storing an analytical information consolidated from a variety of applied automated systems and repositories. Implementation of a Knowledge Management System – a single point of access to any corporate information. An example of how to do this is shown in the "SPARQL endpoint" section below.
6. EKG Provider can become the core of the data-centric enterprise IT architecture, acting as a single "cloud" data storage for applications that does not have their own DBMS.

Presenting data using an ontological model has several important features and advantages:

- Each object can be a member of several classes (types) at the same time. For example, the object "I. Wolf" can be simultaneously included in the "Person" and "Individual entrepreneur" classes.

- Classes can form one or more hierarchies according to the superclass-subclass (superset-subset) principle. For example, the "LLP" class may be a subclass of the "Legal Entity" class.
- Properties can also form hierarchies. For example, the Registration Address property can be a sub-property for the Address property.
- Each property can be applicable to objects of different classes. For example, both individuals and organizations can have the value of the Capital Amount property, which allows to compare, filter, and sort objects of different types using the same properties.
- Each property of each object can have multiple values at the same time. For example, an organization might have multiple addresses or phone numbers.
- Each string property can have multiple values set in different languages. For example, the object "Kazakhstan" can be named in Kazakh and English languages.

All these and other advantages are available when working with data and a model using the EKG Provider API, despite the fact that physically part of the information is located in relational DBMSs.

2. EKG Provider Application architecture

EKG Provider is a Linux application written in C++. It is delivered in a Docker container and is designed to be deployed in standalone mode or on a Kubernetes cluster. Figure 3 shows the architecture of the EKG Provider components.

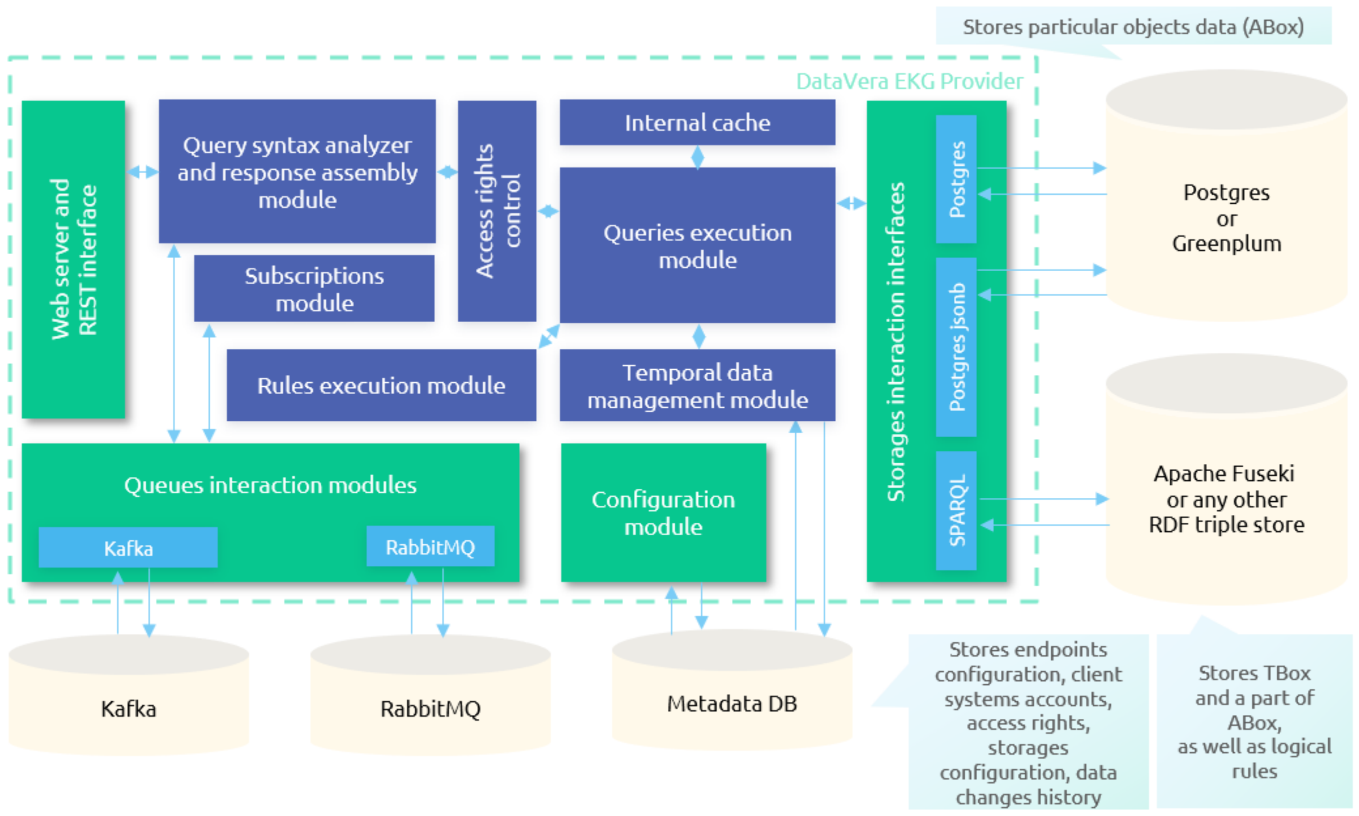

Stores particular objects data (ABox)
DataVera EKG Provider
Web server and REST interface
Query syntax analyzer and response assembly module
Access rights control
Internal cache
Queries execution module
Subscriptions module
Rules execution module
Temporal data management module
Storages interaction interfaces
Postgres
Postgres jsonb
SPARQL
Postgres or Greenplum
Apache Fuseki or any other RDF triple store
Queues interaction modules
Kafka
RabbitMQ
Configuration module
Kafka
RabbitMQ
Metadata DB
Stores endpoints configuration, client systems accounts, access rights, storages configuration, data changes history
Stores TBox and a part of ABox, as well as logical rules

Fig. 3. EKG Provider Component Architecture

At startup, the ekg-provider executable starts several processes:

- A web server that serves the REST API and the SPARQL endpoint. It generates a new process to handle each incoming request (or several processes, if the necessary data is distributed in several storages)
- Listening process for each incoming Kafka or RabbitMQ message queue
- Internal service for sending notifications about changes in data objects to subscribers
- A parent process that monitors changes to the global settings of the application and restarts child processes if necessary.

3. Ontological data models technical aspects

To understand the API structure, we need to provide some information about how knowledge is expressed in ontological models. We recommend that you familiarize yourself in more detail with the ontological modeling methods and technologies using one of the textbooks or manuals available on the Internet.

