Kolmogorov.ai | Morphism | Продуктовая карта

Feature Store

ООО Дата Сапиенс

version 0.20.4 | 12.04.2024

Table of contents

1. Intro

В разделе представлены продуктовые карты и схемы текущей и целевой архитектуры. Продуктовая карта и схемы "1.0" являются целевой архитектурой приложения.

2. Product Map

2.1 Product Map

Section	Code	Description
USER	D01C	Create Dataset *deprecated*
USER	D02R	Get All Datasets
USER	D03D	Delete Datasets
USER	D04R	Get One Dataset
USER	D05U	Save Dataset
USER	D06U	Update Dataset *deprecated*
USER	D07R	Create Dataset CSV
USER	D08C	Validate Dataset *deprecated*
USER	D09R	Get Dataset History *deprecated*
USER	D10R	Get Dataset Statistics
USER	D11R	Get Dataset Statistics Mapping *deprecated*
USER	D12C	Split Dataset *deprecated*
USER	D13C	Impute Dataset *deprecated*
USER	D14R	Get Dataset Status
USER	D15R	Preview Dataset
USER	D16C	Validate Expression
USER	D17R	Get All Dataset Light
USER	D18R	Get All Datasets Paged
USER	E01C	Create Entity
USER	E02R	Get All Entities
USER	E03D	Delete Entities
USER	E04U	Update Entity
USER	E05R	Get One Entity
USER	E08R	Get All Key
USER	E10U	Update Entity Key
USER	E11R	Get One Key
USER	E12R	Get All Entities Paged
USER	E13R	Get All Entity Link
USER	E14U	Update Entity Link
USER	E15R	Get One Entity Link
USER	E18D	Delete Entity Link
USER	E19R	Get All Link Chains
USER	E20C	Add Link Chains Into Ref
USER	E21R	Get All Link Chains From Ref

Section	Code	Description
USER	E22U	Update Link Chain In Ref
USER	E23D	Delete Link Chains From Ref
USER	E24R	Get All Entity Link Paged
USER	E25R	Get All Link Chains Paged
USER	F01C	Create Feature
USER	F02R	Get All Features
USER	F03D	Delete Features
USER	F04U	Update Feature
USER	F05R	Get One Feature
USER	F06R	Get All Feature Column Information
USER	F07U	Update Feature Column
USER	F08R	Get All Feature Light
USER	F09R	Get All Feature Paged
USER	L01C	Create Loader
USER	L02R	Get All Loaders
USER	L03U	Activate Loader *deprecated*
USER	L04U	Deactivate Loader *deprecated*
USER	L05R	Get One Loader
USER	L06U	Update Loader
USER	L07X	Execute Loader
USER	L08D	Delete Loaders
USER	L09U	Recreate Loader
USER	L10R	Get Loader Status
USER	L11C	Create Loader from Loader
USER	L12R	Get All Loaders Paged
ADMIN	G03D	Delete Tags
ADMIN	G02R	Get All Tags
ADMIN	S01C	Create Datasource
ADMIN	S02R	Get All Datasources
ADMIN	S03R	Get All Source Tables
ADMIN	S04R	Get Table Columns
ADMIN	S05U	Update Datasource
ADMIN	T01R	Get All Data Types
ADMIN	T02R	Get One Data Type

Section	Code	Description
ADMIN	T03R	Get All Data Type Stats
ADMIN	T04R	Get All Data Category
ADMIN	Y01C	Upload File
ADMIN	Y02R	Download File
ADMIN	Y03D	Delete File
ADMIN	Y04R	Get List of Files
ADMIN	Y05R	Get Text File Content
ADMIN	Y06C	Create Text File
ADMIN	Z01R	Export Catalogue
ADMIN	Z02C	Import Catalogue
ADMIN	Z03C	Migrate Catalogue Config
ADMIN	Z04R	Get Import History
ADMIN	Z05C	Stop Current Import

2.2 Catalogue

Раздел описывает методы работы с импортом/экспортом каталога приложения.

Структура конфига импорта/экспорта описана в конце страницы.

2.2.1 Export Catalogue (GET /catalogue/export)

Уровень доступа	**admin**
Request Body	**-**
Response Body	**File**

Экспортирует в json-конфиг с объектами каталога:

- Справочники типов данных
- Сущности и их ключи
- Переменные
- Автосвязи
- External-загрузчики (только загрузчики переменных и связей, без загрузчиков сегментов)
- Загрузчики из файлов (только загрузчики переменных и связей, без загрузчиков сегментов)
- Задания расчета (только по экспортируемым загрузчикам из файлов)
- Справочник цепочек (только по экспортируемым связям)
- Права доступа на загрузчики

В качестве ссылок на связанные объекты вместо суррогатных ключей (`rk`) в экспортируемом конфиге указываются их названия - `feature.name`, `entity.name`, etc. Для загрузчиков значение схемы в конфиге подменяется на параметр, если схема определена для источника -1 или в переменной окружения `DB__DATASET_SCHEMA`:

- `v100_source.work_schema` -> `"{work}"`
- `v100_source.store_schema` -> `"{store}"`
- `DB__DATASET_SCHEMA` -> `"{dataset}"`

NB! По файловым загрузчикам экспортируются только метаданные, загруженные данные не экспортируются

2.2.2 Import Catalogue (POST /catalogue/import)

Уровень доступа	**admin**
Request Body	**File**
Response Body	**%import_task**

Импортирует объекты каталога из json-конфига через запуск асинхронной задачи импорта. При этом в один момент времени может быть запущена только одна задача импорта. При попытке выполнить параллельный запуск еще одной задачи импорта, вернется ошибка.

Возможные параметры

"Признак импорта прав доступа" - `import_permissions`

Если параметр = `True`, то права доступа импортируются из конфига, в противном случае по созданным загрузчикам будут добавлены базовые права доступа = `'full'`.

"**Проверка источника**" - `check_source`.

Параметр отвечает за проверку таблиц-источников загрузчиков. Возможные значения:

- `disabled` - проверка не осуществляется
- `tables` - проверяется наличие таблиц-источников загрузчиков в источнике
- `columns` - проверяется наличие таблиц-источников загрузчиков в источнике и наличие в них полей, указанных в маппинге загрузчика

Проверка источника производится только для external-загрузчиков. Для файловых загрузиков проверки наличия файлов в s3 и их структуры **не производятся**.

"**Режим импорта**" - `mode`

Возможные значения:

- `replace` - создаются новые объекты, обновляются существующие объекты, удаляются все остальные. При этом из датасетов удаляются только объекты, которые в терминах v2 являются загрузчиками или связями.
- `upsert` - создаются новые объекты и обновляются существующие (без очистки)
- `append` - только создаются новые объекты

NB! В режиме `upsert` невозможно изменить тип данных существующего ключа, а также состав ключей существующей сущности

Блокировка инфокарты (!)

Ввиду того, что импорт выполняется в одну транзакцию (в целях предотвращения неконсистентности метаданных), на время его выполнения могут блокироваться некоторые операции с инфокартой (ИК) в зависимости от режима импорта. В приложении такие блокировки выглядят как "зависание" и длятся до завершения работы задачи импорта.

Подробнее про особенности блокировок для каждого режима импорта см. ниже:

1. В режиме `append` ничего не блокируется, доступны все операции
2. В режиме `upsert` не гаранитруется корректное выполнение операций изменения/удаления объектов, которые есть в импортируемом конфиге, т.к. эти объекты пересоздаются. Остальные операции работают стабильно.
3. В режиме `replace` стабильно работает только создание загрузчиков сегментов и датасетов

NB! Для операций по изменению ИК, запущенных во время работы импорта в режимах `upsert` и `replace`, не гарантируется корректное выполнение. После завершения работы импорта такие операции могут как примениться к результату импорта, так и упасть с ошибкой. Рекомендуется не проводить такие операции или прерывать их.

Алгоритм импорта

Данные конфига импортируются в следующем порядке:

1. Справочники типов данных
2. Переменные
3. Ключи сущностей
4. Сущности
5. Датасеты (external-загрузчики + загрузчики файлов + автосвязи)
6. Задания расчета (загрузчиков)
7. Права доступа (опционально)
8. Справочник цепочек

ПЕРЕМЕННЫЕ

Для переменных, которые присутствуют в метаданных на момент импорта (совпадает `name`), сохраняются `feature_rk`.

КЛЮЧИ СУЩНОСТЕЙ

Для ключей, которые присутствуют в метаданных на момент импорта (совпадает `name` + `fs_type`), сохраняются `key_rk`.

СУЩНОСТИ

Для сущностей, которые присутствуют в метаданных на момент импорта (совпадает `name` + `keys`), сохраняются `entity_rk`.

ДАТАСЕТЫ

Для датасетов, их переменных и инстансов, которые присутствуют в метаданных на момент импорта (совпадает `name`), сохраняются `dataset_rk`, `feature_rk` и `instance_rk`.

Также для пересоздаваемых датасетов-загрузчиков-из-файлов удаляется таблица с данными, за исключением пересоздаваемых датасетов, по которым сохраняется `dataset_rk` и не меняется структура таблицы (`db_schema` + `db_table` + все `db_column`)

Если в конфиге присутствуют загрузчики из файла, то переменная `S3__ENABLED` должна быть = `True`, иначе вернется ошибка.

ЗАДАНИЯ РАСЧЕТА

Импортируются задания расчета из конфига вместе с привязкой к датасетам. Значения `job_rk` не сохраняются.

Задания расчета импортируются только по импоритруемым датасетам (если датасет создается или пересоздается). Если датасет из конфига не импортируется (например, существующий датасет в режиме `append`), то существующее задание расчета в этом случае остается без изменений.

ПРАВА ДОСТУПА

Импортируются права доступа из конфига, если установлен `import_permissions = True`. Сохранение значений `permission_rk` не гарантируется.

2.2.3 Migrate Catalogue Config (POST /catalogue/migrate_config)

Уровень доступа	**admin**
Request Body	**File**
Response Body	**File**

Приводит переданный конфиг к последней версии, возвращает файл с json-конфигом с актуальной схемой.

2.2.4 Get Import History (GET /catalogue/import/hist)

Уровень доступа	**admin**
Request Body	-
Response Body	%import_task

Возвращает список задач импорта с учетом параметров фильтрации:

- `task_id`: `list[uuid]` - список идентификаторов задач
- `status`: `list[enum]` - список статусов
- `last_flg`: `bool` - признак последней задачи импорта

Параметры `limit` и `offset` отвечают за размер и сдвиг выборки соответственно. При значении `limit: null` возвращается дефолтное количество объектов, которое определяется параметром `APP__DEFAULT_ITEMS_LIMIT`. Максимальное значение `limit` определяется параметром `APP__MAX_ITEMS_LIMIT`.

2.2.5 Stop Current Import (POST /catalogue/import/stop)

Уровень доступа	**admin**
Request Body	-
Response Body	%import_task

Выполняет попытку остановки текущей задачи импорта. Если задачу удалось остановить, то по ней проставляются следующие значения:

- `FINISH_DTTM` = дата-время остановки задачи
- `STATUS` = `failed`
- `DETAIL` = значение параметра `reason: str`

Возвращает ошибку при отсутствии текущей задачи импорта или если задачу не удалось остановить.

2.2.6 Структура конфига импорта/экспорта

Импортируется и экспортируется всегда только актуальная версия конфига, выбрать версию конфига для импорта/экспорта нельзя.

Для приведения конфига к актуальной версии необходимо воспользоваться эндпоинтом `POST /catalogue/migrate_config`.

- Актуальная версия - %catalogue_v4
- Архивные версии:
- %catalogue_v1
- %catalogue_v2
- %catalogue_v3 - структура совпадает с `v2`, отличается только набор валидаций в части прав доступа. В `v3` могут быть только `no_access`, `read` и `full`, в `v2` к этому списку добавляется `view` и `edit`.

2.3 Datasource

Источники данных переехали в архитектуру v100 Sources→

2.4 File

Работа с файлами теперь происходит только в архитектуре v100 Files→

2.5 Tag

2.5.1 Получение списка тегов (GET /tag)

Уровень доступа	**read**
Request Body	**-**
Response Body	**list[str]**

Возвращается полный список существующих тегов.

Поддерживает фильтрацию по маске тега (параметр запроса `mask`), в т.ч. исключающую фильтрацию при добавлении символа `^` в начало значения маски фильтрации.

2.5.2 Удаление тегов (DELETE /tag)

Уровень доступа	**load**
Request Body	**list[str]**
Response Body	**-**

Удаляет тег из справочника тегов. Со связанных объектов тег не удаляется.

2.6 Dataset

2.6.1 Create

Провалидировать формулу (POST /dataset/validate_expression)

Уровень доступа	**read**
Request Body	-

Валидирует формулу из версий фичей.

В выражении допустимо использовать:

- Суррогатный ключ версии переменной из каталога (`column_rk`) - `"{1}"`, `"{2}"`, `"{3}"`.
- Ключ сущности - `"<1>"`
- Алиасы других выражений из запроса, например: `"coalesce(dataset_feature_1, dataset_feature_2 + dataset_feature_3)"`

Валидация выражения включает в себя **только** проверку совместимости типов данных.

Перед версией переменной или перед ключом сущности допустимо указать цепочку (например, `"[1,2]{1}"` или `"[3,4]<5>"`). В этом случае цепочка игнорируется и не проверяется.

Создание датасета через конструктор (POST /dataset)

Deprecated

Стратификация (POST /dataset/{dataset_rk}/split)

Deprecated

Импутация (POST /dataset/impute)

Deprecated

Провалидировать конфигурацию (POST /dataset/validate)

Deprecated

2.6.2 Delete

Удалить датасеты (DELETE /dataset)

Уровень доступа	**transform**
Request Body	**-**

В фоновом режиме удаляются датасеты из списка переданных `dataset_rk` как в метаданных, так и физические таблицы (при наличии).

NB! Для датасетов-сегментов, загруженных external-загрузчиком, таблица датасета в БД удаляется только при передаче параматра force_flg = true. Если force_flg = false (значение по умолчанию), то таблица датасета не удаляется.

2.6.3 Edit

Изменить датасет (PATCH /dataset/{dataset_rk})

Уровень доступа	**transform**
Request Body	**-**

Info:

```
Поле запроса "SCHEDULE" НЕ ИСПОЛЬЗУЕТСЯ!
```

Изменяет имя и описание датасета При редактировании накладываются все те же ограничения и проверки что и при создании датасета.

Изменить конфигурацию датасета (PUT /dataset/{dataset_rk})

Deprecated

2.6.4 Info

Получить список датасетов (GET /dataset)

Уровень доступа	**read**
Request Body	**-**

Возвращает полный список датасетов с расширенной информацией по ним.

Для каждого датасета выводится информация о статусе датасета (STATUS), статусе его последнего запуска (LAST_RUN_STATUS) и статусе готовности CSV с данными датасета. Также для сегментов (датасеты, загруженные с помощью загрузчиков) выводится информация о статусе соответствующего загрузчика (LOADER_STATUS).

Получить пагинированный список датасетов (GET /dataset/page)

Уровень доступа	**read**
Request Body	**-**

Метод является копией оригинального `GET /dataset` с добавлением аргументов пагинации `limit` и `offset` для регулирования выборки.

Параметры `limit` и `offset` отвечают за размер и сдвиг выборки соответственно. При значении `limit: null` возвращается дефолтное количество объектов, которое определяется параметром `APP__DEFAULT_ITEMS_LIMIT`. Максимальное значение `limit` определяется параметром `APP__MAX_ITEMS_LIMIT`.

Получить список датасетов 2.0 (GET /dataset/light)

Уровень доступа	**read**
Request Body	**-**

Возвращает полный список датасетов с ограниченным набором атрибутов. Метод поддерживает фильтрацию результирующего списка по значениям параметров, указанных ниже.

Параметры фильтрации:

- name: mask - название датасета
- description: mask - описание датасета
- entity_rk: list[int] - суррогатный ключ сущности(-ей)
- segment_flg: bool - флаг сегмента (True, если датасет получен через загрузчик)
- empty_flg: bool - флаг пустого датасета (True, если датасет пустой)
- status: str - статус датасета
- job_rk: int - суррогатный ключ задания

Реализована возможность получать список датасетов порциями с возможностью регулирования размера выборки.

DATASET STATUS, LOADER STATUS, LAST RUN STATUS

Статус	Описание
new	Объект зарегистрирован, но задание расчета не запускалось
running	Задание расчета выполняется
success	Задание расчета успешно завершено
failed	Задание расчета завершилось с ошибкой

CSV STATUS

Статус	Описание
no_file	CSV-файл для датасета не создавался
running	Форимрование CSV-файла в процессе
old	Текущее наполнение CSV-файла устарело
actual	Текущее наполнение CSV-файла соответствует наполнению датасета

Метод поддерживает фильтрацию возвращаемого списка по значениям параметров, указанных ниже, где для параметров имени и описания доступен исключающий поиск с добавлением символа `^` в начало значения маски фильтрации.

Реализована возможность получать список датасетов порциями с возможностью регулирования размера выборки.

Получить полную информацию о датасете (GET /dataset/{dataset_rk})

Уровень доступа	**read**
Request Body	-

Возвращает подробную информацию по датасету, включая полную конфигурацию расчета со списком всех используемых переменных и связей зерен (если датасет построен через конструктор)

Экспорт данных в CSV (POST /dataset/dataset_csv/{dataset_rk})

Уровень доступа	**read**
Request Body	-

Запускает создание CSV-файла с данными датасета (разделитель - запятая, заголовки включены), возвращает путь к CSV-файлу на S3. Доступно только для непустых датасетов.

Превью датасета (GET /dataset/{dataset_rk}/preview)

Уровень доступа	**read**
Request Body	-

Возвращает запрошенное количество строк из таблицы датасета по всем полям кроме технических.

Получить статус датасета (GET /dataset/{dataset_rk}/status)

Уровень доступа	**read**
Request Body	**-**

Возвращает актуальный статус датасета.

Параметр	Описание
loader_rk: int	Суррогатный ключ датасета

Получить список расчетных дат (GET /dataset/history/{dataset_rk})

Deprecated

2.6.5 Statistics Info

Получить значения метрик (GET /dataset/stats/{dataset_rk})

Уровень доступа	**read**
Request Body	-

Возвращает значения статистических показателей по датасету, рассчитанных после завершения его загрузки.

С точки зрения глубины используемых данных, статистики делятся на две категории:

- За всю историю
- В разрезе отчетной даты датасета

> **Info**
>
> Статистики в разрезе отчетной даты не рассчитываются, если: - В датасете одна отчетная дата (т.к. являются дублями статистик за всю историю) - Отчетная дата датасета берется из сегмента

> **Info**
>
> Для датасетов, полученных из источника данных через загрузчик (сегмент) считается только статистика ROWS_COUNT.

Более подробная классификация статистик приведена ниже.

ОБЩИЕ СТАТИСТИКИ ПО ТАБЛИЦЕ ДАТАСЕТА ЗА ВСЮ ИСТОРИЮ

Показатель	Описание
ROWS_COUNT	Количество строк в датасете
ENTITY_COUNT	Количество значений сущности в датасете
REPORT_DTTM_COUNT	Количество отчетных дат (срезов) в датасете

ОБЩИЕ СТАТИСТИКИ ПО ТАБЛИЦЕ ДАТАСЕТА ЗА ОТЧЕТНЫЕ ДАТЫ

Показатель	Описание
ROWS_COUNT_PER_DTTM	Количество строк в датасете в разрезе отчетной даты

ОБЩИЕ СТАТИСТИКИ ПО ВСЕМ ПЕРЕМЕННЫМ ДАТАСЕТА ЗА ВСЮ ИСТОРИЮ

Показатель	Описание
NULL_COUNT	Количество null-ов
NULL_SHARE	Доля null-ов от общего количества строк в датасете
COUNT_DISTINCT	Количество уникальных значений

> **Info**
>
> Статистики NULL_COUNT, NULL_SHARE, COUNT_DISTINCT считаются для каждой переменной датасета, если количество строк в нем больше нуля.

ОБЩИЕ СТАТИСТИКИ ПО ВСЕМ ПЕРЕМЕННЫМ ДАТАСЕТА ЗА ОТЧЕТНЫЕ ДАТЫ

Показатель	Описание
NULL_COUNT_PER_DTTM	Количество null-ов
NULL_SHARE_PER_DTTM	Доля null-ов в разрезе отчетной даты
COUNT_DISTINCT_PER_DTTM	Количество уникальных значений в разрезе отчетной даты

КАСТОМНЫЕ СТАТИСТИКИ ПО ПЕРЕМЕННЫМ ДАТАСЕТА

Настраиваются пользователем при регистрации датасета

Показатель	Описание
MAXIMUM	Максимальное значение
MINIMUM	Минимальное значение
MAXIMUM_NO_OUTLIER	Максимум за исключением 5% записей с наибольшими значениями
MINIMUM_NO_OUTLIER	Минимум за исключением 5% записей с наименьшими значениями
AVG	Среднее значение
MEDIAN	Медианное значение
MODE	Мода
STDDEV	Стандартное отклонение
NEG_COUNT	Количество отрицательных значений
NEG_SHARE	Доля отрицательных значений
MODE_EQUAL	Количество значений, равных моде
MODE_EQUAL_SHARE	Доля всех значений, заполненная модой
AVG_NO_OUTLIER	Среднее, за исключением 5% записей с наибольшими/наименьшим значениями

КАСТОМНЫЕ СТАТИСТИКИ ПО ПЕРЕМЕННЫМ ДАТАСЕТА ЗА ОТЧЕТНЫЕ ДАТЫ

Настраивается пользователем, не рассчитывается для дат из сегмента

Показатель	Описание
MAXIMUM_PER_DTTM	Максимальное значение
MINIMUM_PER_DTTM	Минимальное значение
MAX_NO_OUTLIER_PER_DTTM	Максимум за исключением 5% записей с наибольшими значениями
MIN_NO_OUTLIER_PER_DTTM	Минимум за исключением 5% записей с наименьшими значениями
AVG_PER_DTTM	Среднее значение
MEDIAN_PER_DTTM	Медианное значение
MODE_PER_DTTM	Мода
STDDEV_PER_DTTM	Стандартное отклонение
NEG_COUNT_PER_DTTM	Количество отрицательных значений
NEG_SHARE_PER_DTTM	Доля отрицательных значений
MODE_EQUAL_PER_DTTM	Количество значений, равных моде
MODE_EQUAL_SHARE_PER_DTTM	Доля всех значений, заполненная модой
AVG_NO_OUTLIER_PER_DTTM	Среднее, за исключением 5% записей с наибольшими/наименьшим значениями

Метод поддерживает фильтрацию результирующего списка по значениям параметров, указанных ниже.

Получить список установленных метрик (GET /dataset/stats/mapping/{dataset_rk})

Deprecated

2.7 Data Type

2.7.1 Info

Получить список типов данных (GET /data_type)

Уровень доступа	**read**
Request Body	**-**

Возвращает справочник зарегистрированных типов данных. Метод поддерживает фильтрацию результирующей выборки по параметрам, указанным ниже.

Получить полную информацию о типе данных (GET /data_type/{data_type_rk})

Уровень доступа	**read**
Request Body	**-**

Возвращает детальную информацию по одному типу данных.

Получить справочник типов данных статистик (GET /data_type_stats)

Уровень доступа	**read**
Request Body	**-**

Возвращает справочник соответствия статистических показателей типам данных. Т.е. какие статистические показатели доступны для расчета для типа данных.

Ограничение:

```
Используются только записи из справочника, по которым атрибут STATS_TYPE = "DATASET". Подразумевалось,
что показатели можно будет считать и по датасетам и по переменным из каталога. В реальности их можно
считать только по датасетам
```

Получить справочник категорий типов данных (GET /data_type_category)

Уровень доступа	**read**
Request Body	**-**

Возвращает справочник дефолтных типов данных axiom для типов данных БД.

Т.е. для каждого типа данных в терминах БД определяется тип данных axiom, используется для авторазметки столбцов таблиц. Кроме типа данных по умолчанию определяется также категория типа данных - это группы типа "число", "строка" и т.п.

2.8 Loader

2.8.1 Create

Создание загрузчика (POST /loader/create)

Осуществляется регистрация данных и их загрузка в контур приложения. Загрузка осуществляется только для источников: файл, таблица, sql-скрипт.

ЗАГРУЗКА ФИЧЕЙ

Уровень доступа	load
Request Body	%loader_feature

Регистрация новых версий переменных.

ЗАГРУЗКА ДАТАСЕТА

Уровень доступа	load
Request Body	%loader_keys

В каталог датасетов добавляется новый датасет.
В процессе регистрации датасета через загрузчик происходит автоматическое определение фичей и их `fs_type`.
Фичами считаются все поля датасета, которые не участвовали в маппинге ключей сущности.

NB! Загрузка датасета недоступна из sql-скрипта.

ЗАГРУЗКА СВЯЗИ СУЩНОСТЕЙ (РУЧНАЯ)

Уровень доступа	load
Request Body	%loader_entity_link

В каталог связей сущностей добавляется новая пользовательская связь.

Создание загрузчика из другого загрузчика (POST /loader/create_from/{loader_rk})

Параметр	Описание
loader_rk: int	Суррогатный ключ загрузчика

Создается загрузчик типа external (ссылка на таблицу), где источником является таблица-приемник из другого загрузчика.

Запрос на создание загрузчика - такой же, как и для `POST /loader/create`, при этом:

- Поле `LOADER_KIND` игнорируется - создаваемый загрузчик всегда `external`
- Поля `LOADER_TABLE` / `FILE_DESC` / `LOADER_SQL` игнорируются - в качестве источника всегда используется таблица-приемник исходного загрузчика

2.8.2 Delete

Удаление загрузчиков (DELETE /loader)

Уровень доступа	**load**
Request Body	**-**

В фоновом режиме удаляет загрузчики вместе с:

- версиями переменных (для загрузчиков готовых переменных/предагрегатов)
- связью (для загрузчиков связей)
- датасетом (для загрузчиков сегмента)
- данными

Правила удаления данных регулируются с помощью параметра force_flg:

- Если force_flg = true, то удаляются таблицы источника и приемника загрузчика без ограничений
- Если force_flg = false (значение по умолчанию), то удаляются таблицы источника и приемника за исключением:
- таблиц external-загрузчиков
- таблиц-источников table-загрузчиков

2.8.3 Edit

Редактировать загрузчик (PATCH /loader/{loader_rk})

Уровень доступа	**load**
Request Body	-

Изменяет имя, описание и расписание загрузчика.

При редактировании накладываются все те же ограничения и проверки что и при создании загрузчика.

Пересоздать загрузчик (PUT /loader/{loader_rk})

Уровень доступа	**load**
Request Body	-

Info

Суррогатные ключи сохраняются только для тех полей источника, которые были в маппинге исходного загрузчика и присутствуют в новом. Если поле отсутствует в новом маппинге, информация о суррогатном ключе версии переменной для такого поля удаляется из метаданных

Синхронно удаляет и создаёт новый загрузчик взамен удаленного, при этом сохраняются:

1. Суррогатный ключ загрузчика `loader_rk`
2. Суррогатный ключ версий переменных `column_rk` (для загрузчика переменных).
3. Суррогатный ключ датасета `dataset_rk` (для загрузчика сегмента)
4. Суррогатный ключ связи `link_rk` (для загрузчика связи)

Ограничения:

1. Пересоздать можно только external-загрузчики
2. Тип источника изменить нельзя (новый загрузчик тоже должен быть external!)

Запустить загрузчик (POST /loader/{loader_rk}/execute)

Уровень доступа	**transform**
Request Body	-

Выполняет запуск загрузчика.

Алгоритм работы в зависимости от источника:

- **Таблица (ссылка)**: Ничего не делает, возвращает успешный ответ
- **Таблица (загрузка), Файл, SQL**: Выполняется новый запуск загрузки данных в store

Для загрузчика из файла можно передать новый файл для загрузки данных из него.

Активировать загрузчик (POST /loader/activate)

Deprecated

Деактивировать загрузчик (POST /loader/deactivate)

Deprecated

2.8.4 Info

Получить список загрузчиков (GET /loader)

Уровень доступа	**read**
Request Body	**-**

Возвращает полный список загрузичков с расширенной информацией по ним.

Для каждого загрузчика выводится информация о статусе загрузчика (STATUS) и статусе его последнего запуска (LAST_RUN_STATUS):

Статус	Описание
new	Объект зарегистрирован, но задание расчета не запускалось
running	Задание расчета выполняется
success	Задание расчета успешно завершено
failed	Задание расчета завершилось с ошибкой

Метод поддерживает фильтрацию результирующего списка по значениям параметров, где для параметров имени и описания доступен исключающий поиск с добавлением символа `^` в начало значения маски фильтрации.

Реализована возможность получать список загрузчиков порциями с возможностью регулирования размера выборки.

Получить пагинированный список загрузчиков (GET /loader/page)

Уровень доступа	**read**
Request Body	**-**

Метод является копией оригинального `GET /loader` с добавлением аргументов пагинации `limit` и `offset` для регулирования выборки.

Параметры `limit` и `offset` отвечают за размер и сдвиг выборки соответственно. При значении `limit: null` возвращается дефолтное количество объектов, которое определяется параметром `APP__DEFAULT_ITEMS_LIMIT`. Максимальное значение `limit` определяется параметром `APP__MAX_ITEMS_LIMIT`.

Получить детальную информацию о загрузчике (GET /loader/{loader_rk})

Уровень доступа	**read**
Request Body	**-**

Возвращает атрибуты загрузчика и загружаемой сущности (версии переменных, связь зерен, сегмент).

Параметр	Описание
loader_rk: int	Суррогатный ключ загрузчика

Получить статус загрузчика (GET /loader/{loader_rk}/status)

Уровень доступа	**read**
Request Body	**-**

Возвращает актуальный статус загрузчика.

Параметр	Описание
loader_rk: int	Суррогатный ключ загрузчика

2.9 Entity

2.9.1 Create

Создание сущности (POST /entity)

Уровень доступа	load
Request Body	-

В каталог сущностей добавляется новая сущность с выбранным набором ключей. При создании сущности можно использовать как существующие ключи, так и создать новые. Новая сущность должна иметь не менее одного ключа.

При создании сущности новые ключи добавляются в каталог ключей сущностей.

Требования и ограничения:

- Создавать ключи отдельно от сущностей нельзя.
- Разрешается создавать сущности с набором ключей в точности, как у уже существующей сущности.
- Название и сущности и ключа должно соответствовать маске \^[a-zA-Z][a-zA-Z0-9_]*$.
- Название сущности должно быть уникально в разрезе каталога сущностей.
- Название ключа должно быть уникально в разрезе каталога ключей.

Для сущности можно задать список тегов. Отношение сущностей к тегам = M:N - один тег может быть проставлен на нескольких сущностях. Каталог сущностей может быть отфильтрован по тегу/списку тегов.

NB! Метод недоступен во время выполнения задачи импорта.

СВЯЗЬ СУЩНОСТЕЙ (АВТОМАТИЧЕСКАЯ)

Создание автоматической связи осуществляется во время создания новой сущности. Пользователь не может создать автоматическую связь сущностей отдельно от процесса создания сущности.

Автоматическая связь между двумя сущностями появляется в случае, если у созданной сущности есть пересечение по ключам с уже существующей сущностью в каталоге.

Возможные типы автоматической связи:

- 1:1 связь образуется, когда обе сущности имеют одинаковый набор ключей.
- 1:M связь образуется, когда ключи одной из сущностей полностью входят в состав другой сущности. В таком случае "`parent`" (1) будет сущность, ключи которой полностью входят в состав другой сущности, которая будет "`child`" (M).
- M:N связь образуется, когда обе сущности имеют минимум по одному ключу, не входящих в набор другой сущности.

2.9.2 Delete

Удалить сущность (DELETE /entity)

Уровень доступа	**load**
Request Body	**-**

Удаляет сущность. Вместе с сущностью также удаляются её ключи, которые не используются в других сущностях.

При этом у сущности в каталоге должны отсутствовать неудаленные загрузчики. Если это условие не выполняется, то метод возвращает ошибку.

При передаче нескольких сущностей: если недоступно удаление хотя бы одной сущности из списка, то не удаляется ни одна сущность из списка.

2.9.3 Edit

Изменить сущность (PATCH /entity/{entity_rk})

Уровень доступа	**load**
Request Body	**-**

Изменение названия, описания или тегов существующей сущности.

Изменить ключ сущности (PATCH /entity_key/{entity_key_rk})

Уровень доступа	**load**
Request Body	**-**

Изменение названия и/или описания существующего ключа.

2.9.4 Info

Получить список сущностей (GET /entity)

Уровень доступа	**read**
Request Body	-

Возвращается полный список существующих сущностей со всей метаинформацией и списком ключей. Метод поддерживает фильтрацию возвращаемого списка по параметрам сущности и ключа, где для параметров имени и описания доступен исключающий поиск с добавлением символа `^` в начало значения маски фильтрации.

Реализована возможность получать список сущностей порциями с возможностью регулирования размера выборки.

Получить пагинированный список сущностей (GET /entity/page)

Уровень доступа	**read**
Request Body	-

Метод является копией оригинального `GET /entity` с добавлением аргументов пагинации `limit` и `offset` для регулирования выборки.

Параметры `limit` и `offset` отвечают за размер и сдвиг выборки соответственно. При значении `limit: null` возвращается дефолтное количество объектов, которое определяется параметром `APP__DEFAULT_ITEMS_LIMIT`. Максимальное значение `limit` определяется параметром `APP__MAX_ITEMS_LIMIT`.

Получить подробную информацию о сущности (GET /entity/{entity_rk})

Уровень доступа	**read**
Request Body	-

Для выбранной сущности из каталога возвращается полная метаинформация о самой сущности и ее ключах.

Получить список ключей сущности (GET /entity_key)

Уровень доступа	**read**
Request Body	-

Возвращается полный список существующих ключей с метаинформацией. Метод поддерживает фильтрацию возвращаемого списка по значениям параметрам ключа, где для параметров имени и описания доступен исключающий поиск с добавлением символа `^` в начало значения маски фильтрации.

Получить подробную информацию о ключе сущности (GET /entity_key/{entity_key_rk})

Уровень доступа	**read**
Request Body	-

Для выбранного ключа сущности из каталога возвращается полная метаинформация о нем.

2.10 Entity Link

2.10.1 Create

Создать автоматическую связь сущностей

Автоматические связи создаются при создании новой сущности.

Создать ручную связь сущностей

Ручная связь создается через создание загрузчика связи.

Добавить цепочку связей в справочник (POST /ref/link_chains)

Уровень доступа	**load**
Request Body	**-**

Добавляет новую запись в справочник цепочек связей, при этом валидация на корректность цепочки не производится.

2.10.2 Delete

Удалить связи сущностей (DELETE /entity_link)

Уровень доступа	**load**
Request Body	-

В фоновом режиме удаляет связи сущностей. Для связей типа `custom` удаляется соответствующий загрузчик.

Удалить цепочки связей из справочника (DELETE /ref/link_chains)

Уровень доступа	**load**
Request Body	-

Удаляет из справочника цепочки связей с заданными именами.

2.10.3 Edit

Изменить связь сущностей (PATCH /entity_link/{link_rk})

Уровень доступа	**load**
Request Body	**-**

Изменяет существующую связь. Допускается:

1. Изменить основную информацию - название, описание, мощность (1:1, 1:M, M:N)
2. Поменять местами родительскую и дочернюю сущности

Изменить запись о цепочке связей в справочнике (PATCH /ref/link_chains/{name})

Уровень доступа	**load**
Request Body	**-**

Позволяет изменить имя, последовательность связей в цепочке, описание или признак видимости цепочки.

2.10.4 Info

Получить список связей сущностей (GET /entity_link)

Уровень доступа	**read**
Request Body	**-**

Возвращается полный список существующих связей сущностей со всей метаинформацией по связи и сущностям, участвующих в связи.

Реализована возможность получать список связей порциями с возможностью регулирования размера выборки.

Получить пагинированный список связей сущностей (GET /entity_link/page)

Уровень доступа	**read**
Request Body	**-**

Метод является копией оригинального `GET /entity_link` с добавлением аргументов пагинации `limit` и `offset` для регулирования выборки.

Параметры `limit` и `offset` отвечают за размер и сдвиг выборки соответственно. При значении `limit: null` возвращается дефолтное количество объектов, которое определяется параметром `APP__DEFAULT_ITEMS_LIMIT`. Максимальное значение `limit` определяется параметром `APP__MAX_ITEMS_LIMIT`.

Получить подробную информацию по связи сущностей (GET /entity_link/{link_rk})

Уровень доступа	**read**
Request Body	**-**

Для выбранной связи из каталога возвращается полная метаинформация о ней и ее сущностях, а также список связанных объектов - датасетов и загрузчиков.

Получить список цепочек связей зерен (GET /entity_link/chains/{entity_rk})

Уровень доступа	**read**
Request Body	**-**

Цепочка связей представляет собой корректную последовательность связей сущностей с целью осуществления перехода от одного сущности к другой.

Для выбранной сущности (`entity_rk`) из каталога возвращается список цепочек связей с другими сущностями (ограничения см. ниже). Цепочки связей используются при конфигурации объектов в датасете, при этом допускается использовать комбинации полученных цепочек.

Правила отбора цепочек:

1. Одна и та же связь не может быть использована дважды;
2. Две автоматические связи подряд запрещены;
3. Цепочка обрывается при возвращении к исходной сущности - связи с исходным сущностью могут быть только в начале и в конце цепочки

При параметре `target_entity_rk = None` возвращаются все возможные цепочки до всех сущностей, к которым они могут быть составлены. При заданном `target_entity_rk` возвращаются цепочки от выбранной сущности `entity_rk` до указанной в `target_entity_rk`.

Если методу передан параметр `only_visible_flg = true`, то выводятся только те цепочки, по которым есть запись в справочнике цепочек с установленным признаком `visible_flg = true`.

Получить пагинированный список цепочек связей зерен (GET /entity_link/chains/{entity_rk}/page)

Уровень доступа	**read**
Request Body	**-**

Метод является копией оригинального `GET /entity_link/chains/{entity_rk}` с добавлением аргументов пагинации `limit` и `offset` для регулирования выборки.

Параметры `limit` и `offset` отвечают за размер и сдвиг выборки соответственно. При значении `limit: null` возвращается дефолтное количество объектов, которое определяется параметром `APP__DEFAULT_ITEMS_LIMIT`. Максимальное значение `limit` определяется параметром `APP__MAX_ITEMS_LIMIT`.

Получить справочник цепочек связей (GET /ref/link_chains)

Уровень доступа	**read**
Request Body	**-**

Справочник позволяет сохранять целые цепочки связей под одним именем и управлять видимостью этой цепочки.