The entity identifiers in the ontology are IRI, i.e. the object identifier is an expression of the type {prefix}#{unique part of the identifier}. For example, the identifier of the "Person" class can be http://xmlns.com/foaf/0.1/Person, and the identifier of a specific person is http://xmlns.com/foaf/0.1/Person_Jane_Doe. Each identifier refers to a specific namespace, which is called a "prefix". For example, the common FOAF (Friend of a friend) ontology, an example of which identifier is shown above, uses the http://xmlns.com/foaf/0.1/ prefix. All objects in this ontology have an IRI beginning with this expression. There is an abbreviated prefix notation, for example, the foaf:Person record is equivalent to http://xmlns.com/foaf/0.1/ Person.

Each ontology can have a default prefix. In EKG Provider, you can specify it for each access point, i.e. a separate graph. You can omit this prefix when referring to the object identifiers. For example, if the ontology has the default prefix http://datavera.kz/model, object identifier http://datavera.kz/model/Asset will be equivalent to an Asset record.

The author of the ontology can select namespaces and assign arbitrary identifiers to the ontology entities will. However, to ensure interoperability, it is recommended to use common entity identifiers such as org:Organization or sosa:Sensor specified in the W3C specifications and commonly used ontologies. In the same ontology, identifiers from different namespaces can be used together. This does not have any consequences as for a DBMS any IRI is just a string. You can use any namespace for self-defined identifiers. Individual object identifiers can be assigned arbitrarily. We recommend using abstract values for them that are not related to the properties of the object that can change over time, such as a guid or any hash.

The W3C specifications that define how to express ontologies (RDF, RDFS, OWL) also have their own namespaces. These spaces define identifiers for basic entity types and some properties. For example, four main entity types are defined:

- owl:Class
- owl:ObjectProperty – property that is a reference from one object to another
- owl:DatatypeProperty is a property whose value is literal (string, number, etc.). XSD types are used, such as xsd:string, xsd:integer, and so on.
- owl:NamedIndividual is an individual object that typically belongs to one or more classes and has property values.

Among the standard properties (predicates) it is important to mention:

- rdf:type – the relationship of an entity to type. Used to specify the "primary" type of each entity in the list above, and to indicate that an individual object belongs to specific classes.
- rdfs:label – readable entity name
- rdfs:subClassOf – the relationship between a subclass and a superclass
- rdfs:domain – a set of classes whose objects can have the value of a property (property scope)
- rdfs:range – property value range

All requests to the EKG Provider API can specify URIs in either full or abbreviated format. EKG Provider responses return only a unique portion of the URI for items whose URI prefix matches the default prefix. In the other cases, the full URI is returned.

4. EKG Provider features overview

EKG Provider provides the following functions groups available using the API:

- Retrieving Data Objects
  - A specific object by known identifier
  - Object groups by set of conditions
- Retrieving the Structure of an Ontological Model TBox, a set of Classes and Properties
- Working with temporal information
  - Get data object state for a specific point in time
  - Retrieve the changes history for a specific data object
  - Get a set of objects that met the conditions of the specified filters at a particular point in time
- Create a data object or change its property values
- Bulk load data objects
- Delete a data object
- Apply the constraints and inference rules to the data object
- Subscribe to receive messages about changing/deleting objects of certain classes, change the subscription properties
- Get a list of active subscriptions
- Delete a subscription

An SPARQL endpoint interface is also available, which in the current version only supports SELECT data retrieval queries with some limitations.

The main API functions are designed primarily for performing operations on data that are typical for traditional DBMS: CRUD operations (Create, Update, Delete), as well as working with data in subscription mode and working with temporal data. In all cases, these functions are working with individual data objects or with groups of homogeneous objects.

The SPARQL interface is designed to perform operations that are more typical for graph DBMS: searching for the graph patterns, which are sets of elements related in a certain way. Using the SPARQL endpoint, it is convenient to detect relationships between heterogeneous objects or build long chains of connections. Examples of such requests are given below in the "SPARQL endpoint" section.

The main API is designed primarily for the master and operational data management tasks, and the SPARQL endpoint is intended for analytical processing of complex interrelated data.

Data retrieval queries can combine information of objects from different physical stores. Data change requests can handle the situation of moving an object from one store to another due to a change in its classes membership or storing an object in multiple stores at once.

At the EKG Provider configuration level, the following settings are defined:

- A set of "endpoints" – separate data sets, access to which is provided by the EKG Provider
- A set of data physical storages and the data objects distribution rules between them, mapping rules which map the ontology structure elements to the elements of the storage internal structure
- Set of client systems
- Access rights of client systems to objects of various classes
- Kafka/RabbitMQ queues through which clients can send asynchronous requests to the API.

The structure of the ontological model in the current version of the platform can be managed using the open source Protégé editor.

Another component of our platform, DataVera EKG Explorer, is intended to manage (search, edit) individual data objects.

5. EKG Provider REST API and interaction via Kafka/RabbitMQ queues

EKG Provider provides two ways to interact with its API:

- Synchronous, using REST API access points or SPARQL endpoint
- Asynchronous, through Kafka queues (recommended) or RabbitMQ.

The functions exposed through the REST API and through queues are exactly the same. The Swagger description of the REST API is available at http://api.k8s.datavera.kz/. Any API method has the following general parameters:

- URL is a method type. There are the following URLs:
    - Entity – working with one entity, one information object
    - Entities – working with a group of objects that meet specific conditions
    - Bulk – data objects bulk loading
    - Validate – applying constraints and inference rules to the data object
    - Model – obtaining TBox, the structure of the ontological model
    - History – working with historical, temporal data
    - Subscription – work with subscriptions
    - Endpoints – obtaining endpoints list
    - Languages – obtaining languages list

  When working with the REST API, it is specified as part of the request URL. When working through queues, it is passed as a "URL" parameter in JSON request body, for example "URL": "entity".
- Method. When accessing the REST API, it is specified as an HTTP method. When working through queues, it is passed as a "Method" parameter in JSON request body, for example, "Method": "POST". The following methods are supported: POST (replaces GET, because any request has a body), PUT/PATCH (equivalent), DELETE.
- An endpoint identifier. An endpoint is a separate set of data within EKG Provider, and you can think of it as a "separate database". Within each access point, there is an ontological model. Data at different access points cannot reference each other. When working with the REST API, it is specified as part of the URL, when working through queues, it is passed in the JSON request body as an "Endpoint" parameter, for example "Endpoint": "demo".
- Client – the identifier of the EKG Provider client system. This is the name of the client account that is used to determine the level of data access rights of the client. Always transmitted in the body of the request, for example "Client": "test".
- RID (optional) – the identifier of the specific request. The same identifier will be contained in the response and allows to map the request to the response when working through queues. Always specified in the JSON packet. Always transmitted in the request body.
- The API Version (when working with the REST API only) is always passed as part of the URL.

The REST service URL has the following structure: /v{Version}/{Endpoint}/{URL}, for example: /v1/demo/entity

The request body is a JSON collection that contains the call "payload", and the structure of this collection is different for each method.

Please contact DataVera to obtain the full version of this guide.

6. Platform configuration API methods

6.1. Endpoints list request

6.2. Obtaining an ontology structure description (TBox)

6.3. Subscriptions

7. CRUD operations

7.1. Single object request

7.2. Querying a Set of Objects

7.3. Evaluated expressions

7.4. Get the object changes history

7.5. Create or modify data object

7.6. Data objects bulk load

7.7. Delete a data object

7.8. SPARQL endpoint

EKG Explorer supports some kinds of SPARQL SELECT requests. The web service format is similar to what Apache Fuseki implements, so you can use the Fuseki web console to execute queries and view EKG Provider responses by replacing the Fuseki address with the EKG Provider's SPARQL endpoint in the service address line.

A POST query /v1/{endpoint}/query with the query text in the "query" parameter returns the response as a JSON collection of two sections. The "head" section with the nested "vars" array contains a list of variables that are part of the query response. The "results" section with the nested "bindings" array contains the result elements (tuples) in the following form: { "type": "uri" | " literal", "value": "variable value" }. SPARQL queries can only be made through the REST interface, not through queues.

Here are some examples of SPARQL requests that the EKG Provider can respond to:

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

In this query, the ?a and ?b variables are specified in the subject and predicate positions. The query will return all objects that refer by any property to an object with an identifier http://datavera.kz/demo/John_Doe (in SPARQL requests, object identifiers must be written in full form, even for objects with a prefix that matches the default prefix).

```
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
```

This query will return Company and Organization classes objects that have the values of the Capital and rdfs:label properties, with the Capital property value greater than 1000, and possibly also having the value of the ParticipatesInProject property. The results will be sorted in descending order of the value of the ?name variable, containing the value of the rdfs:label predicate (the readable name of the object).

As mentioned earlier, the SPARQL query language is useful for finding complex chains of object relationships. Here's an example:

```
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> }
```

Such a query will find all the object chains

```
?obj (hasOwner) -> ?owner (any link) -> John_Doe
```

where is the variable ?obj represents an object of Company class, the ?owner describes the owner of this company (which may belong to classes of legal entities or individuals), provided that the owner has any type of connection with the John_Doe object representing a particular person. Such a request will return all the companies owned by John_Doe relatives, organizations owned by him, etc. Complicating the search terms, it is easy to build an algorithm for finding links between affiliated individuals and legal entities.

The current limited SPARQL implementation supports the following features in the SELECT and ASK queries:

- Using patterns with the variables at any position
- Using UNION for alternative patterns (no nested patterns)
- Using OPTIONAL for optional patterns (no nested patterns)
- Using FILTER. A filter can contain several conditions combined by && and || operators. The nested conditions are not supported. A limited set of functions can be used in filters, such as STR(), CONTAINS(), SUBSTR(), CONCAT() and REGEX().
- Using EXISTS and NOT EXISTS expressions
- Indicating a list of returned variables after SELECT keyword
- Using LIMIT and OFFSET

- CONSTRUCT operation
- SHACL rules – only in the form of SPARQL rules

The following features are currently NOT supported in SPARQL queries:

- Nested patterns
- Property paths
- Blank nodes
- Arithmetic expressions (except those used in the SPARQL functions)
- ORDER BY
- GROUP BY and aggregating functions
- The functions, except those mentioned in the previous list
- DISTINCT keyword
- BIND expression (except its usage for binding the SPARQL function result)
- INSERT and DELETE operations.

Let us describe an example of an IT architecture in which EKG Provider plays the role of a central analytical system. Suppose the company has several automated systems that process structured operational data of business processes, and several file storages containing documents that accompany the pace of these processes. The task is to build a single access point for information search, which will allow to find quickly all the information related to a particular business object or event, identify possible chains of relationships between different objects or events, or solve applied problems, such as searching for affiliates, determining failure chains in technological systems etc.

With this usage mode, EKG Provider will only receive information from various sources, but will not distribute it (unless you need to connect a BI system to analyze collected data). We have to implement data collection procedures that can extract information from structured sources in a way that is convenient for them. If any application systems provide web services for information extraction, data collection procedures can periodically access these services, and then update the relevant information in the EKG Provider. If direct access to databases of structured sources is possible (or to copies of such DBMSs automatically generated specifically for data transfer to the Enterprise knowledge graph), the data collection procedure can use triggers in the DBMS to detect changes, and then transfer changes to EKG Provider using queue managers. If structured data can be obtained from structured files, it is necessary to regularly run the procedure for processing fresh files, which will update the relevant information in EKG.

The steps necessary to achieve this task include:

- Create a set of classes and properties of an ontology, as well as the mapping rules between ontology elements and the information sources data structures. Rules can be stored in a machine-readable form so that there is no need to make changes to the integration procedures code when the structure of the data in the source or the ontology changes. This can be implemented using configuration JSON files, or by storing mapping rules in the ontology itself.
- Create reference data enumerations that are used as property values of objects: client types, hardware types, etc. The mapping tables shall contain the codes of the enumeration elements in the EKG and in each of the information sources.
- Define data consolidation rules, including entity identification rules. The integration procedure will make decisions about how to create or update information objects in EKG based on these rules. For example, the customer information can be stored in several sources. When an integration procedure receives another customer from some source, it will need to search for it in EKG in order to update an existing object or create a new one (each business object must exist in EKG in a single copy). To do this, the procedure needs to rely on a set of key properties of the object, which should be represented in the rules. For example, the search for individuals can be performed by IIN, in its absence – by name and date of birth, if this information is incomplete – by phone number.

The integration procedure on the side of each source of information must use its own identifier of the client system (Client) when working with EKG Provider. This allows to trace the source from which the certain elements of information about each object were obtained when analyzing the populated EKG. For example, the client's phone number can be obtained from system A, and its address from system B. If the integration procedures on the side of these systems have used different client IDs when writing data to EKG, information about this can be seen in the response to the POST /v1/{endpoint}/history request.

To index the unstructured information from file storages, a crawler routine must be implemented that periodically scan the entire structure of the file storages to detect new or changed files. For each file it must retrieve metadata from the contents of the file, its name and path (for example, the directory or file name may contain the name of the customer, the date, or type of provided service). A procedure can detect the keywords in the content that allow to establish the document’s relationships with the business objects, processes and events, or to implement semantic processing of the facts contained therein using NLU (Natural Language Understanding) technologies. In any case, the result of each file processing should be recorded in the Enterprise knowledge graph as a set of facts.