Запись в справочнике состоит из следующих атрибутов:

Атрибут	**Описание**
name	имя цепочки
description	описание цепочки
chain	последовательность ENTITY_LINK_RK (связей сущностей), образующих цепочку
visible_flg	выводить ли цепочку в списке доступных цепочек

Реализована возможность получать список цепочек порциями с возможностью регулирования размера выборки.

2.11 Feature

2.11.1 Create

Создать фичу (POST /feature)

Уровень доступа	**load**
Request Body	**-**

Создает в каталоге переменных новую фичу. Имя должно быть уникальным.

2.11.2 Delete

Удалить фичу (DELETE /feature)

Уровень доступа	**load**
Request Body	**-**

Удаляет фичи по переданному списку суррогатных ключей.

Возможно удаление только тех фичей по которым нет версий. Если хотя бы для одной из фичей это требования не выполняется, то вызов метода завершается ошибкой.

2.11.3 Edit

Редактировать фичу (PATCH /feature/{feature_rk})

Уровень доступа	**load**
Request Body	**-**

Изменяет имя и описание фичи. При редактировании накладываются все те же ограничения и проверки, что и при создании.

Редактировать версию фичи (PATCH /feature_column/{column_rk})

Уровень доступа	**load**
Request Body	**-**

Изменяет имя, описание, тип (feature/domain) и теги у версии фичи.

2.11.4 Info

Получить детальную информацию о фиче (GET /feature/{feature_rk})

Уровень доступа	**read**
Request Body	-

Возвращает атрибуты фичи и список её версий, так же с атрибутами.

Получить список версий фичей (GET /feature)

Уровень доступа	**read**
Request Body	-

Возвращает полный список версий фичей с расширенной информацией по ним. Метод поддерживает фильтрацию результирующего списка по значениям параметров, указанных ниже, где для параметров имени (фичи и версии) и описания (фичи и версии) доступен исключающий поиск с добавлением символа `^` в начало значения маски фильтрации.

Параметры фильтрации:

- entity_rk: List[int] - суррогатный ключ сущности
- table_rk: int - суррогатный ключ таблицы в store
- name: mask - название фичи
- description: mask - описание фичи
- column_rk: int - суррогатный ключ версии фичи
- column_name: mask - название версии фичи
- column_desc: mask - описание версии фичи
- column_kind: str - тип версии фичи (фича/домен)
- column_data_type_rk: int - суррогатный ключ типа данных версии фичи
- tags: list[str] - список тегов (фильтрация по частичному пересечению)

Если у фичи нет ни одной версии, то вернется один элемент на фичу, где будут заполнены только поля, относящиеся к самой фиче. Атрибуты версии фичи не будут заполнены.

Реализована возможность получать список версий фичей порциями с возможностью регулирования размера выборки.

Получить пагинированный список версий фичей (GET /feature/page)

Уровень доступа	**read**
Request Body	-

Метод является копией оригинального `GET /feature` с добавлением аргументов пагинации `limit` и `offset` для регулирования выборки.

Параметры `limit` и `offset` отвечают за размер и сдвиг выборки соответственно. При значении `limit: null` возвращается дефолтное количество объектов, которое определяется параметром `APP__DEFAULT_ITEMS_LIMIT`. Максимальное значение `limit` определяется параметром `APP__MAX_ITEMS_LIMIT`.

Получить список версий фичей 2.0 (GET /feature/light)

Уровень доступа	**read**
Request Body	**-**

Возвращает полный список версий фичей с ограниченным набором атрибутов. Метод поддерживает фильтрацию результирующего списка по значениям параметров, указанных ниже, где для параметров имени (фичи и версии) и описания (фичи и версии) доступен исключающий поиск с добавлением символа `^` в начало значения маски фильтрации.

Параметры фильтрации:

- entity_rk: List[int] - суррогатный ключ сущности
- table_rk: int - суррогатный ключ таблицы в store
- name: mask - название фичи
- description: mask - описание фичи
- column_rk: int - суррогатный ключ версии фичи
- column_name: mask - название версии фичи
- column_desc: mask - описание версии фичи
- column_data_type_rk: int - суррогатный ключ типа данных версии фичи
- table_kind: str - тип данных в таблице
- tags: list[str] - список тегов (фильтрация по частичному пересечению)

Если у фичи нет ни одной версии, то вернется один элемент на фичу, где будут заполнены только поля, относящиеся к самой фиче. Атрибуты версии фичи не будут заполнены.

Реализована возможность получать список версий фичей порциями с возможностью регулирования размера выборки.

Получить список загруженных дат актуальности по версии фичи (GET /feature/history/{feature_rk})

Уровень доступа	**read**
Request Body	**-**

Возвращает список периодов актуальности за которые есть данные по версии фичи. Результат представляет собой список интервалов [FROM_DTTM, TO_DTTM]

3. Product Map 1.0

3.1 Product Map 1.0

Code	Description
+M01X	Transform
+M02X	Automap
+M03X	Automap File
+D01C	Create Dataset
+D02R	Get Datasets
+D03D	Delete Datasets
+D04R	Get Dataset Info
+D05U	Update Dataset
+D06R	Get Datasets Jobs
+D07R	Get Dataset Health
+L01C	Create Dataset Links
+L02R	Get Dataset Links
+L03U	Update Dataset Link
+L04D	Delete Dataset Links
+F02R	Get Features
+F04R	Get Feature Info
+F05U	Update Feature
+I02R	Get Instances
+I04R	Get Instance Info
+I05U	Update Instance
+Q01R	Get Dataset Statistics
+Q02D	Delete Dataset Statistics
+Q03R	Get Dataset Common Statistics
+Q04D	Delete Dataset Common Statistics
+Q05R	Get Dataset Instance Statistics
+Q06D	Delete Dataset Instance Statistics
+Q07R	Get Dataset Feature Statistics
+Q08D	Delete Dataset Feature Statistics
+K01C	Create Entity Key
+K02R	Get All Entity Keys
+K03D	Delete Entity Keys
+K04R	Get One Entity Key
+K05U	Update Entity Key
+E01C	Create Entity

Code	Description
+E02R	Get All Entities
+E03D	Delete Entities
+E04R	Get One Entity
+E05U	Update Entity
+Z01R	Get System info
+Z02R	Get System health
+Z03R	Get System metrics
+Z04R	Get System loggers
+Z05R	Get Main Logger Level
+Z06R	Get Worker Readiness
+S01C	Create Data Source
+S02R	Get Data Sources
+S03D	Delete Data Sources
+S04R	Get Data Source Info
+S05U	Update Data Source
+S07R	Get Data Source's tables
+S08R	Get Data Source's table columns
+S09R	Validate Data Source
+S10R	Get Data Source's table preview
+R01C	Create Data Type
+R02R	Get Data Types
+R03U	Recreate Data Types
+R04D	Delete Data Types
+R05C	Create Data Type Implementation
+R06R	Get all Data Type Implementation
+R07U	Recreate Data Type Implementation
+R08D	Delete Data Type Implementation
+R09C	Set default Data Type
+R10R	Get default Data Types
+R11U	Recreate Default Data Types
+R12D	Delete Default Data Types
+R13C	Set Indicator for Data Type
+R14R	Get Indicators for Data Types
+R15U	Recreate Indicators for Data Types

Code	Description
+R16D	Remove Indicators for Data Type
+X01C	Create Execution Profile
+X02R	Get All Available Execution Profiles
+X03U	Update Execution Profile
+X04D	Delete Execution Profile
+J01C	Create Job
+J02R	Get Jobs
+J03D	Delete Jobs
+J04X	Execute Job
+J05R	Get Job Info
+J06U	Update Job
+J07C	Render Job Script
+J08R	Get Job Runs
+J09R	Get Job Status
+J10R	Get Job Run Info
+J11X	Stop Running Job
+J12C	Set Schedule
+J13D	Delete Schedules
+J14R	Get All Schedules
+J15R	Get Schedule
+P01R	Get Permissions
+P02U	Set Permissions
+P03D	Delete Permissions
+P04R	Get User Information
+T01R	Get All Tags
+T02D	Delete Tags
+Y01C	Upload file
+Y02R	Get list of files
+Y04D	Delete files
+Y05C	Create download link for file
+Y06C	Create text file
+Y07R	Get text file content
+Y08R	Describe file
+Z01X	Reset Metastore

Code	Description
+Z03X	Reset Airflow History
+Z04X	Ref Init
+Z05X	Migrate Metastore
+Z06X	Migrate Dags
+Z07X	Reset system refs
+Z08R	Get actual logger profile
+Z09C	Create logger profile
+Z10R	Download logs
+Z11R	Get logger profiles
+Z12R	Get logger profile
+Z13U	Update logger profile
+Z14R	Get Task Information From Executor
+Z15R	Get application lock
+Z16D	Release application lock

3.2 Формат JSONQ

Модуль jsonq создан для удобного создания простых SQL конструкций по текстовой конфигурации в формате JSON

3.2.1 Спецификация формата

```
{
    "SELECT":   ["expression", ...],
    "FROM":     ["table_element", ...],
    "WHERE":    ["expression", ...],
    "GROUP BY": ["expression", ...],
    "HAVING":   ["expression", ...],
    "ORDER BY": ["expression", ...],
    "OPTIONS":  ["option", ...],
    "LIMIT":    int,
    "OFFSET":   int
}
```

```
expression
    =
    | <str>                       # имя столбца; короткий формат
    | [ <expr>, <as> ]            # выражение с алиасом; формат list
    ;

<expr> : любое поддерживаемое выражение в fs_alchemy.parsing.expression
```

```
table_element
    =
    | <str>                       # имя таблицы
    | [ <tab_expr>, <as>, <on>, <on>, ... ]  # имя таблицы, алиас, условие для join; формат list
    ;

<tab_expr> : любое поддерживаемое выражение в fs_alchemy.parsing.segment
```

```
option = 'distinct' | 'inner' ;
```

```
// если в секции всего один элемент и это str, то скобки списка можно опустить
{
    ...
    "SECTION1": ["str", ...],
    "SECTION2": "str"
    ...
}
```

Поддержка переменных в запросе

Элементы \<expr>, \<tab_expr>, \<as>, \<on> поддерживают подстановку перенных. Переменной является alphanumeric текст в фигурных скобках, например "{prev_task}". Виды поддерживаемых переменных:

- **простые типы**: *int*, *float*, *bool*, *str* - по ним подстановка осуществляется методом str.replace()
- **сложные**: *SQLAlchemy Selectable* - поддерживается только в \<tab_expr> и только если выражение состоит ровно из одной переменной

3.2.2 Примеры запросов

```
// базовые SELECT
{
    "SELECT": "*",
    "FROM": "schema1.users"
}
{
    "SELECT": ["name", "gender"],
    "FROM": ["schema1.users"]
}
{
    "SELECT": [["1", "const_int"], ["'abc'", "const_str"]]
}
```

```
// алиасы
{
    "SELECT": ["u.name", "u.gender"],
    "FROM": [["schema1.users", "u"]]
}
```

```
{
    "SELECT": [
        ["u.name", "user_name"],
        "u.gender"
    ],
    "FROM": [["schema1.users", "u"]]
}
```

```
// джойны
{
    "SELECT": "*",
    "FROM": [
        ["schema1.users", "u"],
        ["schema1.salary", "s", "u.name = s.name"], // LEFT by default
    ]
}
{
    "OPTIONS": "inner",
    "SELECT": "*",
    "FROM": [
        ["schema1.users", "u"],
        ["schema1.salary", "s", "u.name = s.name"], // INNER
    ]
}
```

```
// distinct
{
    "OPTIONS": "distinct",
    "SELECT": "name",
    "FROM": "schema1.users"
}
```

```
// фильтры
{
    "SELECT": "*",
    "FROM": "schema1.users",
    "WHERE": "name = 'vasya'"
}
{
    "SELECT": "*",
    "FROM": "schema1.users",
    "WHERE": ["name = 'vasya'", "age > 18"] // условия через AND
}
```

```
// GROUP BY, HAVING, ORDER BY
{
    "SELECT": ["name", "gender", "count(1)"],
    "FROM": "schema1.users",
    "GROUP BY": ["name", "gender"],
    "HAVING": ["count(1) > 0"],
    "ORDER BY": "name" // ASC
}
```

```
// flat params
// column_name = 'name'
// salary_table = 'salary'
// min_pay = 1000
{
    "SELECT": "u.{column_name}",
    "FROM": [
        ["users", "u"],
        ["{salary_table}", "s", "u.{column_name} = s.{column_name}"],
    ],
    "WHERE": "gender = 'm' or salary_amt>{min_pay}",
}
```

```
// table params
{
    "id": "salary_subquery",
    "type": "cte",
    "body": {
        "SELECT": ["name", "salary_amt"],
        "FROM": "dwh.salary",
        "WHERE": ["salary_src = 'gbc'"]
    }
},
{
    "id": "query",
    "type": "cte",
    "body":{
        "SELECT": "u.name",
        "FROM": [
            ["users", "u"],
            ["{salary_subquery}", "s", "u.name = s.name"],
        ],
        "WHERE": "gender = 'm' or salary_amt > 1000",
    }
}
```

3.3 Actions

3.3.1 Transform

Transform

ЗАПУСТИТЬ ТРАНСФОРМАЦИЮ (POST /TRANSFORM)

Уровень доступа	**transform**
Request Body	**%transform**
Response Body	**%transform_summary**

Более верхнеуровневая обертка над job, которая позволяет в одно действие создать и запустить задачу на расчет.

В качестве параметров запроса принимает:

- `execute` - признак запуска задачи сразу после регистрации
- `source_rk` - суррогатный ключ источника, на котором необходимо выполнить задачу. Запуск узлов `consturctor` корректно выполняются только на источнике -1. Компиляция задания производится как для источника -1 (валидации, имплементация типов данных и т.д.), хотя само задание выполняется на том источнике, который передан в параметры запроса. **!NB Джентльменское соглашение** - запускать узлы `constructor` только на источнике -1.

По сравнению с `job` метод transform принимает композитные узлы `constructor`, `service` и прочие.

Для узлов `constructor` перед созданием джоба производятся следующие дополнительные действия:

- Результат работы узла регистрируется как датасет в meta, если **не** указана опция `drop_at_end=false`
- Узел заменяется на соответствующую ему подцепочку, состоящую только из базовых узлов.

Если в transform не указан явно `"dispose"`, то он устанавливается равным `"smart"`.

Для заданий, в результате выполнения которых не регистрируются новые датасеты (например, расчет статистик, копирование), значение `"dispose"` по умолчанию `"on_success"`.

Узел `constructor`

Как и `select`, узел `constructor` используется для трансформации данных внутри БД. Однако в отличие от `select`, узел `constructor` является композитным и перед исполнением должен быть преобразован в подграф из базовых узлов. Это происходит перед созданием задания: все найденные композитные узлы заменяются на соответствующие им микро-задания, которые встраиваются в родительскую: `"id"` результирующего узла микро-задания будет совпадать с `"id"` исходного узла `constructor`, а `"id"` вспомогательных узлов сформированы по принципу `"{constructor_task_id}_{unique_postfix}"`.

```
{
   ~"id": str,  # автогенерация внутри блоков par и seq, если не указан явно
   *"type": "constructor",
   *"body": {...},  # формат body определяется значением options.format
   *"options": {
        # формат конструктора (подробнее см. Форматы)
       *"format": str,
        # тип объекта в БД, по умолчанию {auto}; датасеты регистрируются в метаданных только для "table"
        "to": "table" | "view" | "cte" | "subquery" | "{auto}",
        # имя схемы в БД, по умолчанию {auto}
        "to_schema": str | "{auto}",
        # имя объекта в БД, по умолчанию {auto}
        "to_name": str | "{auto}",
        # желательный тип промежуточных объектов в БД, по умолчанию {auto}
        "aux_to": "table" | "view" | "cte" | "subquery" | "{auto}",
        # удалить ли промежуточные объекты после выполнения цепочки, по умолчанию true
        "aux_drop_at_end": bool,
        # удалить ли промежуточные объекты после выполнения цепочки, по умолчанию true
        "stats": list[enum]
    }
}
```

Форматы:

- fat_compose
- fat_map
- fat_merge
- fat_split
- fat_impute
- fat_resample
- fat_union_all

В процессе компиляции таких узлов один узел constructor превращается в следующую цепочку:

1. Расчет объекта с данными (table/view/cte)
2. Если объект регистрируется как датасет (`options.drop_at_end=false` или отсутствует), то по получившемуся датасету производятся дополнительные обновления в метаданных:

a. расчет базовых статистик, указанных в `options.stats`

b. расчет колоночных статистик (отключаемо через `ENGINE__SKIP_FEATURES_STATS`)

c. установка `dataset.empty_flg=False`

d. вставка новых записей в `dataset_hist`

e. установка `dataset.status='success'`

Доступные базовые статистики:

- `ROWS_COUNT` - количество строк в датасете;
- `ENTITY_COUNT` - количество значений зерна в датасете
- `REPORT_DTTM_COUNT` - количество отчетных дат (срезов) в датасете

Узел `service`

Узел `service` используется для выполнения сервисных операций над существующими датасетами. В отличие от узлов `constructor`, по результатам работы узлов `service` не происходит создания новых датасетов.

```
{
    ~"id": str,  # автогенерация внутри блоков par и seq, если не указан явно
    *"type": "service",
    *"body": {...},  # формат body определяется значением options.format
    *"options": {
        *"format": str  # вызываемый сервис
     }
}
```

Форматы:

- copy
- fat_stats

Узел `ingest`

Узел `ingest` создает новый датасет в области `v100` и ассоциирует его с заданием (Job) на загрузку

```
{
    ~"id": str,  # автогенерация внутри блоков par и seq, если не указан явно
    *"type": "ingest",
    *"body": {...},  # формат body определяется значением params.format
    *"params": {
        *"format": str,     # формат загрузки
         "name": str,       # имя датасета
         "desc": str,       # описание датасета
         "db_schema": str,  # схема в БД, где создать таблицу, по умолчанию {auto}
         "db_table": str,   # имя таблицы, по умолчанию {auto}
         "origin": str,     # метка о происхождении, по-умолчанию "ingest/<format>"
     }
}
```

Форматы:

- file
- table
- sql

Узел `extract:file`

Узел создает файл в S3 хранилище с данными из таблицы датасета. Исходный датасет ассоциируется с заданием на выгрузку данных ролью `extract`

```
{
  *"type": "extract:file",
  *"body": {
     *"dataset": int | str,  # dataset_rk | "dataset name" | "{task_id}"
     *"path": str            # адрес файла, например 's3://foo.txt'
   }
}
```

1. Все данные датасета, указанные в dataset, выгружаются в файл, указанный в path
2. Возможные форматы ссылки на dataset:

- `integer`: взять датасет с таким dataset_rk
- `"text"`: взять датасет таким name
- `"{text}"`: если ссылка представляет собой имя взятое в фигурные скобки, то взять датасет, формируемый в таске с task_id=text

Узел `stats`

Узел `stats` рассчитывает статистики над датасетом и сохраняет их в metastore.

```
{
    ~"id": str,  # автогенерация внутри блоков par и seq, если не указан явно
    *"type": "stats",
    *"body": {...}
}
```

Подробнее.

copy

Формат реализует функциональность копирования данных датасета в любые таблицы внутри источника исполнения задания, к которым есть доступ у приложения.

> **Info**
>
> Если таблица-приемник существует и режим копирования = "insert", то все копируемые поля должны присутствовать в приемнике с такими же названиями и совместимыми типами данных, иначе процесс копирования упадет с ошибкой. В момент регистрации трансформа структура таблицы приемника не проверяется.