As a result, a complete, relevant and coherent array of information extracted from all available sources will be formed. All that's left is to build a user interface that will allow user to search in this array. In general, the user interface can consist of two main blocks. The first block implements a structured information search (faceted search), where the user selects the type (class) of the objects of interest, then sets the search criteria for specific objects, and performs a search. Basic functions of this kind are performed by the EKG Explorer, a component of our platform. Each user search query must be translated into the POST /v1/{endpoint}/entities method call, which shall reflect all user-defined search criteria. The result of the query is a table of data objects where the user can apply additional filtering, sorting, export objects to Excel, or go to view detailed information about each object.

The second block of the interface implements the search for relationships between objects. To implement this function, you actually need to create an SPARQL query constructor. One implementation option may be to sequentially select the source object and the type of the related objects to be searched, followed by semi-automatic or automatic generation of a possible relationships structure between these objects based on the ontology structure, which can be obtained by querying POST /v1/{endpoint}/model method. The result of this process will be an SPARQL query, which execution will return the desired chain of relationships between objects.

Other options for user interaction with EKG are also possible, including a graphical display of both the structure of the conceptual level of the model (possible interrelations of objects of different classes) and the relationships of specific objects.

8. Data quality

8.1. Constraints

8.2. SPARQL functions and the control ratios

8.3. Inference rules and duplicates search

8.4. Rules execution modes

8.5. Data consolidation – creating reference data set

8.6. Normalization functions

8.7. Data quality mertics

9. EKG Provider configuration

The EKG Provider directory must contain config.json file, containing the values of the following keys:

- postgres: Postgresql service database connection string as (example): "user=postgres password=1 dbname=ekg". The Postgresql server must allow the md5 authorization method by setting the appropriate setting in the pg_hba file.conf.
- internal_broker: the type of broker used for the internal subscription queue – Kafka or RabbitMQ
- broker_host: Broker's IP address
- broker_port: Broker Port
- broker_user and broker_password: login and password to connect to the broker (in the current version applicable only for RabbitMQ)

- broker_queue: the name of the queue (topic) for posting subscription tasks.

All other EKG Provider settings are specified in the configuration database and can be managed using the ekg-provider console commands [command] [parameters]. If the EKG Provider is running in a container, then the command must be executed in the container.

After executing any of the commands described below, the EKG Provider container must be restarted for the changes to take effect.

In all cases, use the _ symbol at the appropriate position in the command parameter set to omit a parameter value or set it to an empty value. If the command parameter consists of multiple words (for example, the name of an access point or storage), enclose it in quotation marks.

Please contact DataVera to obtain the full version of this guide.

9.1. Endpoints

9.2. Storages

9.3. Classes objects distribution between storages

9.4. Mapping Ontology Properties to Storage Columns

9.5. Client systems

9.6. Managing Access Rights for Client Systems

9.7. Languages

9.8. Listening queues

10. Logging and monitoring

EKG Provider writes the log file /log/ekg.log (the path relative to the root folder of the container). It is recommended to mount the log subdirectory of the container file system in a shared file storage. It is also recommended to implement the logs collection from these files to ELK, especially when deploying in Kubernetes, when the ekg-provider process runs in multiple instances, and each instance maintains its own log file.

Each log line is a JSON that has the following keys:

- Time – event time in timestamp format
- Readable_time – the time of the event in a readable format
- Code – HTTP code of the event type: 200 for normal processing of requests, 503 for errors, 403 for failure events due to lack of access rights, 404 for calls to non-existent URLs.
- Message – message text
- Extra – additional textual information (for example, an explanation of the error text)

- Value – additional numerical information (for example, pid of the process in which the error occurred)

EKG Provider has an HTTP interface for collecting metrics in Prometheus, available at /metrics of the web server. It provides the following metrics:

- average_time – gauge, average query execution time. The metric is collected in the dimensions of API URL (entity, entities, model, etc.) and method type (get, put, delete)
- rps - gauge, the average number of requests processed per second. Collected in the same dimensions
- errors_counter – counter, total number of errors encountered
- errors_rps is a gauge, the average number of errors encountered per second.

There are also /health/live and /health/ready methods intended for setting up liveness probe и readiness probe in Kubernetes. The /health/live responds with 200 OK HTTP code if the service is alive. The /health/ready method responds with 200 OK HTTP code if the server is alive and has connection to the Postgres database, and with 503 code otherwise.

The Fig. 4 shows an example of Grafana dashboard with the EKG Provider functioning metrics.

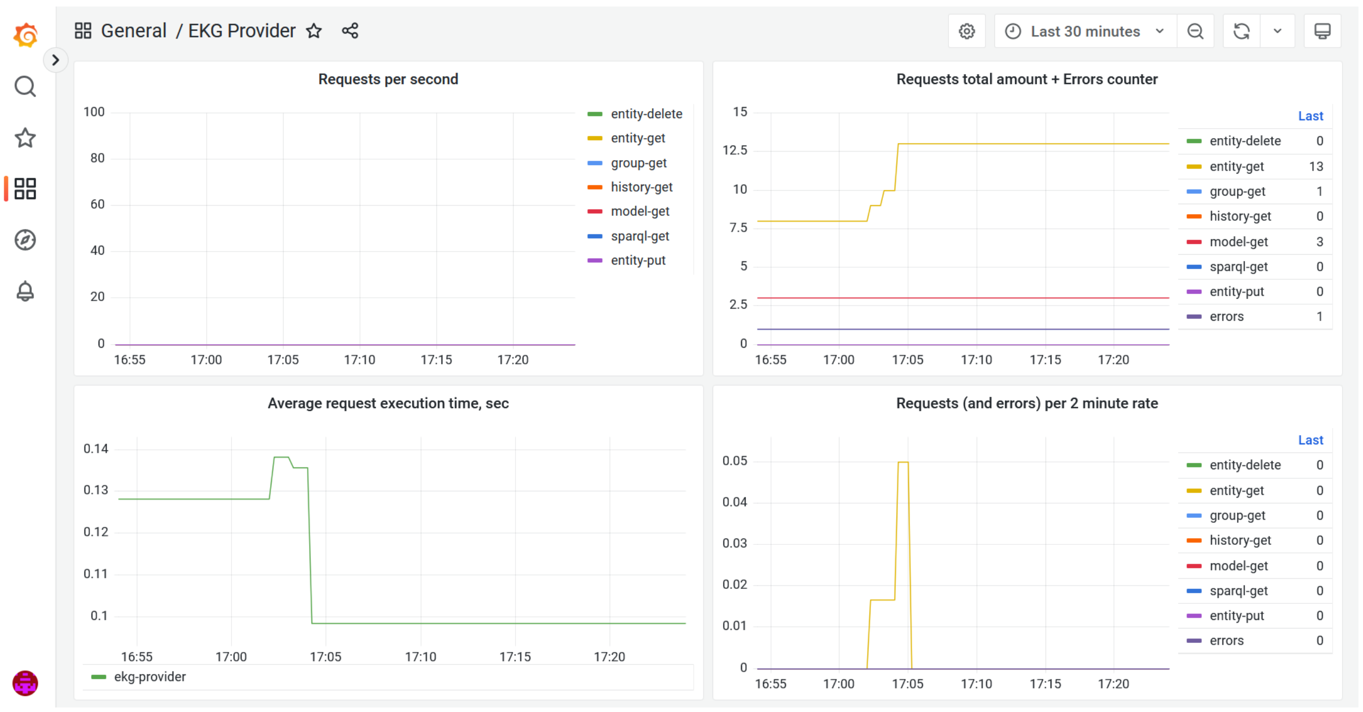


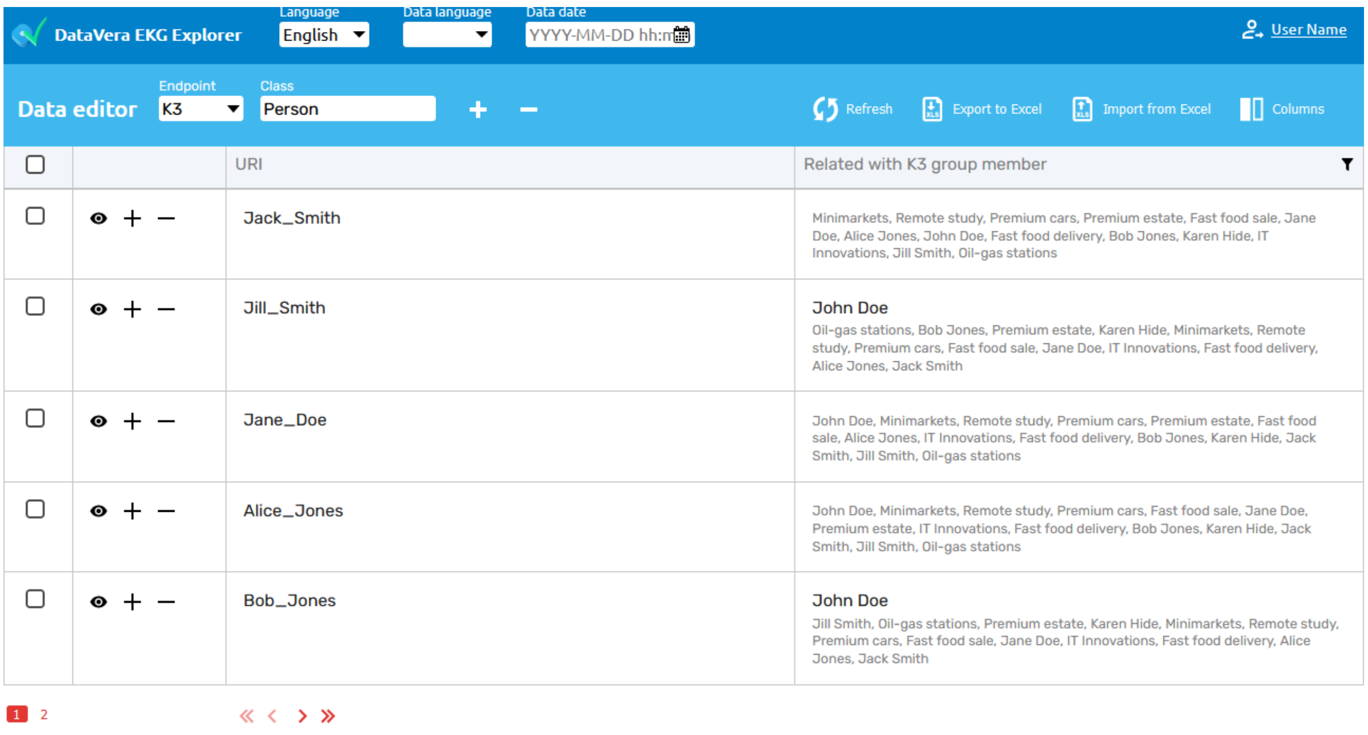
Fig. 4. An example of Grafana dashboard with the EKG Provider functioning metrics

11. Working with data using EKG Explorer

DataVera EKG Explorer is a platform component designed for the user interaction with the content of the corporate knowledge graph. This web application uses KeyCloak to authorize and authenticate users. Before starting work, you need to create a group(s) of KeyCloak users and assign each group to the EKG Provider client system. To do this, select a group in the KeyCloak administration panel, go to the Attributes tab, create an attribute with the "datavera" key and the value: { "priority": 10, "client": "system" }. After that, KeyCloak users belonging to this group will work with EKG Provider data using the EKG Explorer interface with rights corresponding to the client system whose identifier is specified in the client key.

11.1. Navigation and main operations with the data

The overall view of EKG Explorer interface is shown on the Fig. 5.



		URI	Related with K3 group member
☐			
☐	👁 + –	Jack_Smith	Minimarkets, Remote study, Premium cars, Premium estate, Fast food sale, Jane Doe, Alice Jones, John Doe, Fast food delivery, Bob Jones, Karen Hide, IT Innovations, Jill Smith, Oil-gas stations
☐	👁 + –	Jill_Smith	John Doe Oil-gas stations, Bob Jones, Premium estate, Karen Hide, Minimarkets, Remote study, Premium cars, Fast food sale, Jane Doe, IT Innovations, Fast food delivery, Alice Jones, Jack Smith
☐	👁 + –	Jane_Doe	John Doe, Minimarkets, Remote study, Premium cars, Premium estate, Fast food sale, Alice Jones, IT Innovations, Fast food delivery, Bob Jones, Karen Hide, Jack Smith, Jill Smith, Oil-gas stations
☐	👁 + –	Alice_Jones	John Doe, Minimarkets, Remote study, Premium cars, Fast food sale, Jane Doe, Premium estate, IT Innovations, Fast food delivery, Bob Jones, Karen Hide, Jack Smith, Jill Smith, Oil-gas stations
☐	👁 + –	Bob_Jones	John Doe Jill Smith, Oil-gas stations, Premium estate, Karen Hide, Minimarkets, Remote study, Premium cars, Fast food sale, Jane Doe, IT Innovations, Fast food delivery, Alice Jones, Jack Smith