Схема:

```
{
    *"dataset": str,                # ссылка на датасет {1} или узел constructor {task_id}
    *"db_schema": str,              # схема таблицы-приемника
    *"db_table": str,               # название таблицы-приемника
    *"mode": "replace" | "insert",  # режим копирования
    *"columns": [str | "{features}" | "{date}" | "{keys}" | "{tech}"]  # копируемые поля
}
```

РУКОВОДСТВО ПО ЗАПОЛНЕНИЮ ПОЛЕЙ СХЕМЫ

- `dataset` - ссылка на копируемый датасет или узел constructor
- `db_schema` - имя схемы таблицы приемника в БД
- `db_table` - имя таблицы приемника в схеме
- `mode` - режим копирования данных:
- `"replace"` - перезапись приемника данными датасета (drop + create)
- `"insert"` - вставка записей из датасета в приемник без предварительной очистки приемника (insert)
- `columns` - список копируемых полей датасета с поддержкой параметров. Список доступных параметров приведен ниже. При этом, если для параметра в датасете не найдены поля, то такой параметр игнорируется, ошибка не возникает.
- `"{date}"` - поле отчетной даты (`report_dttm`)
- `"{keys}"` - все поля ключей сущности
- `"{tech}"` - все технические поля (`_run_rk`, `_created_dttm`)
- `"{features}"` - все переменные датасета без технических полей, ключей сущности и отчетной даты

Алгоритм:

1. Если таблица приемник отсутствует, то создается новая таблица с необходимыми полями и типами данных
2. Данные из датасета копируются в таблицу-приемник по полученному списку полей

fat_compose

Формат реализует функциональность конструктора датасетов.

Результатом работы узла является табличная структура, следующего формата:

Поле(-я)	Тип	Комментарий
Ключи сущности	`...`	Ключи из `entity_rk` датасета
`report_dttm`	`datetime`	Бизнес дата данных
Фичи датасета	`...`	Все что перечислено в `output`
`_run_rk`	`int`	Технический идентификатор запуска
`_created_dttm`	`datetime`	Техническая дата вставки

Схема:

```
{
   *"name": str,        # название датасета
    "description": str, # описание датасета
   *"entity_rk": int,   # сущность датасета
   *"date": "now()" | "today()" | "segment" | [str, ...],  # дата расчета
    "segment": str,     # id узла в формате {task_id} или ссылка на датасет в формате {dataset_rk},
                        # по которым фильтруем сущность
    "filter": str | <%tree>,         # bool выражение, которым фильтруем по значениям в переменных
    "output": "{all}" | [str, ...],  # список полей в результирующем селекте
   *"calc_units": [{
       *"code": str | <%tree>,  # выражение для расчета элемента, может содержать ссылки на:
                                # - переменные каталога "[links]{column_rk}",
                                # - ключи сущности из связи "[links]<key_rk>"
                                # - переменные сегмента {"feature"},
                                # - итоговые поля датасета:
                                #    - переменные "feature"
                                #    - ключи сущности датасета "entity_key_column"
                                #    - дату расчета "{report_dttm}"
        "alias": str,   # основной алиас переменной
        "agg":{  # настройка агрегации
           *"functions": [{  # агрегирующий функции
               *"name": str,  # имя функции
                "alias": str  # алиас функции
            }],
            "conditions": [{  # условия агрегации
               *"group": str,  # группа условий - тег, по которому группируются условия
               *"code": str | <%tree>,  # выражение условия, может содержать ссылки на:
                                        # - переменные "[links]{column_rk}"
                                        # - ключи сущности из связи "[links]<key_rk>"
                                        # - дату расчета "{report_dttm}
                                        # - поля версионности предагрегата {from_dttm}, {to_dttm}"
                "alias": str  # алиас условия
            }]
       }
    }]
}
```

РУКОВОДСТВО ПО ЗАПОЛНЕНИЮ ПОЛЕЙ СХЕМЫ

info: Поддерживаемые функции в выражениях code (в т.ч. функции агрегации) и filter см. тут

- `name` - название датасета, должно быть уникально во всем каталоге датасетов
- `description` - описание датасета
- `entity_rk` - основная сущность датасета (клиент, договор и т.д.)
- `segment` - датасет-источник значений сущности (подставляется в from sql-запроса). Допустимо использовать ссылку на узел constructor или суррогатный ключ датасета - `"{task_1}"`, `"{task_2}"`, `"{1}"`, `"{2}"`. Сущность датасета-сегмента должна совпадать с основной сущностью конфигурируемого датасета. Если не задан, в качестве сегмента будет использован реестр сущности по загрузчику из опции `fat_registry`.
- `date` - дата расчета, на которую рассчитываются данные в датасете. Допустимо использовать:
- `"segment"` - дата из сегмента (только если в поле `segment` указан датасет с датой)
- `"now()"` - актуальный срез, current_timestamp
- `"today()"` - текущая дата, current_date
- `["2023-01-01 00:00:00", "2023-06-01 23:59:59", ...]` - список конкретных дат. **NB! Добавилось ограничение, что длина списка должна быть = 1** в формате `YYYY-MM-DD hh:mm:ss`
- `calc_units` - конфигурация переменных датасета. Названия переменных датасета генерируются с помощью конкатенации алиасов всех составляющих элементов по следующей формуле: `calc_unit[agg][functions][alias] + calc_unit[alias] +`

`calc_unit[agg][conditions][alias]`. Подробнее см. "Пример конфигурации агрегата". Переменные датасета доступны для включения в output и filter.

- `code` - код расчета переменной. Допустимо использовать:
- Суррогатный ключ версии переменной из каталога (`column_rk`) - `"{1}"`, `"{2}"`, `"{3}"`. Если сущность версии переменной отличается от основной сущности конфигурируемого датасета, то требуется определить цепочку связей, по которой эту переменную взять в датасет. Для этого перед переменной необходимо указать список связей для перехода в порядке от основной сущности датасета к сущности переменной - `"[1,2,3]{1}"`, `"[1]{2}"`. Подробнее см. "Цепочки связей".
- Переменная из сегмента - `"{segment_feature}"`
- Ключ сущности из последней связи по цепочке - `"[1,2,3]<1>"`. Недопустимо использование связей, приводящих к размножению строк: 1->M (направление важно!) и M->N. Ограничение: последняя связь цепочки должна быть ручная (не автоматическая).
- Формула второго уровня, например: `"coalesce(dataset_feature_1, dataset_feature_2 + dataset_feature_3)"`. Формула второго уровня - это переменная, которая рассчитывается над другими полями датасета. В формуле второго уровня могут использоваться: другие переменные датасета (в т.ч. формулы второго уровня), поля ключей сущности и дата расчета - `"{report_dttm}"`.
- `alias` - основной алиас переменной
- `agg` - конфигурация агрегата. Настройка агрегации используется только для версий переменных из каталога и является допустимой и обязательной только если переменная отбирается из предагрегата или в цепочке для ее получения присутствует связь 1->M (направление важно!) или M->N.
- `functions` - настройка агрегирующих функций
- `name` - код агрегирующей функции. Помимо стандартных агрегирующих функций (`count`, `sum`, `avg`, etc.) можно использовать кастомные функции. Принцип работы кастомных функций такой же, как и для стандартных - они работают по данными предагрегата в разрезе ключа сущности. Список доступных кастомных функций:
- `count_distinct` - количество уникальных значений = count(distinct ...)
- `count_rows` - количество строк с подходящими условиями из conditions; при отсутствии таких строк возвращается 0
- `any` - если условия из conditions выполняются хотя бы для одной строки в группе, то 1, иначе 0
- `all` - если условия из conditions выполняются для всех строк в группе, то 1, иначе 0
- `alias` - алиас агрегирующей функции
- `conditions` - условия (подставляется внутрь агрегирующих функций в case)
- `group` - группа условий. Разные группы условий выполняются параллельно (И), внутри каждой группы условия перебираются. Подробнее см. "Пример конфигурации агрегата".
- `code` - код условия, допустимо использовать:
- Переменные из того же предагрегата, что и агрегируемая переменная - `"[1,2,3]{2}"`, `"[1]{3}"`
- Готовые переменные (не предагрегаты!) с использованием подцепочки агрегируемой переменной. Подцепочка должна входить в состав цепочки агрегируемой переменной начиная с первой связи. Примеры (подцепочка -> цепочка):
- **Ок:** `[1]` --> `[1,2,3]`
- **Ок:** `[1,2]` --> `[1,2,3]`
- **Ок:** `[1,2,3]` --> `[1,2,3]`
- **Не ок**: `[1,2]` -/> `[1,3,2]`
- **Не ок**: `[1,2]` -/> `[]` (отсутствие цепочки)
- Ключи из используемой цепочки связей или её подцепочки
- Дата расчета - параметр `"{report_dttm}"`
- Даты версионности (только для транзакционного предагрегата) - параметры `"{from_dttm}"` и `"{to_dttm}"`

- `output` - список переменных датасета из calc_unit для включения в поля датасета. Если передано '{all}', то в датасет включаются все переменные.
- `filter` - формула фильтрации датасета (подставляется в where sql-запроса). В формуле могут использоваться переменные датасета из calc_unit, ключи сущности датасета и дата расчета - `"{report_dttm}"`.

Цепочки связей

Правила формирования цепочки:

- Цепочка связей состоит из направленных связей между сущностями и соединяет сущность датасета с сущностью переменной или ключа.
- Исходная сущность первой связи цепочки должна совпадать с сущностью датасета.
- Исхдная сущность каждой последующей связи цепочки должна совпадать с целевой сущностью предыдущей связи.
- Целевая сущность последней связи цепочки должна совпадать с сущностью переменной или содержать в себе искомый ключ.

Ограничение: в цепочке нельзя использовать несколько автоматических связей подряд.

Пример использования цепочки

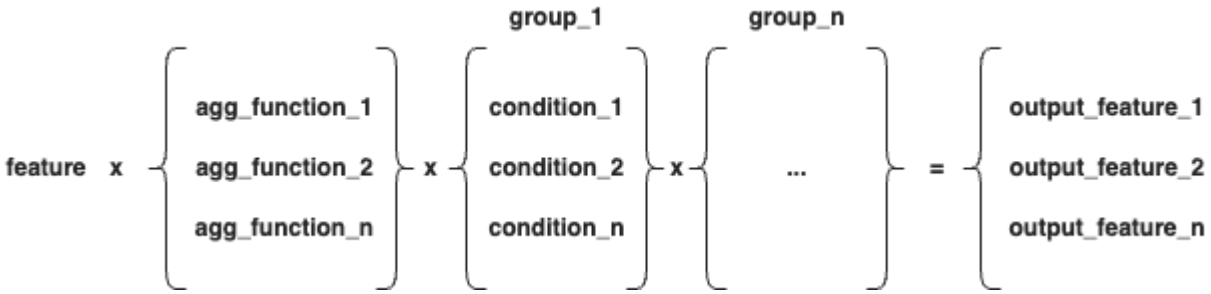
Допустим, имеется формула "[1, 2]{5}", где:

- Переменная {column_rk = 5} загружена под сущностью entity_rk = 3;
- Связь link_rk = 1 - связывает сущности entity_rk = 1 и entity_rk = 2;
- Связь link_rk = 2 - связывает сущности entity_rk = 2 и entity_rk = 3.

Выражение "[1, 2]{5}" читается как "получить переменную {column_rk = 5} по связям {link_rk = 1} -> {link_rk = 2}". Тогда на уровне сущностей цепочка выглядит следующим образом: 1 (сущность датасета) -> [1 -> 2] (связь link_rk = 1) -> [2 -> 3] (связь link_rk = 2) -> 3 (сущность переменной).

Пример конфигурации агрегата

В процессе конфигурирования агрегатов производится настройка агрегирующих функций и условий фильтрации агрегируемых данных. Итоговый набор переменных датасета генерируется по формуле:



Пример - конфигурация предагрегата trm_amt. Для упрощения вместо ссылок на переменные в примере используются их названия:

- Агрегирующие функции
- Функция 1
- code = `avg`
- alias = `"AVG"`
- Функция 2
- code = `count`
- alias = `"COUNT"`
- Группа условий `buy_time`
- Условие 1
- code = `time_cd = 'MORNING' or (time_cd is null and weekday_flg = true)`
- alias = `"MRNG"`
- Условие 2
- code = `time_cd = 'DAY' or (time_cd is null and coalesce(weekday_flg, false) = false)`
- alias = `"DAY"`
- Условие 3
- code = `time_cd in ('NIGHT', 'EVENING')`
- alias = `"NIGHT"`
- Группа условий `currency`
- Условие 1
- code = `curr_iso_cd = 'RUB'`
- alias = `"RUB"`
- Условие 2
- code = `curr_iso_cd = 'USD'`
- alias = `"USD"`

Итоговый набор переменных датасета:

1. `AVG_TRN_AMT_MRNG_RUB` (avg & (time_cd = 'MORNING' or (time_cd is null and weekday_flg = true)) and (curr_iso_cd = 'RUB'))
2. `AVG_TRN_AMT_MRNG_USD` (avg & (time_cd = 'MORNING' or (time_cd is null and weekday_flg = true)) and (curr_iso_cd = 'USD'))
3. `AVG_TRN_AMT_DAY_RUB` (avg & (time_cd = 'DAY' or (time_cd is null and coalesce(weekday_flg, false) = false)) and (curr_iso_cd = 'RUB'))
4. `AVG_TRN_AMT_DAY_USD` (avg & (time_cd = 'DAY' or (time_cd is null and coalesce(weekday_flg, false) = false)) and (curr_iso_cd = 'USD'))
5. `AVG_TRN_AMT_NIGHT_RUB` (avg & (time_cd in ('NIGHT', 'EVENING')) and (curr_iso_cd = 'RUB'))
6. `AVG_TRN_AMT_NIGHT_USD` (avg & (time_cd in ('NIGHT', 'EVENING')) and (curr_iso_cd = 'USD'))
7. `COUNT_TRN_AMT_MRNG_RUB` (count & (time_cd = 'MORNING' or (time_cd is null and weekday_flg = true)) and (curr_iso_cd = 'RUB'))
8. `COUNT_TRN_AMT_MRNG_USD` (count & (time_cd = 'MORNING' or (time_cd is null and weekday_flg = true)) and (curr_iso_cd = 'USD'))
9. `COUNT_TRN_AMT_DAY_RUB` (count & (time_cd = 'DAY' or (time_cd is null and coalesce(weekday_flg, false) = false)) and (curr_iso_cd = 'RUB'))
10. `COUNT_TRN_AMT_DAY_USD` (count & (time_cd = 'DAY' or (time_cd is null and coalesce(weekday_flg, false) = false)) and (curr_iso_cd = 'USD'))
11. `COUNT_TRN_AMT_NIGHT_RUB` (count & (time_cd in ('NIGHT', 'EVENING')) and (curr_iso_cd = 'RUB'))
12. `COUNT_TRN_AMT_NIGHT_USD` (count & (time_cd in ('NIGHT', 'EVENING')) and (curr_iso_cd = 'USD'))

Допустимые *options*

- `to` - тип объекта, который создается в БД. Возможны `"table", "view", "cte", "subquery", "{auto}"`. Если установлено "{auto}", то будет использовано значение default_to из job_options (по умолчанию совпадает с `$TEMP_OBJECTS_AS`). Если определено значение `table`, то способ создания таблицы (`create+insert` или `create table as select`) регулируется переменной `$FINAL_OBJECTS_AS`
- `to_name` - имя объекта, который создается в БД. Если установлено {auto}, то генерируется на основе имени создаваемого датасета.
- `to_schema` - имя схемы, в которой создастся объект в БД. Если установлено "{auto}", то будет использовано значение `$DATASET_SCHEMA_NM`
- `drop_at_end` - если установлено true, то итоговый объект будет удален после завершения работы задания. По умолчанию `false`
- `aux_to` - тип промежуточных объектов. Возможны `"table", "view", "cte", "subquery", "{auto}"`. Если установлено "{auto}", то будет использовано значение `$TEMP_OBJECTS_AS`
- `aux_drop_at_end` - если установлено false, то промежуточные объекты не будут удаляться после завершения работы задания. По умолчанию `true`
- `skip_agg_check` - пропустить проверку необходимости агрегации для предагрегатов и переменных, полученных по связи :M. Датасет, созданный со значением параметра `true` не может быть использован в других узлах. По умолчанию `false`.

fat_impute

Формат реализует функциональность импутации датасета с возможностью задать правила замены пропущенных значений.

Результатом работы узла является табличная структура, следующего формата:

Поле(-я)	Тип	Комментарий
Ключ сущности	`...`	Ключи из сущности датасета из `dataset`
`report_dttm`	`datetime`	Бизнес дата данных, если присутствует в исходном датасете
Фичи датасета	`...`	Все поля исходного датасета
`_run_rk`	`int`	Технический идентификатор запуска
`_created_dttm`	`datetime	Техническая дата вставки

Схема:

```
{
   *"dataset": str,               # Ссылка на исходный датасет в формате {dataset_rk}
                                  #   или {task_id} узла
    "name": str,                  # Название нового датасета
    "desc": str,                  # Описание датасетов, полученных в результате стратификации
    "rules": [%fat_impute_rule]   # Правила импутации
}
```

Схема правил импутации пропущенных значений `%fat_impute_rule`:

```
{
    *"alias": list[str], # переменные датасета
    *"function": "mean" | "median" | "mode" | "max" | "min" | "const",  # функция импутации (enum)
    "group": list[str],  # поля группировки
    "value": str         # значение, заменяющее NULL
}
```

РУКОВОДСТВО ПО ЗАПОЛНЕНИЮ ПОЛЕЙ СХЕМЫ ИМПУТАЦИИ ЗНАЧЕНИЙ

- `alias` - название фичи в исходном датасете.
- `function` - функция импутации.
- `group` - список полей датасета для группировки значений перед применением функции `function`. Если не задано поле `group`, то функция применяется по всем строкам датасета.
- `value` - значение, подставляемое при отсутствии значения, полученного в результате работы функции. Поле является обязательным для функции `const`, где все пропущенные значения будут заполнены введенным значением.

ПРИМЕР

Правило импутации:

```
{
    "alias": ["customer_age"],
    "function": "min",
    "group": ["gender", "marital_status", "report_dttm"],
    "value": 36
}
```

Алгоритм SQL:

```
with calc_impute_1 as (
    select
        "gender",
        "marital_status",
        "report_dttm",
        min("customer_age") as "customer_age"
    from dataset.OLD_DATASET
    group by
        "gender",
        "marital_status",
        "report_dttm"
```

```
)

select
    old."SOME_KEY_1",
    old."report_dttm",
    cast(
        coalesce(
            old."customer_age", calc_impute_1."customer_age", 36
        ) as <data_type for customer_age>
    ) as "customer_age"
    ...
from dataset.OLD_DATASET old
left join calc_impute_1
    on 1=1
    and calc_impute_1."gender" = old."gender"
    and calc_impute_1."marital_status" = old."marital_status"
    and calc_impute_1."report_dttm" = old."report_dttm"
```

fat_map

Формат реализует функциональность перехода с одной сущности на другую с возможностью отфильтровать записи и сохранить переменные.

Результатом работы узла является табличная структура, следующего формата:

Поле(-я)	Тип	Комментарий
Ключ сущности	`...`	Ключи из `entity_rk`
`report_dttm`	`datetime`	Бизнес дата данных, если присутствует в исходном датасете
Фичи датасета	`...`	Все что перечислено в `columns`
`_run_rk`	`int`	Технический идентификатор запуска
`_created_dttm`	`datetime	Техническая дата вставки

Схема:

```
{
   *"dataset": str,          # идентификатор датасета или узла конструктора
                             # {dataset_rk} или {constructor_task_id}
   *"name": str,             # название нового датасета
    "description": str,      # описание нового датасета
   *"chain": [int, ...],     # цепочка по которой выполняется переход от dataset к entity_rk
   *"entity_rk": int,        # целевое зерно, к которому выполняется переход
    "columns": '{all}' | [str, ...] ,  # названия колонок
    "priority": [str, ...]   # правила сортировки для row_number()
                             # анализируется только если используется связь "M:1/N"
                             # и заполнен columns, иначе игнорируется; заполняется так же как
                             # в split
    "filter": str | <%tree>  # выражение фильтрации, выполняемой при осуществлении перехода
}
```

РУКОВОДСТВО ПО ЗАПОЛНЕНИЮ ПОЛЕЙ СХЕМЫ

- `dataset` - идентификатор исходного датасета в формате `{id}`, где id - суррогатный ключ существующего датасета (`dataset_rk`) или task_id узла;
- `name` - название нового датасета, должно быть уникально во всем каталоге датасетов;
- `description` - описание нового датасета;
- `chain` - цепочка связей перехода, которая должна удовлетворять следующим условиям:
- первая связь цепочки содержит сущность исходного датасета;
- последняя связь цепочки содержит сущность нового датасета, указанную в поле `entity_rk`;
- у связей в цепочке есть общая сущность с каждым соседом;
- `entity_rk` - суррогатный ключ сущности нового датасета, к которой осуществляется переход от сущности исходного датасета, через цепочку связей `chain` (целевая сущность перехода);
- `columns` - список переменных исходного датасета, которые будут включены в новый датасет. Если передано `{all}`, то в новый датасет включаются все переменные из исходного датасета;
- `priority` - правила сортировки данных итогового датасета в формате `<поле датасета> asc|desc nulls last|first` или `random()`, если необходима случайная сортировка;
- `filter` - выражение фильтрации записей при осуществлении перехода. Допустимо использовать:
- Готовые переменные под сущностью исходного датасета без цепочки - `{1}`, `{3}`, etc.
- Готовые переменные, полученные с использованием цепочки - `[2]{3}`, `[1,2]{2}`, etc.
- Ключи сущности из цепочки - `[2]<1>`, `[1,2]<2>`, etc.
- Переменные исходного датасета - `{DATASET_FEATURE_1}`, `{DATASET_FEATURE_2}`, etc.
- Поля с ключами сущности исходного датасета - `{ENTITY_KEY_1}`, `{ENTITY_KEY_2}`, etc.
- Поле с датой расчета исходного датасета - `{report_dttm}`

NB! В фильтре в качестве цепочки допустимо использовать только подцепочки из цепочки связей перехода. Например, если переход осуществляется по цепочке `[1,2,3,4]`, то возможно использовать готовые переменные или ключи сущности с одной из следующих цепочек:

- `[1]`
- `[1,2]`
- `[1,2,3]`
- `[1,2,3,4]`

Алгоритм:

- Если в цепочке не существует связей "M:1/N", то атрибуты просто копируются в датасет под новой сущностью с предварительной фильтрацией данных, если передан фильтр
- Если связь "M:1/N" существует, то на результирующую выборку накладывается условие дедубликации записей по ключам целевой сущности: row_number()=1 с сортировкой по полям, указанным в priority (при этом в конец полей сортировки всегда добавляются ключи зерна и дата из датасета для обеспечения детерминированности). Фильтрация данных в этом случае производится до дедубликации, если передан фильтр.

Если последняя таблица перехода не содержит всех ключей целевой сущности, то в качестве последней таблицы связи добавляется опорная таблица. Опорная таблица определяется по приоритетам:

1. Реестр сущности по загрузчику из опции сущности `fat_registry`
2. Объединение ключей из всех таблиц с готовыми переменными, которые зарегистрированы под целевой сущностью.

NB!: Для всех таблиц перехода соблюдается правило - отбираются записи (связи), период актуальности которых включает в себя расчетную дату датасета-источника.

Узел поддерживает подстановку переменных в:

- dataset (ссылка на узел constructor или суррогатный ключ датасета)
- columns (только {all} - все переменные датасета)

Допустимые *options*

- `to` - тип объекта, который создается в БД. Возможны *"table", "view", "cte", "subquery", "{auto}"*. Если установлено "{auto}", то будет использовано значение default_to из job_options (по умолчанию совпадает с `$TEMP_OBJECTS_AS`). Если определено значение `table`, то способ создания таблицы (`create+insert` или `create table as select`) регулируется переменной `$FINAL_OBJECTS_AS`
- `to_name` - имя объекта, который создается в БД. Если установлено {auto}, то генерируется на основе имени создаваемого датасета.
- `to_schema` - имя схемы, в которой создастся объект в БД. Если установлено "{auto}", то будет ипользовано значение *$DATASET_SCHEMA_NM*
- `drop_at_end` - если установлено true, то итоговый объект будет удален после завершения работы задания. По умолчанию *false*
- `aux_to` - тип промежуточных объектов. Возможны *"table", "view", "cte", "subquery", "{auto}"*. Если установлено "{auto}", то будет использовано значение *$TEMP_OBJECTS_AS*
- `aux_drop_at_end` - если установлено false, то промежуточные объекты не будут удаляться после завершения работы задания. По умолчанию *true*

fat_merge

Формат реализует функциональность объединения/пересечения/исключения датасетов с возможностью сохранить переменные.

Результатом работы узла является табличная структура, следующего формата:

Поле(-я)	Тип	Комментарий
Ключ сущности	`...`	Ключи из сущности датасетов в `code`. Должна совпадать у всех
`report_dttm`	`datetime`	Бизнес дата данных, если присутствует во всех датасетах
Фичи датасета	`...`	Все что перечислено в `columns`
`_run_rk`	`int`	Технический идентификатор запуска
`_created_dttm`	`datetime	Техническая дата вставки

Схема:

```
{
   *"code": str,                      # формула в текстовом формате с указанием ссылок на датасеты
                                      # или узлы конструктора в виде {dataset_rk} или {constructor_task_id}
   *"name": str,                      # название нового датасета
    "description": str,               # описание нового датасета
    "columns": '{auto}' | '{all}' | [str | %fat_merge_column]  # поля для включения в новый датасет
}
```

Схема `%fat_merge_column`

```
{
   *"feature": str,  # переменная из исходного датасета
   *"alias": str,    # алиас переменной в новом датасете
    "dataset": str   # исходный датасет из формулы "code" из которого берется переменная
}
```

РУКОВОДСТВО ПО ЗАПОЛНЕНИЮ ПОЛЕЙ СХЕМЫ

- `code` - формула для построения нового датасета из существующих с использованием функций `UNION`, `INTERSECT`, `EXCEPT`. В качестве идентификатора датасета в формулах используется `{id}`, где id - суррогатный ключ существующего датасета (`dataset_rk`) или task_id узла;
- `name` - название нового датасета, должно быть уникально во всем каталоге датасетов;
- `description` - описание нового датасета;
- `columns` - список переменных из используемых датасетов, которые будут включены в новый датасет. Допустимо использовать:
- `'{all}'` - добавить все переменные из всех датасетов из формулы `code`
- `'{auto}'` - добавить только те переменные, которые есть в каждом датасете из формулы `code`
- `['field1', 'field2', %fat_merge_column]` - добавить конкретные переменные

Указание датасета и алиаса возможно только при указании переменной через схему `%fat_merge_column`, в противном случае считается, что датасет и алиас не указаны.

Если для переменной указан алиас, то имя итоговой переменной берется из него. Иначе имя итоговой переменной совпадает с именем исходной переменной.

Если для исходной переменной задана ссылка на датасет, то такая переменная отбирается только из заданного датасета. Иначе исходная переменная отбирается из всех датасетов, где она присутствует.

Если для одной итоговой переменной определено несколько исходных переменных, то такая переменная рассчитывается через `coalesce` с перечислением исходных переменных внутри. При этом порядок исходных переменных в `coalesce` определяется порядком указания датасетов в формуле `code`.

NB! Для расчета одной итоговой переменной из одного датасета может использоваться только одна исходная переменная.

Алгоритм:

Переменные отбираются через coalesce из соответствующих датасетов в порядке их указания в формуле. При сборке приоритетов не учитываются исключающие датасеты (второй и последующие аргументы функции except).

Узел поддерживает подстановку переменных в:

- code (ссылка на узел constructor или суррогатный ключ датасета)
- columns (auto - автоопределение и выбор всех допустимых переменных)

Допустимые *options*

- `to` - тип объекта, который создается в БД. Возможны *"table", "view", "cte", "subquery", "{auto}"*. Если установлено "{auto}", то будет использовано значение default_to из job_options (по умолчанию совпадает с `$TEMP_OBJECTS_AS`). Если определено значение `table`, то способ создания таблицы (`create+insert` или `create table as select`) регулируется переменной `$FINAL_OBJECTS_AS`
- `to_name` - имя объекта, который создается в БД. Если установлено {auto}, то генерируется на основе имени создаваемого датасета.
- `to_schema` - имя схемы, в которой создастся объект в БД. Если установлено "{auto}", то будет ипользовано значение *$DATASET_SCHEMA_NM*
- `drop_at_end` - если установлено true, то итоговый объект будет удален после завершения работы задания. По умолчанию *false*
- `aux_to` - тип промежуточных объектов. Возможны *"table", "view", "cte", "subquery", "{auto}"*. Если установлено "{auto}", то будет использовано значение *$TEMP_OBJECTS_AS*
- `aux_drop_at_end` - если установлено false, то промежуточные объекты не будут удаляться после завершения работы задания. По умолчанию *true*

fat_resample

Формат реализует функциональность создания нового датасета с определенными строками из существующего датасета с обеспечением возможности ручной замены значений сущности и/или переменных для копируемых строк

Результатом работы узла является табличная структура, следующего формата:

Поле(-я)	Тип	Комментарий
Ключ сущности	`...`	Ключи сущности исходного датасета
`report_dttm`	`datetime`	Дата расчета: now()
Фичи датасета	`...`	Переменные исходного датасета
`_run_rk`	`int`	Технический идентификатор запуска
`_created_dttm`	`datetime	Техническая дата вставки

Схема:

```
{
   *"dataset": str,          # {dataset_rk} или ссылка на {constructor_task_id}
   *"name": str,             # имя датасета
    "description": str,      # описание датасета
    "by_entities": {         # строки для копирования
       *"data": [
            {
               *"src_entity": {str: LispTree},     # значения ключей сущности в исходном датасете
                                                   # в формате {"<key_rk>": value}
                "entity": {str: LispTree},         # значения ключей сущности в целевом датасете
                                                   # в формате {"<key_rk>": value}
                "columns": {str: LispTree | None}  # данные для замены переменных датасета
                                                   # в формате {"<db_column>": value}
            }
        ]
    },
    "by_rows": {              # строки для копирования
        "sort": list[str],    # правила сортировки в формате "<db_column> asc|desc nulls first|last"
       *"data": [
            {
               *"row_number": int,                 # номер строки
                "entity": {str: LispTree},         # значения ключей сущности в целевом датасете
                                                   # в формате {"<key_rk>": value}
                "columns": {str: LispTree | None}  # данные для замены переменных датасета
                                                   # в формате {"<db_column>": value}
            }
        ]
    }
}
```

РУКОВОДСТВО ПО ЗАПОЛНЕНИЮ ПОЛЕЙ СХЕМЫ

- `dataset` - Идентификатор исходного датасета в формате `{id}`, где id - суррогатный ключ существующего датасета (`dataset_rk`) или task_id узла `constructor`
- `name` - Название нового датасета, должно быть уникально во всем каталоге датасетов
- `description` - Описание нового датасета
- `by_entities` - Строки для копирования, отбираемые по значению ключей сущности
- `src_entity` - Исходные значения ключей сущности для копируемой строки
- `entity` - Целевые значения ключей сущности для копируемой строки (если необходимо заменить исходные). Т.к. целевые значения сущности должны быть уникальны среди всех копируемых строк, то в секции `by_rows` поле `entity` не может быть заполнено только для части копируемых строк (или по всем указано или по всем отсутствует). К секции `by_entities` такое ограничение не применяется
- `columns` - Целевые значения **переменных** датасета для строки (если необходимо заменить исходные). Тип данных целевого значения должен быть совместим с типом данных переменной в исходном датасете
- `by_rows` - Строки для копирования, отбираемые по номеру строки с учетом сортировки (за исключением строк, указанных в `by_entities`)

- `sort` - Правила сортировки исходного датасета для нумерации строк. Допустимо использовать переменные датасета и ключи сущности
- `row_number` - Номер копируемой строки после применения сортировки

АЛГОРИТМ

1. Создается таблица целевого датасета (копия структуры таблицы исходного датасета)
2. Выполняется копирование данных из исходного датасета для каждой строки из `by_entities`

- Отбирается строка со значением сущности из `src_entity`
- Если передано поле `entity` - подставляются указанные ключи сущности
- Если передано поле `columns` - подставляются значения для указанных переменных
- В качестве даты подставляется `now()`
- Для всех остальных колонок исходного датасета сохраняются оригинальные значения
- Полученная строка вставляется в целевой датасет

3. Выполняется копирование данных из исходного датасета для каждой строки из `by_rows`

- Фильтруется исходный датасет - убираются строки со значениями `coalesce(entity, src_entity)` из `by_entities`
- Полученная таблица сортируется по правилам из `sort` и нумеруются полученные строки. Если правила сортировки не указаны, то используется сортировка по умолчанию по ключам сущности (`asc nulls last`). Сортировка по умолчанию всегда добавляется в конец полей сортировки для обеспечения детерминированной выборки.
- Отбирается строка под номером из `row_number`
- Выполняется копирование строки в целевой датасет с заменой значений по аналогии с `by_entities`

Допустимые *options*

- `to` - тип объекта, который создается в БД. Возможны *"table", "view", "cte", "subquery", "{auto}"*. Если установлено "{auto}", то будет использовано значение default_to из job_options (по умолчанию совпадает с `$TEMP_OBJECTS_AS`). Если определено значение `table`, то способ создания таблицы (`create+insert` или `create table as select`) регулируется переменной `$FINAL_OBJECTS_AS`
- `to_name` - имя объекта, который создается в БД. Если установлено {auto}, то генерируется на основе имени создаваемого датасета.
- `to_schema` - имя схемы, в которой создастся объект в БД. Если установлено "{auto}", то будет ипользовано значение *$DATASET_SCHEMA_NM*
- `drop_at_end` - если установлено true, то итоговый объект будет удален после завершения работы задания. По умолчанию *false*
- `aux_to` - тип промежуточных объектов. Возможны *"table", "view", "cte", "subquery", "{auto}"*. Если установлено "{auto}", то будет использовано значение *$TEMP_OBJECTS_AS*
- `aux_drop_at_end` - если установлено false, то промежуточные объекты не будут удаляться после завершения работы задания. По умолчанию *true*

fat_split

Формат реализует функциональность стратификации датасетов с возможностью задать правила группировки и сортировки, применяемые при нумерации строк исходного датасета для дальнейшей нарезки. В результате работы метода формируются новые датасеты, согласно правилам нарезки.

Результатом работы узла является табличная структура, следующего формата:

Поле(-я)	Тип	Комментарий
Ключ сущности	`...`	Ключи из сущности датасета из `dataset`
`report_dttm`	`datetime`	Бизнес дата данных, если присутствует в исходном датасете
Фичи датасета	`...`	Все поля исходного датасета
`_run_rk`	`int`	Технический идентификатор запуска
`_created_dttm`	`datetime	Техническая дата вставки

Схема:

```
{
   *"dataset": str,             # Ссылка на исходный датасет в формате {dataset_rk}
                                #   или {task_id} узла
   *"code": [int | float, str]  # Инструкции для стратификации e.g. [1, 'part_1', 200, 'part_2']
                                #   или [0.5, 'part 1', -1, 'part_2']
   *"by": 'rows' | 'share',     # Тип разделения - строки (rows) или доли (share);
    "name": str,                # Общая часть в названии всех датасетов, полученных в
                                #   результате стратификации
    "description": str,         # Описание датасетов, полученных в результате стратификации
    "group": [str],             # Название переменных для группировки
    "sort": [str],              # Название переменных для сортировки в формате
                                #   ["dataset_field asc|desc nulls last|first" | "random()"]
}
```

РУКОВОДСТВО ПО ЗАПОЛНЕНИЮ ПОЛЕЙ СХЕМЫ

- `dataset` - ссылка на стратифицируемый датасет.
- `code` - список правил нарезки исходного датасета в формате `<размер выборки>, <название выборки>`. Должна быть задана хотя бы одна выборка. Размер выборки может быть задан как в виде количества строк, так и в виде доли от общего числа строк в исходной таблице (от 0 до 1). Последним значением выборки может выступать -1, что обозначает "все оставшиеся строки". При использовании нарезки по долям, сумма размеров всех долей выборок должна быть <= 1. Минимальный размер выборки = 0.
- `by` - если размер выборки в коде задан количеством строк, то "rows", иначе "share".
- `name` - общая часть в названии всех создаваемых датасетов. Правила формирования названия создаваемого датасета:
- Если `name` передан и название выборки != "", то = `name` _\<название выборки>;
- Если `name` не передан и название выборки != "", то = <название выборки>;
- Если `name` передан и название выборки == "", то = `name`;
- Если не передан "name" и название выборки == "", то ошибка.
- `description` - описание создаваемых датасетов (общее).
- `group` - список полей группировки, используемый при нумерации строк исходного датасета. Могут быть указаны любые поля датасета, кроме технических.
- `sort` - список полей сортировки, используемый при нумерации строк исходного датасета, в формате `<поле датасета> asc|desc nulls last|first` или `random()`, если необходима случайная сортировка. Если не передано, то сортировка не используется, порядок строк при нарезке не гарантируется.

Нарезка датасета осуществляется по следующему алгоритму:

1. Все строки исходного датасета нумеруются с помощью функции row_number(), где в partition by указываются поля из `group`, а в order by - из `sort`. Если `sort` передан, то в конец полей сортировки всегда добавляются ключи сущности и отчетная дата для обеспечения детерминированной сортировки с указанием asc nulls last. Если `sort` не передан, то сортировка не добавляется, порядок записей при нарезке не гарантируется.
2. Производится нарезка датасета по правилам `row_num > левая_граница_выборки and row_num <= правая_граница`. Правая граница = левая граница выборки + размер выборки. Левая граница первой выборки = 0. Левая граница каждой последующей выборки = правая граница предыдущей выборки.
3. Для каждой выборки формируется новый датасет.

   Созданные датасеты доступны для обращения в последующих узлах графа по переменной `task_id`_<название выборки>.