Рис. 5. The overall view of DataVera EKG Explorer interface

Work in the interface begins with choosing of an endpoint and a class. The endpoint is selected from the list of endpoints available to the current user in accordance with his/her rights, using the drop-down "Endpoint" menu. The class is selected in the pop-up dialog box shown in Fig. 6:

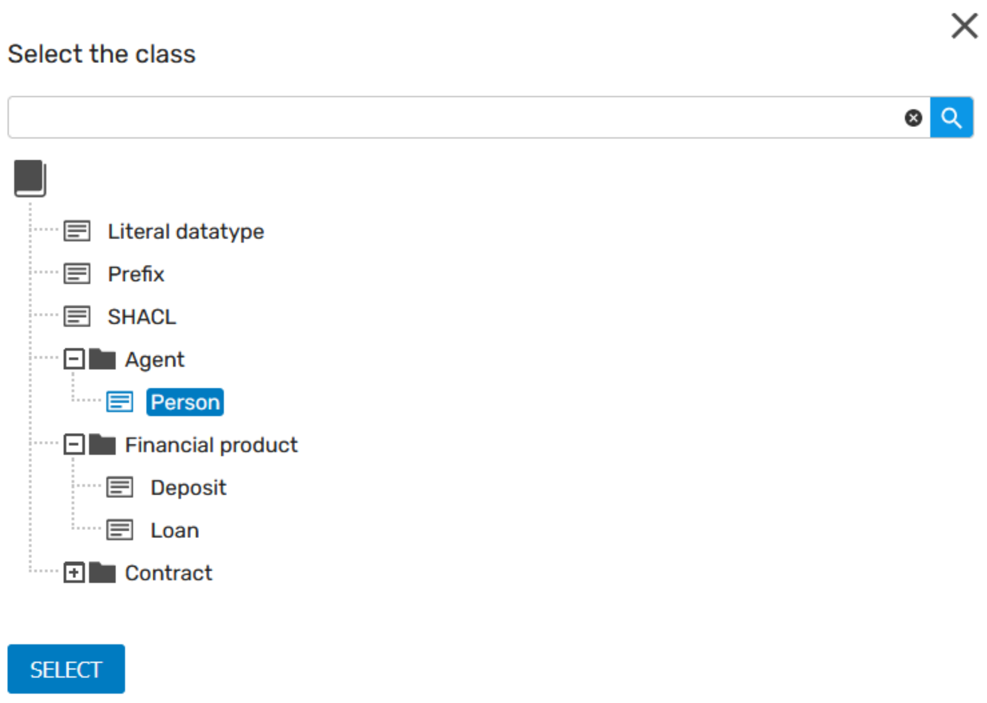


Fig. 6. Class selection dialog

After selecting a class from the list, the main part of the page displays a list of the individual objects of that class. Each line of this list corresponds to one data object.

To the right of the class selector above the list are the "+" and "-" buttons. Clicking on the "+" button adds a new line to the list, which allows you to create a new data object. Clicking on "-" allows you to delete objects marked with a switch on the left side of the line.

There are also three buttons on the left side of each row. The button with the "eye" icon opens the window for viewing the properties of the object, which is described below. The "+" button allows you to add a line to create a new data object after this line. The "-" button is intended for deleting the data object corresponding to this line. Further in the table there are columns, each of which corresponds to some property of the class objects.

On the right side of the page, above the list, there is a menu consisting of several buttons:

- Refresh
- Export to Excel
- Import from Excel
- Choose columns

Clicking on the "Update" button allows you to re-request data from the EKG Provider – for example, if they have changed as a result of importing data there via the API. The remaining buttons are discussed below.

The topmost line of the EKG Explorer interface displays controls for selecting the interface language, data language, and the time for which the state of data is viewed:

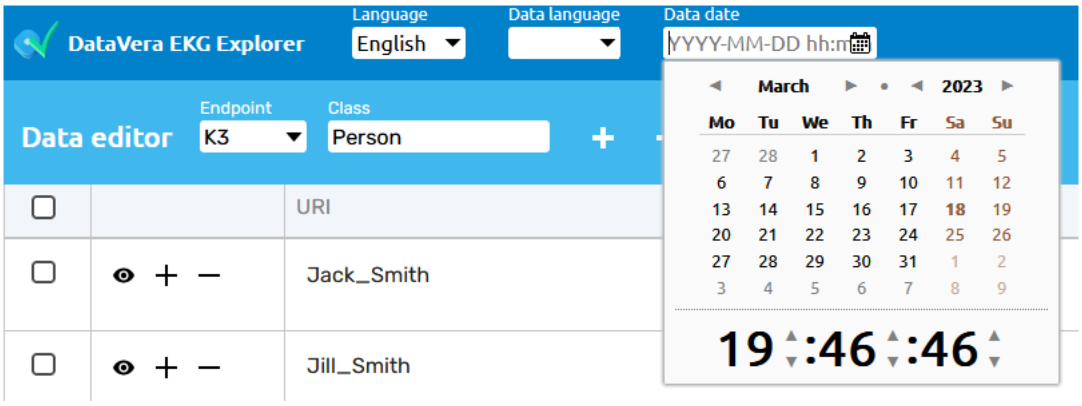


Fig. 7. The controls at the top of the page

The choice of the interface language only affects the language in which the labels of the interface elements are displayed.

The choice of data language affects the display and editing of data. Values of string properties of objects and names of referenced objects in the selected language will be displayed, as well as values for which no language is specified. When editing, all string values written to storage will receive a language label corresponding to the selected data language.

When you select a date for viewing and editing data, the table will display not the current state of the data objects, but the one that was relevant on the selected date (the system will respond slower due to the historical information extraction). When saving data in set date mode, the current state of the objects in the database is changed, but a change history is recorded for the date that the user has chosen.

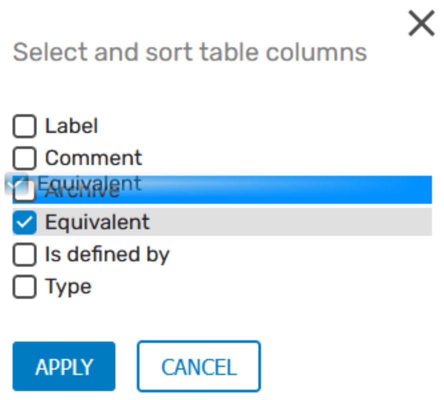
11.2. Objects list operations

Clicking on the heading of any table column allows you to sort objects by the value of the property contained in this column. Once the sort is set, a triangle will appear on the right side of the column heading to indicate the sorting direction. Clicking the heading of the same column again sorts the list in reverse order.

There is a filter icon in the header of each column. When you click on it, a field for entering a value is displayed. If you enter a combination of the characters to search and press the Enter button, the list will be filtered so that only the objects having the entered value in their properties will remain.

The column selection button on the right side of the page above the list allows you to open a dialog box in which you can mark the columns for which properties you want to display in the list (see Fig. 8). By dragging the elements in this dialog box, you can change the order of the columns.

The list displays all properties applicable to objects of the selected class according to the model (TBox).



Select and sort table columns

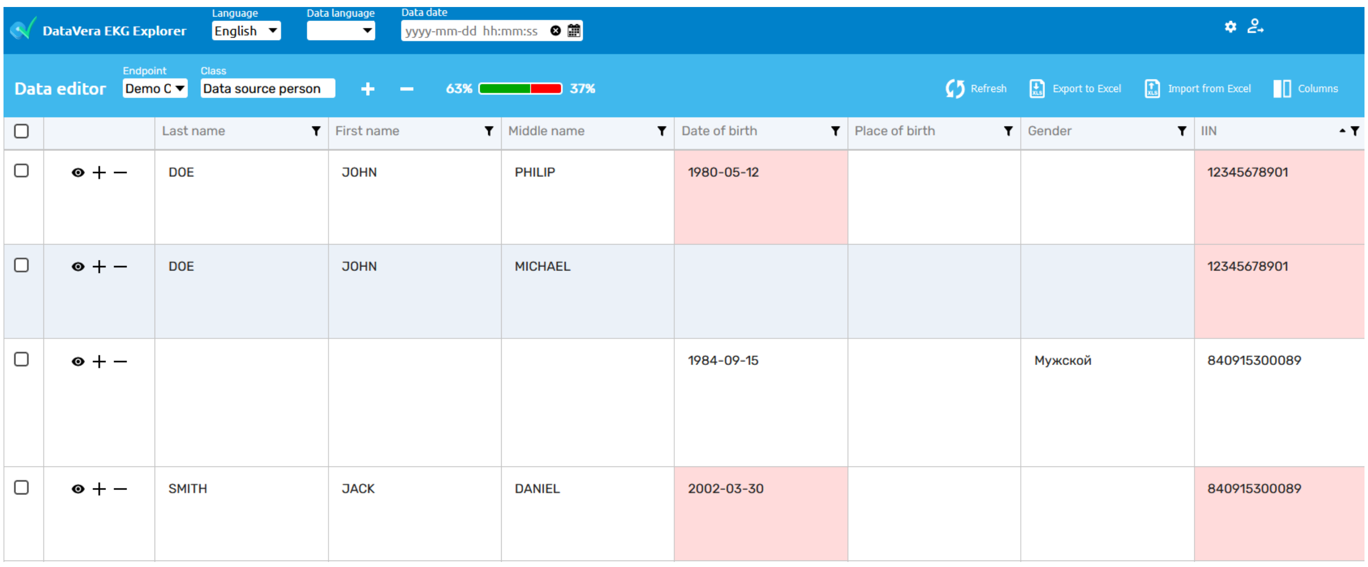
- ☐ Label
- ☐ Comment
- ☐ Equivalent
- ☐ Archive
- ☑ Equivalent
- ☐ Is defined by
- ☐ Type

APPLY CANCEL

Fig. 8. The columns selection window

11.3. Viewing data quality

If the data quality control rules (constraints) are set up for a particular class, you will see the data quality indicator above the objects list:



☐		Last name	First name	Middle name	Date of birth	Place of birth	Gender	IIN
☐	● + –	DOE	JOHN	PHILIP	1980-05-12			12345678901
☐	● + –	DOE	JOHN	MICHAEL				12345678901
☐	● + –				1984-09-15		Мужской	840915300089
☐	● + –	SMITH	JACK	DANIEL	2002-03-30			840915300089

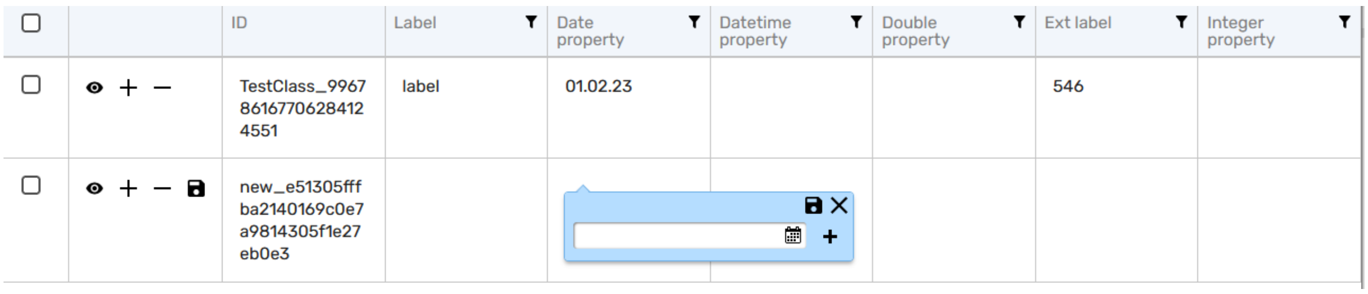
Fig. 9. Data quality indicator

The green part of the indicator reflects amount of the objects not violating the constraints, the red part reflects the amount of objects having at least one violation. Clicking on the indicator you can filter the objects by showing only those having the data quality problems (this will be combined with the other filters).

The cells of the list containing object property values violating some constraints will be highlighted in red. You can see violation description by hovering the cell. The constraints violations are also displayed in the object properties dialog.

11.4. Data records editing

You can add a new line to the list by clicking on the "+" button above the list or in any of its existing lines. A new line will appear displaying the identifier for the generated object:



☐		ID	Label	Date property	Datetime property	Double property	Ext label	Integer property
☐	+ −	TestClass_99678616770628412 4551	label	01.02.23			546	
☐	+ −	new_e51305fffba2140169c0e7a9814305f1e27eb0e3						

Fig. 10. Adding a new object to the list

If there are mandatory fields among the object properties, i.e. having minCardinality = 1 or more, pop-up elements for entering or selecting the values of these properties will immediately appear. You can invoke such a window for any other property by double-clicking on a table cell.