   Узел поддерживает подстановку переменных в:

   - `dataset` (ссылка на узел constructor или суррогатный ключ датасета)

   Допустимые *options*

   - `to` - тип объекта, который создается в БД. Возможны *"table", "view", "cte", "subquery", "{auto}"*. Если установлено "{auto}", то будет использовано значение default_to из job_options (по умолчанию совпадает с `$TEMP_OBJECTS_AS`). Если определено значение `table`, то способ создания таблицы (`create+insert` или `create table as select`) регулируется переменной `$FINAL_OBJECTS_AS`
   - `to_name` - имя объекта, который создается в БД. Если установлено {auto}, то генерируется на основе имени создаваемого датасета датасета для выборки. Если заполнено и != {auto}, то используется в качестве префикса - итоговое имя объекта в таком случае определяется как конкатенация to_name + "_" + <название выборки>
   - `to_schema` - имя схемы, в которой создастся объект в БД. Если установлено "{auto}", то будет использовано значение *$DATASET_SCHEMA_NM*
   - `drop_at_end` - если установлено true, то итоговый объект будет удален после завершения работы задания. По умолчанию *false*
   - `aux_to` - тип промежуточных объектов. Возможны *"table", "view", "cte", "subquery", "{auto}"*. Если установлено "{auto}", то будет использовано значение *$TEMP_OBJECTS_AS*
   - `aux_drop_at_end` - если установлено false, то промежуточные объекты не будут удаляться после завершения работы задания. По умолчанию *true*

---

fat_stats

Формат реализует функциональность расчета статистик по датасету, при этом позволяет рассчитывать как базовые статистики, так и статистики в разрезе переменных.

Если запрошен расчет базовых статистик, то после завершения расчета происходит замена всех значений текущих базовых статистик рассчитанными показателями (все текущие базовые статистики удаляются и вместо них добавляются новые значения).

Если переданы статистики по переменным, то при запуске расчета производится замена текущего маппинга статистик на переданные показатели. Также после завершения расчета происходит замена текущих статистик по переменным рассчитанными показателями (все текущие статистики по всем переменным удаляются и вместо них добавляются новые значения).

Схема:

```
{
    *"dataset": str,           # ссылка на датасет {1} или узел {constructor_task_id}
     "stats": list[str]        # список базовых статистик
     "feature_stats": dict[str, list[str]]  # статистики по переменным
}
```

РУКОВОДСТВО ПО ЗАПОЛНЕНИЮ ПОЛЕЙ СХЕМЫ

- `dataset` - ссылка на датасет или узел constructor, по которому нужно выполнить расчет статистик
- `stats` - базовые статистические показатели для расчета
- `feature_stats` - статистики по переменным в формате {"переменная датасета": ["список статистик"]}

В задании на расчет статистик должно быть передано поле `stats` и/или `feature_stats` с хотя бы одной статистикой.

Перечень допустимых статистик, их классификацию и описание см. тут.

Статистики по переменным должны быть совместимы с типом данных переменных по которым они считаются. Совместимость определяется по справочнику "индикаторы для типов данных".

Алгоритм:

1. Если переданы статистики по перменным, то выполняется обновление маппинга статистик в метаданных - текущий маппинг заменяется показателями из `feature_stats`
2. Выполняется расчет статистик
3. Выполняется обновление статистик в метаданных

fat_union_all

Формат реализует функциональность объединения датасетов функцией `UNION ALL` с возможностью сохранить текущие переменные и рассчитать новые. Итоговый датасет будет зарегистрирован под сущностью исходных датасетов.

Результатом работы узла является табличная структура, следующего формата:

Поле(-я)	Тип	Комментарий
Ключ сущности	`...`	Ключи из сущности датасетов в `datasets`
`report_dttm`	`datetime`	Бизнес дата данных (`default=datetime.now()`)
Фичи датасета	`...`	Все, что перечислено в `columns`
`_run_rk`	`int`	Технический идентификатор запуска
`_created_dttm`	`datetime	Техническая дата вставки

Схема:

```
{
   *"datasets": list[str],            # список ссылок на датасеты и/или узлы конструктора
                                      # в виде {dataset_rk} или {constructor_task_id}
   *"name": str,                      # название нового датасета
    "description": str,               # описание нового датасета
    "columns": "{auto}" | "{all}" | [%fat_union_column]  # поля для включения в новый датасет
                                                         # значение по умолчанию {auto}
}
```

Схема `%fat_union_column`

```
{
   *"code": LispTree,  # код расчета переменной
   *"alias": str,      # алиас переменной в новом датасете
    "dataset": str     # исходный датасет из формулы "code" из которого берется переменная
}
```

РУКОВОДСТВО ПО ЗАПОЛНЕНИЮ ПОЛЕЙ СХЕМЫ

- `datasets` - Список датасетов для объединения. В качестве идентификатора датасета в формулах используется `{id}`, где id - суррогатный ключ существующего датасета (`dataset_rk`) или `task_id` узла. На объединяемые датасеты накладываются следующие ограничения:
- По всем датасетам должна совпадать сущность
- Если в датасете отсутствует `report_dttm`, при объединении этот атрибут будет заполнен текущей датой функцией `now()`. Правило распространяется на все используемые в поле `datasets` датасеты
- `name` - Название нового датасета, должно быть уникально во всем каталоге датасетов
- `description` - Описание нового датасета
- `columns` - Список переменных, которые будут включены в новый датасет. Допустимо использовать:
- `{all}` - Все переменные из всех датасетов из списка `datasets`. **NB!** Для переменных, присутствующих только в части датасетов, из остальных датасетов отбирается `null`!
- `{auto}` - *(значение по умолчанию)* Только те переменные, которые есть в каждом датасете из списка `datasets`
- `list[%fat_union_column]` - Конкретные переменные / рассчитать новые переменные

Схема `%fat_union_column`

Если поле `dataset` не заполнено, то переменная `alias` рассчитывается по формуле `code` для всех датасетов, где присутствуют все необходимые поля для расчета формулы.

Если поле `dataset` заполнено, то переменная `alias` рассчитывается только для указанного датасета.

NB! Если переменная `alias` определена только для части датасетов, то из остальных датасетов для неё отбирается `null`.

Пример

body:

```
{
  "datasets": ["{1}", "{2}"],
  "name": "new_dataset",
  "columns": [
    {
      "code": "777",
      "alias": "new_feature",
      "dataset": "{1}"
    }
  ]
}
```

Псевдо-sql-скрипт:

```
select
  customer_rk as customer_rk,
  report_dttm as report_dttm,
  777 as new_feature
from store.customer_table_1
union all
select
  customer_rk as customer_rk,
  now() as report_dttm,
  null as new_feature
from store.customer_table_2
```

Допустимые *options*

- `to` - тип объекта, который создается в БД. Возможны *"table", "view", "cte", "subquery", "{auto}"*. Если установлено "{auto}", то будет использовано значение default_to из job_options (по умолчанию совпадает с `$TEMP_OBJECTS_AS`). Если определено значение `table`, то способ создания таблицы (`create+insert` или `create table as select`) регулируется переменной `$FINAL_OBJECTS_AS`
- `to_name` - имя объекта, который создается в БД. Если установлено {auto}, то генерируется на основе имени создаваемого датасета.
- `to_schema` - имя схемы, в которой создастся объект в БД. Если установлено "{auto}", то будет ипользовано значение *$DATASET_SCHEMA_NM*
- `drop_at_end` - если установлено true, то итоговый объект будет удален после завершения работы задания. По умолчанию *false*
- `aux_to` - тип промежуточных объектов. Возможны *"table", "view", "cte", "subquery", "{auto}"*. Если установлено "{auto}", то будет использовано значение *$TEMP_OBJECTS_AS*
- `aux_drop_at_end` - если установлено false, то промежуточные объекты не будут удаляться после завершения работы задания. По умолчанию *true*

ingest

Задание реализует создание нового датасета с загрузкой данных.

Результатом работы узла является табличная структура, следующего формата:

Поле(-я)	Тип	Комментарий
Ключи сущности	`...`	Ключи из `entity_rk` датасета
Фичи датасета	`...`	Все что перечислено в `features`
Дата начала записи	`datetime`	Бизнес дата данных (опционально)
Дата окончания записи	`datetime`	Бизнес дата данных (опционально)
`_run_rk`	`int`	Технический идентификатор запуска
`_created_dttm`	`datetime`	Техническая дата вставки

ОБЩАЯ ЛОГИКА РАБОТЫ

В зависимости от `params.format` структура `body` различается, но глобально его схема состоит из двух секций:

- Атрибуты, относящиеся к источнику данных. Это специфичные для формата атрибуты
- Атрибуты, описывающие создаваемую таблицу-приемник. Эта часть у всех форматов одинаковая

Описание приёмника

```
{
    # Информация о приемнике
   *"history": "replace" | "insert" | "snapshot"  # Режим загрузки данных
    "db_date": str
    "db_date_end": str
    "instances": [%instance]  # instance schema from dataset model
    "features": [%feature]    # feature schema from dataset model
}
```

Режимы загрузки:

- `replace` - без истории. Производится REPLACE таблицы при запуске
- `insert` - без истории. Производится INSERT новых срезов без проверок. Подходит для таблиц фактов, например транзакций
- `snapshot` - с историей. У записей у которых в датасете date_end=infinity проставляется `date_end=\<пришедшая date\> - 1 microsec`. Новые срезы вставляются с `date=\<пришедшая date\>`, `date_end=\<следующая пришедшая date -1 microsec\> or infinity`

Данные атрибуты напоминают схему %dataset неспроста. Они описывают структуру будущей таблицы, т.е. какие в ней будут сущности, переменные и даты актуальности и в каких столбцах они находятся. Логика описания структуры ровно та же самая, что и для датасета и повторно описываться здесь не будет. Важным нюансом является то, что атрибуты описывающие столбцы относятся к *исходным* данным, а атрибуты `name` к *целевым*.

Например, если переменная в секции `"features"` описана следующим образом:

```
{
    "name": "new_name",
    "db_column": "old_name",
    "fs_type": "INT"
}
```

то в результате работы узла `ingest` в схеме `store` появится таблица с именем колонки максимально приближенным к `new_name` (или совпадающим с ним), с типом `INT` и в ней будут данные из колонки `old_name` исходной таблицы. Если окажется что fs_type не совместим с данными в столбце `old_name`, то загрузка упадет.

Параметры создания

```
"params": {
    "format": "file" | "table" | "sql",
    "name": str,       # имя датасета
    "desc": str,       # описаниве датасета
    "db_schema": str,  # схема в БД, где создать таблицу, по умолчанию {auto}
    "db_table": str,   # имя таблицы, по умолчанию {auto}
    "origin": str      # метка о происхождении, по-умолчанию "ingest/<format>"
}
```

Если `params.name` не задано, то созданная таблица не будет сохранена как "датасет" в метаданных, но при этом по прежнему будет иметь структуру полноценного датасета **без технических полей**. Это может понадобиться если таблицу планируется использовать только как промежуточный шаг с данными для последующих узлов.

- `name` - Если задано имя датасета, то произойдет регистрация таблицы в метаданных в качестве "датасета" с указанным именем
- `desc` - Описание датасета, может быть пустым
- `db_schema` - Схема, в которой должна быть создана таблица датасета, по умолчанию это `DB__STORE_SCHEMA`.
- `db_table` - Имя с которым создать таблицу, по умолчанию это `name` в котором заменены все "опасные" символы на `"_"`
- `origin` - Метка о происхождении датасета. Значение перекладывается as-is в dataset.origin

> **Info**
>
> Если `name` не задан, то `db_schema` и `db_table` будут проигнорированы. Управлять именем объекта в БД можно только для датасетов, сохраняемых в метаданных. В режиме "промежуточного узла" для таблицы будет сформировано уникальное непроизносимое имя в схеме `DB__WORK_SCHEMA`

ФОРМАТ `file`

Источником данных является файл в S3 хранилище

- %ingest_file

```
{
    # информация об источнике
    path: "s3://<file path",  # адрес файла в s3
    format: "csv" | "xlsx",   # формат файла
    encoding: str,         # кодировка файла
    delimiter: str,        # разделитель столбцов
    ...
    # информация о приемнике см в секции "Общая логика"
}
```

ФОРМАТ `table`

Источником является таблица в источнике исполнения задания

- %ingest_table

```
{
    # информация об источнике
    db_table: str,    # имя таблицы в БД
    db_schema: str,   # имя схемы в БД
    ...
    # информация о приемнике см в секции "Общая логика"
}
```

ФОРМАТ `sql`

Источником данных является результат выражения SELECT в источнике исполнения задания

- %ingest_sql

```
{
    # информация об источнике
    statement: str,  # исполняемая SELECT инструкция на языке SQL
    ...
    # информация о приемнике см в секции "Общая логика"
}
```

stats

Задание запускает расчет статистик над датасетом. Статистики бывают:

- По всей таблице датасета: `DATE_MIN`, `DATE_MAX`, `DATES_COUNT`
- В разрезе даты: `ROWS_COUNT`
- В разрезе экземпляра сущности и даты: `COUNT_DISTINCT`
- В разрезе переменной и даты: `DISTINCT`, `COUNT_DISTINCT`

Описание статистических показателей:

- `DATE_MIN` = минимальное значение даты в поле `db_date` датасета
- `DATE_MAX` = максимальное значение даты в поле `db_date` датасета
- `DATES_COUNT` = количество уникальных дат в поле `db_date`
- `ROWS_COUNT` = количество строк в срезе на дату
- `COUNT_DISTINCT` = количество уникальных значений в срезе на дату
- `DISTINCT` = список уникальных значений в срезе на дату и количество строк по ним

Статистика `DISTINCT` хранится в json в следующем формате:

```
{
   *"values": [  # список уникальных значений
       {
           "value": json,  # значение
           "count": int  # количество строк, в которых встретилось значение
       },
   ],
   *"limit": int,  # лимит на длину списка values, берется из APP__STAT_DISTINCT_LIMIT
   *"overflow": bool  # переполнение статистики (true, если количество уникальных значений больше limit)
}
```

Статистики в разрезе даты рассчитываются с учетом фильтра на дату:

- Если в датасете есть `db_date` и `db_date_end`: {дата} between db_date and db_date_end
- Если в датасете только `db_date`: {дата} = db_date
- Если в датасете нет полей `db_date` и `db_date_end`, то расчет статистик выполняется без условий на дату. Для статистик `DATE_MIN`, `DATE_MAX` и `DATES_COUNT` в таком случае записывается null.

Рассчитанные статистики обновляются в metastore в режиме `upsert` по бизнес-ключу (если статистика по бизнес-ключу уже существует, то её значение обновляется, иначе - добавляется новая статистика). Бизнес-ключ определяется в зависимости от типа статистики:

- По всей таблице датасета: Суррогатный ключ датасета, Показатель
- В разрезе даты: Суррогатный ключ датасета, Показатель, Дата
- В разрезе экземпляра сущности и даты: Суррогатный ключ экземпляра сущности, Показатель, Дата
- В разрезе переменной и даты: Суррогатный ключ переменной, Показатель, Дата

СХЕМА **body** УЗЛА РАСЧЕТА СТАТИСТИК

```
{
   *"dataset": int | str,  # Датасет, по которому необходимо рассчитать статистики
    "stats": list[enum],   # Статистики по всей таблице датасета
    "by_date": {           # Статистики, которые могут быть рассчитаны только на конкретные даты
        "dates": list[datetime],                # Даты расчета статистик
        "common_stats": list[enum],             # Статистики в разрезе даты
        "instance_stats": dict[str, list[enum]],  # Статистики в разрезе экземпляра сущности и даты
        "feature_stats": dict[str, list[enum]]    # Статистики в разрезе переменной и даты
    }
}
```

РУКОВОДСТВО ПО ЗАПОЛНЕНИЮ ПОЛЕЙ СХЕМЫ

- `dataset` - датасет по которому необходимо выполнить расчет статистик. Может быть задан в виде суррогатного ключа, имени датасета или ссылки на узел `ingest` в формате `{ingest_task_id}`. Указанный датасет не должен быть виртуальным (`virtual=False`).
- `stats` - список статистических показателей по всей таблице датасета, которые необходимо рассчитать
- `dates` - список дат, на которые необходимо рассчитать статистические показатели в разерзе даты. Если поле не передано, то статистики рассчитываются на дату запуска. Может принимать следующие значения:
- `YYY-MM-DD hh:mm:ss` = конкретная дата
- `{now}` = дата запуска (общая на весь джоб)
- `{infinity}` = +бесконечность
- `common_stats` - список статистических показателей в разрезе даты, которые необходимо рассчитать
- `instance_stats` - статистические показатели в разрезе экземпляра сущности датасета и даты, которые необходимо рассчитать. Указываются в формате `{"instance_name": list["<stat>"]}`
- `feature_stats` - статистические показатели в разрезе переменной датасета и даты, которые необходимо рассчитать. Указываются в формате `{"feature_name": list["<stat>"]}`

3.3.2 Automap

Разметить таблицу (POST /automap)

Уровень доступа	load
Request Body	`%automap_table`
Response Body	`%ingest_table`

Задача метода - облегчить регистрацию таблицы в системе в качесте датасета. Для этого метод сканирует переданную ему таблицу и размечается найденные в ней:

- столбцы как переменные
- ключи как экземпляры сущности
- столбцы, похожие на версионность как `db_date` и `db_date_end`

В качестве параметра запроса принимается `source_rk` - источник таблицы для сканирования (значение по умолчанию -1).

Метод возвращает JSON, который в большинстве случаев можно передавать as-is внутрь метода POST /dataset за исключением случаев, помеченных `???` ниже

1. Поиск полей версионности:
   - Столбец, который имеет имя `report_dttm` или `from_dttm` будет указан как `db_date`, если его типу в БД соответствует `%fs_type` с `py_type='datetime'`
   - Столбец, который имеет имя `to_dttm` будет указан как `db_date_end`, если его типу в БД соответствует `%fs_type` с `py_type='datetime'`
2. Поиск ключей:
   - По каждой сущности, переданной в `"entities"` производится поиск столбцов в таблице, которые соответствуют ключам этой сущности. Если в таблице не найден столбец, имя которого равно `key.name`, то для этого ключа вместо имени столбца будет указано `???`
   - `name` и `desc` для экземпляра заполняются как `name` и `desc` сущности
   - `unique` заполняется как false без фактической проверки уникальности значений
3. Разметка остальных столбцов:
   - Все остальные столбцы размечаются как переменные с `name` равным имени столбца. Имя переменной должно начинаться латинской буквы и содержать только латинские буквы, цифры и знак подчеркивания. Все неподходящие символы в имени столбца заменяются на `'_'`.
   - `desc` заполняется как `null`
   - `db_column` заполняется именем столбца as-is
   - `fs_type` заполняется соответствующим типом

Разметить файл (POST /automap/file)

Уровень доступа	load
Request Body	`%automap_file`
Response Body	`%ingest_file`

Метод работает по той же логике, что и `POST /automap` со следующими отличиями:

- сканирует не таблицу, а файл
- если не переданы `format`, `encoding` или `delimiter` пытается сделать их автоопределение через `GET /files/describe`

3.4 Jobs

3.4.1 Jobs

Задания на расчет.

Архитектура

Все преобразования данных внутри axiom происходят через создание задания %job и последующий его запуск %job_run. Само задание представляет собой стандартный RESTful resource, поддерживающий операции CRUD.

Ресурс "запуск цепочки" (%job_run) поддерживает только операции:

- **CREATE**: Запуск задания `(POST) /jobs/{jobs_rk}`
- **READ**: Получение статуса запуска `(GET) /jobs/{job_rk}/status` или `(GET) /jobs/{job_rk}/runs/{run_rk}`
- **UPDATE**: Остановка запущенного задания `(POST) /jobs/{job_rk}/stop`

Экземпляр запуска несет в себе двойную функцию: с одной стороны он позволяет получить информацию непосредственно о конкретном запуске, с другой стороны он выступает в роли агрегированного "статуса" для всего задания. Это видно в спецификации: `/jobs/{job_rk}/status` возвращает экземпляр %job_run, а именно последний запуск задания.

Структура задания

- %job
- %job_run
- %job_option

Задание (Job) представляет собой граф задач (Task), связанных между собой зависимостями. Задачи в графе далее по тексту будут также называться "**узлами**" - это полный синоним для "задачи".

Зависимости между узлами в задании можно указать одним из двух способов: либо через словарь `"graph": {}`, либо через вспомогательные узлы `par` и `seq`, которые объединяют вложенные в них узлы в блоки для параллельного или последовательного исполнения. Для описания задания можно использовать оба метода указания зависимостей одновременно, препроцессор цепочки сначала раскроет все `par` и `seq` и добавит недостающие связи в словарь `"graph": {...}`.

Переменные в узлах

Некоторые узлы поддерживают подстановку переменных в `"body": {...}` перед исполнением. Переменной является текст вида `"{variable_name}"`, где vaiable_name состоит из латинских букв, цифр или знака `'_'`. Значение переменной для подстановки ищется в следующих местах (поиск останавливается, если значение найдено):

- секция `"params"` внутри узла (%job_task), где найдена переменная
- секция `"params"` узла `par` или `seq`, в который вложен текущий узел. Если узлы вложены несколько раз, то поиск продолжается до самого внешнего уровня.
- секция `"params"` внутри `"job_options"`, определенная на уровне всего задания.

Каждый тип узла определяет для себя свои правила подстановки переменных, но есть общие рекомендации:

- переменные простых типов int, float, str, bool подставляются по тексту простой заменой
- В местах где ожидается одна таблица можно поставить табличную переменную. В этом случае обработчик узла подставит имя таблицы, либо вставит код ее расчета как подзапрос/cte в зависимости от опций.

Виды табличных переменных :

- `"{task_id}"` - где `task_id` - имя узла, на который надо сослаться.
- `"{prev_task}"` - псевдо-переменная которая содержит ссылку на предыдущий узел внутри `seq`

Типы узлов

`par` И `seq`

Вспомогательные узлы для описания графа зависимостей между другими узлами:

- `par` - параллельное выполнение узлов
- `seq` - последовательное выполнение узлов

```
{
   ~"id": str,                  # автогенерация внутри блоков par и seq, если не указан явно
   *"type": "par" | "seq",      # параллельное или последовательное исполение
   *"body": [<%job_task>, ...], # список задач
    "params": dict              # переменные
}
```

`log`

Пишет сообщение в стандартный логгер. Поддерживает подстановку переменных

```
{
   ~"id": str,    # автогенерация внутри блоков par и seq, если не указан явно
   *"type": "log",
   *"body": str,  # сообщение
    "params": {
         "level": str  # "DEBUG" | "INFO" | "WARNING" | "ERROR" | "CRITICAL" # уровень логирования
    }
}
```

`select`

Узел для трансформации данных внутри БД. Результатом узла которого всегда будет select конструкция. В зависимости от options эта select конструкция может быть либо выполнена сразу (to=table/view), либо предварительно скомпилирована для использования в последующих узлах (to=cte/subquery).

Результатом работы узла является select (Примеры)

Узел поддерживает подстановку переменных в: `expr`, `tab_expr`, `as`, `on`

```
{
   ~"id": str,                   # автогенерация внутри блоков par и seq, если не указан явно
   *"type": "select",
   *"body": {
        "FROM": [table, ...],     # Список таблиц в формате expr, [tab_expr, as, on]
        "SELECT": [columns, ...], # Список колонок в формате expr, [expr, as, on]
        "WHERE": [str, ...],      # Список условий
        "GROUP BY": [str, ...],   # Список выражений
        "HAVING": [str, ...],     # Список выражений
        "ORDER BY": [str, ...],   # Список выражений
        "OPTIONS": [str, ..]      # inner, left, distinct, unwrap
   },
  *"params": {
        "to": "table" | "view" | "cte" | "{auto}",  # по умолчанию {auto} = {deafault_to}
        "to_name": str | "{auto}",    # имя объекта в БД, # по умолчанию {auto} = {task_id}
        "to_schema": str | "{auto}",  # имя схемы в БД, # по умолчанию {auto} = {default_schema}
        "drop_at_end": bool,          # удалить ли объект после выполнения цепочки, по умолчанию False
        "key": [str, ...],            # business key объекта, используется для оптимизаций доступа
        "storage_key" [str, ...],     # storage key объекта, используется для оптимизаций хранения
        "analyze": bool,              # запускать ли сбор статистики после создания таблицы
        "create_index": bool          # запускать создание индекса по key после создания таблицы
    }
}
```

`read`

Чтение содержимого таблицы в БД и запись результата в переменную с именем таски.

Формат результата таски - список строк, где каждая строка это словарь `{"column name": "value"}` Узел не поддерживает подстановку переменных

```
{
   ~"id": str,                # автогенерация внутри блоков par и seq, если не указан явно
   *"type": "read",           # чтение таблицы
   *"body": {
       *"src": [<name>, <schema>],  # Имя объекта в БД
       *"columns": list[str],       # список столбцов в таблице
    },
    "params":{
        "order_by": [str, ...],  # списоке колонок для сортировки (asc, nulls_last)
        "limit": int             # количество строк в ответе
    }
}
```

`import`

Узел записывает в таблицу в БД содержимое файла на S3.

Поддерживает подстановку переменных в `params` и `path`

```
{
   ~"id": str,               # автогенерация внутри блоков par и seq, если не указан явно
   *"type": "import",          # чтение таблицы
   *"body": {
       *"path": "s3://<file name>",  # путь на s3
       *"trg": [<name>, <schema>] ,  # имя таблицы в БД
       *"columns": {                 # список стобцов в файле с типами
            "column name": "fs_type",
            "column name": "fs_type",
            ...
        }
    },
    "params": {
        "format": "csv" | "xlsx",         # file format
        "delimiter": <delimiter char>,    # default: ','
        "encoding": <encoding>,           # default: 'utf8'
        "io_chunksize": <size in bytes>,  # optional
        "key": [str, ...],            # business key объекта, используется для оптимизаций доступа
        "storage_key" [str, ...],     # storage key объекта, используется для оптимизаций хранения
        "drop_at_end": bool,          # удалить ли объект после выполнения цепочки, по умолчанию False
    }
}
```

`export`

Узел записывает в файл на S3 содержимое таблицы в БД.

Поддерживает подстановку переменных в `params` и `body`.

```
{
   ~"id": str,               # автогенерация внутри блоков par и seq, если не указан явно
   *"type": "export",        # чтение таблицы
   *"body": {
       *"src": [<name>, <schema>],   # имя таблицы в БД
       *"path": "s3://<file name>",  # путь к файлу на s3
    },
    "params": {
        "delimiter": <delimiter char>,        # default: ','
        "encoding": <encoding>,               # default: 'utf8'
        "io_chunksize": <size in bytes>,      # optional
    }
}
```

`create`

Узел создает пустую таблиц заданной структуры в БД

```
{
   ~"id": str,               # автогенерация внутри блоков par и seq, если не указан явно
   *"type": "create",        # чтение таблицы
   *"body": {
       *"trg": [<name>, <schema>],      # имя таблицы в БД
       *"columns": {                    # список стобцов в файле с типами
            "column name": "fs_type",   # имя колонки: тип данных
            "column name": "fs_type",
            ...
        }
    },
    "params": {
        "scan_db": "not_exists" | "names" | "types",  # Проверить таблицу в БД
        "key": [str, ...],            # business key объекта, используется для оптимизаций соединения
        "storage_key" [str, ...],     # storage key объекта, используется для оптимизаций хранения
        "partition_key" [str, ...],   # partition key объекта, используется для оптимизаций доступа
```

```
        "updatable": bool,            # должна ли таблица подеерживать операцию UPDATE
        "aux": bool,                  # является ли таблица временной
        "drop_at_end": bool,          # удалить ли объект после выполнения цепочки, по умолчанию False
    }
}
```

apply

Узел загружает данные в таблицу в одном из поддерживаемых режимов

```
{
   ~"id": str,               # автогенерация внутри блоков par и seq, если не указан явно
   *"type": "apply",          # чтение таблицы
   *"body": {
       *"src": [<name>, <schema>],      # имя исходной таблицы в БД
       *"trg": [<name>, <schema>],      # имя целевой таблицы в БД
       *"mode": "replace" | "insert" | "snapshot",  # Режим загрузки
       *"mapping": {                    # маппинг колонок
            "trg column": "src column",  # Колонка в целевой таблице: колонка в исходной
            "trg column": "src column",
            ...
        },
        "date": str,                    # имя колонки с датой актуальности
        "date_end": str                 # имя колонки с окончанием даты актуальности
    }
}
```

sql_raw

Узел запускает переданный sql-скрипт

```
{
   ~"id": str,          # автогенерация внутри блоков par и seq, если не указан явно
   *"type": "sql_raw",  # запуск sql-скрипта
   *"body": str         # sql-скрипт
}
```

insert

Узел выполняет вставку строк в таблицу в БД

```
{
   ~"id": str,         # автогенерация внутри блоков par и seq, если не указан явно
   *"type": "insert",  # вставка строк в таблицу в БД
   *"body": {
       *"data": {<column>: <value>},  # данные для вставки
       "trg": [<name>, <schema>]      # имя целевой таблицы в БД
   },
   "params": {
       "batch_size": int,   # количество строк в батче для одной операции вставки
       "executemany": bool  # оптимизация для некоторых СУБД: использовать функцию executemany() с
                            # передачей вставляемых строк как список кортежей
   }
}
```

3.4.2 Create Job

Создать задание (POST /jobs)

Уровень доступа	**admin**
Request Body	**%job**
Response Body	**%job**

Создается новое задание (без запуска):

- заполняются метаданные

Оригинальный граф сохраняется в поле "body". При последующем запуске задания выполняется компиляция графа. Скомпилированный граф отличается от оригинального следующим:

- все ссылки на столбцы и таблицы вида {rk} заменены на конкретные имена
- узлы par и seq раскрыты и id-шники всех вложенных задач сгенерированы

Таким образом в скомпилированной цепочке у каждого узла есть id и он представляет собой неделимую операцию.

Граф задания должен быть уникальным в разрезе существующих заданий.

Подробнее про форматы см. основную статью.

3.4.3 Delete Jobs

Удалить задание (DELETE /jobs)

Уровень доступа	**admin**
Request Body	list[int]
Response Body	**-**

Метод удаляет задания и все их запуски. Принимает на вход список `job_rk`, физически удаляет данные из базы.

Запрещается удалять задания на расписании.

3.4.4 Edit Job

Редактировать задание (PATCH /jobs/{job_rk})

Уровень доступа	**admin**
Request Body	**%job_patch**
Response Body	**%job**

Позволяет изменять следующие атрибуты задания:

- Описание задания
- Оригинальный граф задач
- Параметры цепочки
- Расписание запуска

При этом изменяются только переданные поля.

3.4.5 Execute Job

Запустить задание (POST /jobs/{job_rk})

Уровень доступа	**admin**
Request Body	**%job_run**
Response Body	**%job_run**

Создает job_run со статусом 'queued' и фиксирует параметры запуска.

При запуске задания есть возможность изменить параметры job_options и resource. Алгоритм замены параметров описан ниже.

В качестве параметра запроса принимается `source_rk` - источник для запуска задания (значение по умолчанию -1).

После расчета параметров они записываются в базу в таблицу job_run и запускается процесс `worker`, который будет исполнять граф, записанный в job.body.

ЗАПОЛНЕНИЕ RESOURCE

Приоритет заполнения атрибута resource (от высшего к низшему):

- Из запроса - request.body.resource
- Значение, заданное при Create Job - meta.job.resource

ЗАПОЛНЕНИЕ JOB_OPTIONS

Приоритет в заполнении job_options определяется *по-атрибутно*, т.е. каждую опцию запуска можно задать на более подходящем для нее уровне, а итоговый словарь опций расчитывается как комбинация этих уровней. Для уровня "профиль исполнения" предусмотрен профиль по-умолчанию 'default', если параметр не передан.

Приоритеты следующие (от высшего к низшему):

- Из запроса - request.body.job_options.{option_name}
- Из профиля исполнения - meta.ref_execution_profile.{profile_name}.job_options.{option_name}
- Значение, заданное при Create Job - meta.job.job_options.{options_name}
- Опции по-умолчанию, заданные через переменные окружения

Опции по-умолчанию

```
{
    "async_factor": $ENGINE__ASYNC_FACTOR,
    "execution_profile": "default",
    "params": {
        "default_to": $ENGINE__TEMP_OBJECTS_AS,
        "default_schema": $settings.DB__WORK_SCHEMA,
        "default_storage_key": $ENGINE__DEFAULT_STORAGE_KEY,
        "default_analyze": $settings.ENGINE__AUTO_ANALYZE,
        "default_create_index": $settings.ENGINE__AUTO_INDEX,
        "work_schema": $settings.ENGINE__WORK_SCHEMA
    },
    "advanced": {
        "postgresql": [str, ...],  # строка опции без ведущего SET, напр. "datestyle TO postgres, dmy"
        "greenplum": [str, ...]   # строка опции без ведущего SET
    }
}
```

Остановить выполнение задания (POST /jobs/{job_rk}/stop)

Уровень доступа	**admin**
Request Body	**-**
Response Body	**%job_run**

Останавливает выполнение задания.

Остановка выполняется только для запусков в статусе `queued` и `running`. По остановленному запуску проставляется статус `failed` с описанием ошибки из параметра `reason`. Если параметр `reason` не передан, то в качестве текста ошибки проставляется текст по умолчанию - "Cancelled by user".

3.4.6 Get Job Info

Получить список заданий (GET /jobs)

Уровень доступа	**admin**
Request Body	-
Response Body	%job_summary

Метод возвращает список цепочек (заданий) с учетом заданных фильтров. В качестве элемента списка используется схема %job_summary, в которой присутствуют атрибуты как самой цепочки, так и последнего её запуска.

Для фильтрации запроса доступны параметры:

- `desc` : mask
- `state` : str
- `resource` : mask
- `limit` : int
- `offset` : int

Параметры `limit` и `offset` отвечают за размер и сдвиг выборки соответственно. При значении `limit: null` возвращается дефолтное количество объектов, которое определяется параметром `APP__DEFAULT_ITEMS_LIMIT`. Максимальное значение `limit` определяется параметром `APP__MAX_ITEMS_LIMIT`.

Для параметров `desc` и `resource` доступна исключающая фильтрация при добавлении символа `^` в начало значения маски фильтрации.

Получить информацию по заданию (GET /jobs/{job_rk})

Уровень доступа	**transform**
Request Body	-
Response Body	%job

Возвращает полное описание задания с оригинальным графом (body).

Выгрузить скрипт задания (POST /jobs/{job_rk}/code)

Уровень доступа	**read**
Request Body	-
Response Body	str

Возвращает sql-скрипт задания с учётом диалекта СУБД источника. Источник определяется параметром `source_rk` (значение по умолчанию -1).

Получить информацию по запускам задания (GET /jobs/{job_rk}/runs)

Уровень доступа	**admin**
Request Body	-
Response Body	%job_run

Метод возвращает список запусков с учетом заданных фильтров.

Для фильтрации запроса доступны параметры:

- `job_rk` : int
- `state` : str
- `resource` : mask
- `limit` : int
- `offset` : int

Параметры `limit` и `offset` отвечают за размер и сдвиг выборки соответственно. При значении `limit: null` возвращается дефолтное количество объектов, которое определяется параметром `APP__DEFAULT_ITEMS_LIMIT`. Максимальное значение `limit` определяется параметром `APP__MAX_ITEMS_LIMIT`.

В качестве элемента списка используется схема %job_run.

Получить информацию по последнему запуску задания (GET /jobs/{job_rk}/status)

Уровень доступа	**admin**
Request Body	-
Response Body	**%job_run**

Возвращает последний job_run по заданию, статусом всего задания считается `статус` последнего запуска.

Получить информацию по конкретному запуску (GET /jobs/-/runs/{run_rk})

Уровень доступа	**admin**
Request Body	-
Response Body	**%job_run**

Возвращает job_run по конкретному запуску.

Job Run State

Возможные значения state:

Статус	Описание
new	нет запусков
queued	job_run создан, но еще не запускался
running	job_run запущен и исполняется
success	job_run отработал без ошибок
failed	job_run завершился с ошибкой
disposed	задание удалено/отсутствует

3.4.7 Job Schedules

Установить расписание (PUT /schedules)

Уровень доступа	**admin**
Request Body	%schedule
Response Body	%schedule

Метод устанавливает расписание для задания. Задание будет запускаться согласно расписанию из поля `schedule` на источнике `source_rk` с опциями из `job_options`.

Для одного задания может быть определено только одно расписание. Если на момент установки расписания для задания уже определено расписание, то оно заменяется на новое.

Установить расписание можно только для существующего задания с режимом `dispose` = `smart` или `manual`.

Снять задание с расписания (DELETE /schedules/{schedule_rk})

Уровень доступа	**admin**
Request Body	-
Response Body	-

Метод удаляет переданное расписание по ключу - связанное задание снимается с расписания. Массовое удаление расписаний недоступно.

Получить все расписания (GET /schedules)

Уровень доступа	**admin**
Request Body	-
Response Body	%schedule

Метод возвращает список расписаний заданий с учетом заданных фильтров.

Для фильтрации запроса доступны параметры:

- `schedule_rks` : list[int]
- `job_rk` : list[int]
- `source_rk` : list[int]
- `limit` : int
- `offset` : int

Параметры `limit` и `offset` отвечают за размер и сдвиг выборки соответственно. При значении `limit: null` возвращается дефолтное количество объектов, которое определяется параметром `APP__DEFAULT_ITEMS_LIMIT`. Максимальное значение `limit` определяется параметром `APP__MAX_ITEMS_LIMIT`.

Получить информацию по расписанию (GET /schedules/{schedule_rk})

Уровень доступа	**admin**
Request Body	**-**
Response Body	**%schedule**

Метод возвращает расписание задания по ключу расписания.

3.5 Entities

3.5.1 Entities

Логическая **сущность**, объединяющая ключи для связи данных внутри приложения. Может состоять из нескольких **ключей**.

Пример: CUSTOMER (CUSTOMER_ID), OFFER (CUSTOMER_ID + CAMPAIGN_ID) и т.д.

3.5.2 Create Entity

Создать сущность (POST /entities)

Уровень доступа	**load**
Request Body	**%entity**
Response Body	**%entity**

В каталог сущностей добавляется новая сущность с выбранным набором ключей. При создании сущности можно использовать только существующие ключи. Новая сущность должно иметь не менее одного ключа.

Разрешается создавать сущности с набором ключей в точности, как у уже существующей сущности.

Название сущности должно соответствовать маске \^[a-zA-Z][a-zA-Z0-9_]*$. Название сущности должно быть уникально в разрезе каталога сущностей.

Для сущности можно задать список тегов. Отношение сущностей к тегам = M:N - один тег может быть проставлен на нескольких сущностях. Каталог сущностей может быть отфильтрован по тегу/списку тегов.

Дополнительно для сущности можно указать следующий список применяемых опций - %entity_options:

- Ключ распределения (`storage_key`) - список `key_rk` сущности
- Реестр сущности (`fat_registry`) - ссылка на загрузчик

Ключ распределения отвечает за физическое распределение данных по сущности в узлах кластера. Является более приоритетным по отношению к атрибуту `default_storage_key` в опциях задания расчета или профиля исполнения. Опция применима не для всех СУБД.

Реестр сущности используется как сегмент по умолчанию в конструкторе датасетов %fat_compose и в узлах перехода %fat_map , если последняя (целевая) связь цепочки является автоматической и приводит к расширению состава ключей. Реестр сущности применяется только если в соответствующей опции указан загрузчик готовых переменных с той же сущностью, в противном случае опция будет проигнорирована.