The pop-up window for entering values may look different depending on the type of property values. For example, for fields of type 'date', it contains a 'calendar' body for selecting a date. If the property is a reference to objects of another class, a picker will be displayed in the popup window to select the object, and so on. In any case, it is important to note that the pop-up window allows you to enter or select several values at once for any property of each object, which is a feature of the ontological data model. If a property can only have one value, this must be explicitly specified in the data model by setting that property to maxCardinality = 1.

Clicking on the "+" button to the right of the property value allows you to add a new value, as shown in Fig. 10. Pressing the "-" button removes the value from the list.

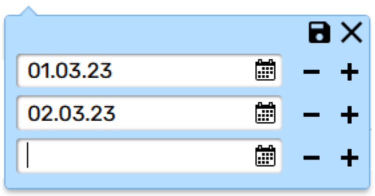


Fig. 11. Property values editing box

After editing the property values, you must click on the save button in the pop-up window or in the line as a whole.

Some string property values can be set in a specific language. The language label is displayed to the left of the value:

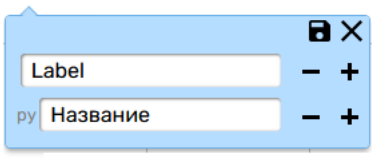
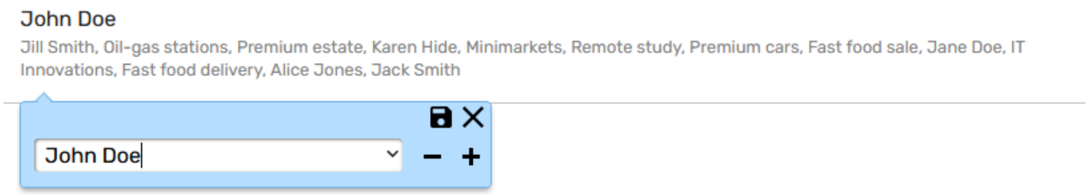


Fig. 12. Language tag for a string value

To set a string property value in a specific language, you must select the data language at the top of the EKG Explorer screen. You cannot change the language of already created values.

Object properties can have not only values assigned explicitly, but also receive values as a result of the inference. Such values are displayed in the list of objects in gray and are not available for editing:


John Doe
Jill Smith, Oil-gas stations, Premium estate, Karen Hide, Minimarkets, Remote study, Premium cars, Fast food sale, Jane Doe, IT Innovations, Fast food delivery, Alice Jones, Jack Smith

John Doe

Fig. 13. Inferenced values display

ПInferenced values are updated automatically when saving data objects.

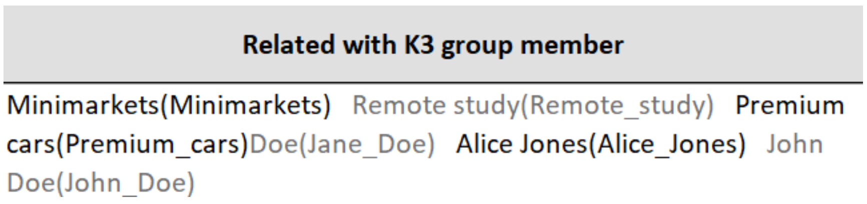
11.5. Excel import and export

The "Export to Excel" button downloads and opens an Excel file containing objects that the user see in the application. The list in Excel has the same format and structure as in the application, i.e. it is subject to column selection, sorting, and filtering settings.

You can edit object property values in the Excel file and load them back into EKG Explorer. To do this, click the "Import from Excel" button. A dialog box will appear in which you need to select a file with modified objects, and click the "Import" button (you cannot change the order and the set of the file columns! You can only edit the values of object properties in it). The "update/replace" switch determines how changes are applied to objects in the database: when you select "update", the property values of objects in the system will be updated with the values contained in the file, while existing property values other than those specified in the file will be saved. The "replace" option allows you to completely replace the objects properties with those contained in the file. In any case, the import will not affect the values of those properties for which there are no columns in the file.

For properties that have a reference type, when exporting, the value in the cell is substituted with the object identifier, followed by the name of the object in brackets. When importing, only the identifier matters, i.e. the name of the object in brackets can be omitted during import.

If a property has multiple values, different values are highlighted in color when exported to Excel, as shown in Figure 14:



Related with K3 group member
Minimarkets(Minimarkets) Remote study(Remote_study) Premium cars(Premium_cars)Doe(Jane_Doe) Alice Jones(Alice_Jones) John Doe(John_Doe)

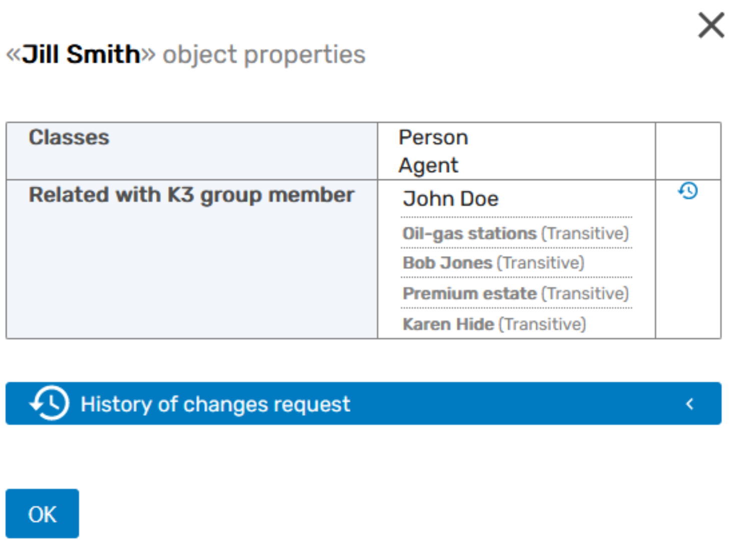
Fig. 14. The result of Excel exporting an object with several values of one property

The specific color in which the property values are specified is not important. When preparing a file for import, you can color-code property values in a way that is convenient for the user.

Inferred property values are not exported to Excel.

11.6. Viewing object properties

The 👁 button in each object row opens the dialog for viewing its properties.



«Jill Smith» object properties

Classes	Person Agent	
Related with K3 group member	John Doe Oil-gas stations (Transitive) Bob Jones (Transitive) Premium estate (Transitive) Karen Hide (Transitive)	

Fig. 15. Object properties window

This window displays the list of classes to which the object belongs, as well as the values of all its properties. For each property, a "history" icon is displayed. Clicking on it opens a dialog box for viewing the history of the property value. Clicking on the "request change history" button at the bottom of the window allows you to display the complete history of this data object.

The inferred object properties are displayed in this window in gray. For each such property value, the name of the SHACL rule is displayed.

When viewing an object card, constraints violations produced by the SHACL Rules are also displayed.