3.5.3 Delete Entities

Удалить сущности (DELETE /entities)

Уровень доступа	**load**
Request Body	list[int]
Response Body	**-**

Метод удаляет указанные сущности. Ключи и тэги, использованные в сущности, **не удаляются**.

3.5.4 Edit Entity

Редактировать сущность (PATCH /entities/{entity_rk})

Уровень доступа	**load**
Request Body	**%entity_patch**
Response Body	**%entity**

Изменяет название, описание, теги или опции сущности с соблюдением следующих правил:

- Название сущности должно соответствовать маске \^[a-zA-Z][a-zA-Z0-9_]*$.
- Название сущности должно быть уникально в разрезе каталога сущностей.

В параметр `fields` передаются названия полей для изменения (name/description/tags). Если атрибут указан в `fields`, а в request body он null или не передан, то будет подставляться значение null.

Набор ключей изменять **нельзя**.

3.5.5 Get Entities

Получить список сущностей (GET /entities)

Уровень доступа	**read**
Request Body	-
Response Body	%entity

Метод возвращает список сущностей. В качестве элемента списка используется схема %entity.

Для фильтрации запроса доступны параметры:

- entity_rk: List[int]
- name: mask
- desc: mask
- key_rk: List[int]
- key_name: List[mask]
- tags: List[str]
- limit: int
- offset: int

Параметры `limit` и `offset` отвечают за размер и сдвиг выборки соответственно. При значении `limit: null` возвращается дефолтное количество объектов, которое определяется параметром `APP__DEFAULT_ITEMS_LIMIT`. Максимальное значение `limit` определяется параметром `APP__MAX_ITEMS_LIMIT`.

Для параметров `name` и `desc` доступна исключающая фильтрация при добавлении символа `^` в начало значения маски фильтрации.

Получить информацию по сущности (GET /entities/{entity_rk})

Уровень доступа	**read**
Request Body	-
Response Body	%entity

Возвращает полную информацию по сущности с раскрытием информации по ключам в поле `keys_info` при значении параметра `expand=True`. Схема элемента в списке `keys_info` соответствует схеме `GET /keys/{key_rk}` при значении `expand=True` При значении `expand=False` поле `"keys_info": null`.

3.6 Keys

3.6.1 Keys

Ключи являются основой для создания сущности, где несколько ключей (от одного и больше) объединяется в один набор.

Фактически является столбцом в таблице.

Пример: CUSTOMER_ID (идентификатор клиента), CAMPAIGN_ID (идентификатор кампании) и т.д.

3.6.2 Create Key

Создать ключ (POST /keys)

Уровень доступа	**load**
Request Body	**%key**
Response Body	**%key**

Схема body: %key

В каталог ключей добавляется новый ключ.

Название ключа должно соответствовать маске \^[a-zA-Z][a-zA-Z0-9_]*$. Название ключа должно быть уникально в разрезе каталога ключей.

3.6.3 Delete Keys

Удалить ключи (DELETE /keys)

Уровень доступа	**load**
Request Body	list[int]
Response Body	**-**

Метод удаляет указанные ключи, если они не используются ни в одной из сущностей.

3.6.4 Edit Key

Редактировать ключ (PATCH /keys/{key_rk})

Уровень доступа	**load**
Request Body	**%key_patch**
Response Body	**%key**

Изменяет название или описание ключа с соблюдением следующих правил:

- Название ключа должно соответствовать маске \^[a-zA-Z][a-zA-Z0-9_]*$.
- Название ключа должно быть уникально в разрезе каталога ключей.

В параметр `fields` передаются названия полей для изменения (name/description). Если атрибут указан в `fields`, а в request body он null или не передан, то будет подставляться значение null.

Тип данных ключа изменять **нельзя**.

3.6.5 Get Keys

Получить список ключей (GET /keys)

Уровень доступа	**read**
Request Body	**-**
Response Body	**%key**

Метод возвращает список ключей. В качестве элемента списка используется схема %key.

Для фильтрации запроса доступны параметры:

- key_rk: List[int]
- name: mask
- desc: mask
- fs_type: List[str]
- limit: int
- offset: int

Параметры `limit` и `offset` отвечают за размер и сдвиг выборки соответственно. При значении `limit: null` возвращается дефолтное количество объектов, которое определяется параметром `APP__DEFAULT_ITEMS_LIMIT`. Максимальное значение `limit` определяется параметром `APP__MAX_ITEMS_LIMIT`.

Для параметров `name` и `desc` доступна исключающая фильтрация при добавлении символа `^` в начало значения маски фильтрации.

Получить информацию по ключу (GET /keys/{key_rk})

Возвращает полную информацию по ключу с раскрытием информации типа данных `fs_type` в поле `fs_type_info` при значении параметра `expand=True`. При значении `expand=False` поле `"fs_type_info": null`.

3.7 Datasets

3.7.1 Создать датасет (POST /datasets)

Уровень доступа	**load**
Request Body	`%dataset`
Response Body	`%dataset`

Info

Существование таблицы и столбцов в ней в при регистрации не проверяется

Схемы:

- `%feature`
- `%instance`

Регистрирует таблицу в системе. Зарегистрированная таблица представляет собой объект "датасет".

Вместе с записью о самой таблице, в системе также формируются записи о переменных и экземплярах сущностей в этой таблице.

Переменные (features)

Схема: `%feature`

Секция описывает всю необходимую информацию о столбцах в таблице, включая описание, тип данных и роль.

Одни и те же столбцы таблицы при регистрации могуть быть заполнены в разных секциях (`features`, `instances`, `db_date`, `db_date_end`), но на один столбец зарегистрируется только одна переменная. В зависимости от того в какой секции указан столбец он получит одну из следующих ролей:

- `"date"` - если столбец найден в `db_date`
- `"date_end"` - если столбец найден в `db_date_end`
- `"key"` - если столбец найден в `instances`
- `"feature"` - если столбец найден в `features`

Экземпляры сущности (instances)

Схема: `%instance`

Секция описывает какие сущности присутствуют в таблице и в каких столбцах находятся их ключи.

В таблице может быть любой набор сущностей, а одна и та же сущность может быть представлена в таблице несколько раз (например 'Заемщик' и 'Созаемщик' лежат в разных полях таблицы, но оба представляют сущность 'Клиент')

При помощи флага `"unique"` указывается, что в таблице нет дублей по ключу этой сущности (например 'Заемщик' в таблице с клиентами должен быть объявлен как `unique=true`, а в таблице с договорами `unique=false`)

3.7.2 Получить список датасетов (GET /datasets)

Уровень доступа	**read**
Request Body	-
Response Body	`%dataset_summary`

Возвращает список датасетов, которые удовлетворяют фильтрам:

- `dataset_rk` : list[int]
- `name` : mask
- `desc` : mask
- `entity_rk` : list[int]
- `entity_name` : mask - не поддерживает not like маски `'^...'`
- `source_rk` : list[int]
- `db_schema` : mask
- `db_table` : mask
- `author` : mask
- `tags` : list[str]
- `limit` : int
- `offset` : int

Параметры `limit` и `offset` отвечают за размер и сдвиг выборки соответственно. При значении `limit: null` возвращается дефолтное количество объектов, которое определяется параметром `APP__DEFAULT_ITEMS_LIMIT`. Максимальное значение `limit` определяется параметром `APP__MAX_ITEMS_LIMIT`.

3.7.3 Удалить датасеты (DELETE /datasets)

Уровень доступа	**load**
Request Body	list[int]
Response Body	`%run_info`

Удаляет запись о таблице из системы.

Удаление происходит в фоновом режиме. Если передан флаг `drop_data=true`, то удаляет еще и саму таблицу с данными.

Если удаление датасета влечет за собой удаление связанного задания, то датасет удалится только если задание не находится на расписании.

3.7.4 Получить детальную информацию о датасете (GET /datasets/{dataset_rk})

Уровень доступа	**read**
Request Body	-
Response Body	`%dataset`

Схемы:

- `%dataset`
- `%feature`
- `%instance`

Возвращает детальную информацию по датасету.

Если передан параметр `expand=true`, то в схемах `%feature` и `%instance` заполняются `+"expand"` поля:

- `%feature.fs_type_info` - детальная информация по `%fs_type`
- `%instance.entity_info` - детальная информация по `%entity`

3.7.5 Изменить датасет (PATCH /datasets/{dataset_rk})

Уровень доступа	**load**
Request Body	- `%dataset_patch`
Response Body	`%dataset`

Info

Существование таблицы и столбцов в ней в при этом не проверяется

- `%dataset_patch`
- `%feature`
- `%instance`

Изменяет информацию о зарегистрированной таблице, при этом изменяются только переданные поля.

- Если переданы только простые поля `"name"`, `"desc"`, `"db_schema"`, `"db_table"`, то обновление происходит тривиальным образом - новые значения заменяют старые
- Если переданы поля, затрагивающие набор переменных и экземпляров (`"db_date"`, `"db_date_end"`, `"features"`, `"instances"`), то произойдет полное пересоздание переменных и экземпляров с учетом новой информации. При этом система постарается сохранить уже сгенерированные ключи `feature_rk` и `instance_rk`, для столбцов с теми же именами, но не гарантирует этого.

3.7.6 Провалидировать датасет (GET /datasets/{dataset_rk}/health)

Уровень доступа	**read**
Request Body	-
Response Body	`%dataset_health`

Выполняет проверки по описанной структуре датасета и возвращает отчет.

Проверки `source`

- Если `virtual` = `false`, то `source_rk` не пустой
- Если `source_rk` не пустой то такой источник существует
- Если `virtual` = `true`, то проверки в бд не производятся даже если задан `source_rk`

Проверки `db_date` и `db_date_end`

- Если колонка задана, то она существует в `features`

Проверки features

- Колонка существует
- `fs_type` фичи совместим с типом колонки в базе через `py_type`

Проверки `instances`

- Сущность существует
- Все keys сущности есть в `instance.columns`
- Все `columns` есть в `entity.keys` в качестве ключа
- Все ключи есть в `features`
- warning! `fs_type` фичи не совпадает с `fs_type` ключа

3.7.7 Получить информацию о связанных заданиях датасета (GET /datasets/-/jobs)

Уровень доступа	**read**
Request Body	-
Response Body	`%dataset_job_summary`

Возвращает список связанных с датасетом заданий, которые удовлетворяют фильтрам:

- `dataset_rk` : list[int]
- `job_rk` : list[int]
- `role` : list[str]
- `state` : list[enum]
- `dispose` : list[enum]
- `limit` : int
- `offset` : int

3.8 Links

3.8.1 Создать связи датасетов (POST /links)

Уровень доступа	**load**
Request Body	**list[%link]**
Response Body	**list[%link]**

Создает новые связи датасетов.

Для каждой переданной связи выполняются следующие проверки:

- Значения `dataset_rk` и `parent_dataset_rk` не совпадают
- Датасеты с ключами `dataset_rk` и `parent_dataset_rk` существуют
- Имя связи уникально
- Пара `dataset_rk` + `parent_dataset_rk` уникальна

3.8.2 Получить список связей датасетов (GET /links)

Уровень доступа	**read**
Request Body	**-**
Response Body	**list[%link]**

Получает список связей датасетов, которые удовлетворяют фильтрам:

- link_rk: list[int]
- dataset_rk: list[int]
- parent_dataset_rk: list[int]
- name: mask
- desc: mask

3.8.3 Изменить связь датасетов (PUT /links/{link_rk})

Уровень доступа	**load**
Request Body	**%link_patch**
Response Body	**%link**

Обновляет связь датасета по переданному ключу, при этом изменяются только переданные аттрибуты.

Для обновленной связи выполнются те же проверки, что и при создании новых связей датасетов.

3.8.4 Удалить связи датасетов (DELETE /links)

Уровень доступа	**load**
Request Body	**list[int]**
Response Body	**-**

Удаляет связи датасетов по ключам из запроса.

3.9 Features

Переменные создаются только в рамках регистрации датасета.

3.9.1 Получить список переменных (GET /features)

Уровень доступа	**read**
Request Body	-
Response Body	`%feature_summary`

Возвращает список переменных, которые удовлетворяют фильтрам:

- `feature_rk` : list[int]
- `name` : mask
- `desc` : mask
- `role` : enum
- `db_schema` : mask
- `db_table` : mask
- `db_column` : mask
- `dataset_rk` : list[int]
- `dataset_name` : mask
- `fs_type` : list[str]
- `entity_rk` : list[int]
- `entity_name` : mask - не поддерживает not like маски `'^...'`
- `author` : mask
- `limit` : int
- `offset` : int

Параметры `limit` и `offset` отвечают за размер и сдвиг выборки соответственно. При значении `limit: null` возвращается дефолтное количество объектов, которое определяется параметром `APP__DEFAULT_ITEMS_LIMIT`. Максимальное значение `limit` определяется параметром `APP__MAX_ITEMS_LIMIT`.**

При этом:

- Часть атрибутов берется непосредственно с датасета (`db_schema`, `db_table`, `dataset_name`)
- Некоторые атрибуты могут выступать фильтрами, но не выводятся в итоговом списке (`entity_rk`, `entity_name`)

3.9.2 Получить детальную информацию о переменной (GET /features/{feature_rk})

Уровень доступа	**read**
Request Body	-
Response Body	`%feature`

Возвращает детальную информаци о переменной.

Если передан параметр `expand=true`, то в схеме заполняются `+"expand"` поля:

- `fs_type_info` - детальная информация по `%fs_type`

3.9.3 Изменить переменную

Уровень доступа	**load**
Request Body	`%feature_patch`
Response Body	`%feature`

Изменяет информацию о переменной, при этом изменяются только переданные поля. Обновление происходит тривиальным образом - новые значения заменяют старые.

3.10 Instances

3.10.1 Получить список экземпляров сущностей (GET /isntances)

Уровень доступа	**read**
Request Body	-
Response Body	`%instance_summary`

Возвращает список экземпляров сущностей, которые удовлетворяют фильтрам:

- `instance_rk` : list[int]
- `name` : mask
- `desc` : mask
- `unique` : bool
- `dataset_rk` : list[int]
- `dataset_name` : mask
- `entity_rk` : list[int]
- `entity_name` : mask
- `author` : mask
- `limit` : int
- `offset` : int

Параметры `limit` и `offset` отвечают за размер и сдвиг выборки соответственно. При значении `limit: null` возвращается дефолтное количество объектов, которое определяется параметром `APP__DEFAULT_ITEMS_LIMIT`. Максимальное значение `limit` определяется параметром `APP__MAX_ITEMS_LIMIT`.

При этом часть атрибутов берется со связанных объектов (`dataset_name`, `entity_name`)

3.10.2 Получить детальную информацию по экземляру сущности (GET /instances/{instance_rk})

Уровень доступа	**read**
Request Body	-
Response Body	`%instance`

Возвращает детальную информацию об экземпляре сущности.

Если передан параметр `expand=true`, то в схеме заполняются `+"expand"` поля:

- `entity_info` - детальная информация по `%entity`

3.10.3 Изменить экземпляр сущности (PATCH /instances/{instance_rk})

Уровень доступа	**load**
Request Body	`%instance_patch`
Response Body	`%instance`

Изменяет информацию об экземпляре сущности, при этом изменяются только переданные поля. Обновление происходит тривиальным образом - новые значения заменяют старые.

3.11 Permissions

3.11.1 Permissions

Управление правами доступа пользователей к датасетам, загрузчикам и источникам данных.

3.11.2 Delete

Удаление прав доступа (DELETE /permissions)

Уровень доступа	**transform**
Request Body	list[int]
Response Body	**-**

Допускается удаление *только персональных* прав доступа при наличии максимального уровня доступа к объекту.

3.11.3 Edit

Редактирование прав доступа (PATCH /permissions)

Уровень доступа	transform
Request Body	%permission
Response Body	%permission

Допускается изменение прав доступа как общих, так и персональных, при наличии максимального уровня доступа к объекту.

3.11.4 Info

Получить список доступов (GET /permissions)

Уровень доступа	**read**
Request Body	-
Response Body	%permission

Возвращается полный список доступов к объектам (датасеты, загрузчики, источники данных), включая в себя как общие правила доступа, так и персональные.

Для фильтрации запроса доступны параметры:

- `object_type` : str
- `object_rk` : list[int]
- `name` : mask
- `limit` : int
- `offset` : int

Параметры `limit` и `offset` отвечают за размер и сдвиг выборки соответственно. При значении `limit: null` возвращается дефолтное количество объектов, которое определяется параметром `APP__DEFAULT_ITEMS_LIMIT`. Максимальное значение `limit` определяется параметром `APP__MAX_ITEMS_LIMIT`.

Получить информацию о пользователе (GET /user)

Уровень доступа	**read**
Request Body	-
Response Body	%user

3.12 Files

Файлы в подключенной файловой системе **S3**.

3.12.1 Загрузить файл (POST /files/s3)

Уровень доступа	**load**
Request Body	-
Response Body	str

Возвращает ссылку для загрузки файла напрямую в s3 под именем, указанным параметр в `path`. Для использования ссылки необходимо выполнить запрос PUT и передать контент в body запроса.

3.12.2 Получить список файлов (GET /files/s3)

Уровень доступа	**read**
Request Body	-
Response Body	list[str]

Возвращает список файлов в файловом хранилище. В response body отсутствуют файлы из поддиректорий. Если в качестве `path_prefix` передать имя директории с символом `/` на конце, то в ответе будет содержимое поддиректории

3.12.3 Удалить файлы (DELETE /files/s3)

Уровень доступа	**load**
Request Body	list[str]
Response Body	-

Удаляет файлы из файлового хранилища. Если в качестве ключа передать имя поддиректории с символом / на конце, то удалится вся поддиректория вместе с содержимым

3.12.4 Создать ссылку для скачивания файла (POST /files/s3/link)

Уровень доступа	**read**
Request Body	str
Response Body	-

Возвращает ссылку на скачивание файла из файлового хранилища. Ссылка имеет срок жизни и перестает работать через некоторое время.

3.12.5 Создать файл из текста (POST /files/s3/text)

Уровень доступа	**load**
Request Body	str
Response Body	str

Создает файл внутри файлового хранилища и записывает в него переданный текст.

3.12.6 Получить содержание текстового файла (GET /files/s3/text)

Уровень доступа	**read**
Request Body	-
Response Body	str

Возвращает содержимое текстового файла в файловом хранилище axiom.

На текущий момент никак не проверяется, является ли файл текстовым.

3.12.7 Получить список столбцов в файле (GET /files/s3/describe)

Уровень доступа	**load**
Request Body	-
Response Body	**%file**

Метод читает до N первых строк файла и возвращает описание файла со списком столбцов в нем, с предполагаемыми типами данных fs_type.

Требование к файлу:

- имеет расширения .csv или .xlsx и является корректным csv или xlsx форматом соответственно
- первая строка в файле - заголовок с именами столбцов
- кодировка файла utf8

Атрибуты %column в респонсе заполняются следующим образом:

- column_num - порядковый номер столбца
- db_column - имя столбца из заголовка в файле
- db_type - тип py_type /ref/fs_type
- fs_type - результат перекодировки по справочнику /ref/type_default

3.13 Sources

Источники данных.

Подробнее про подключение различных источников данных см. "Подключение к данным" Admin Guide.

3.13.1 Создать источник данных (POST /sources)

Уровень доступа	**admin**
Request Body	%source
Response Body	%source

Метод создает запись об источнике данных в таблице meta.v100_source.

3.13.2 Получить список источников данных (GET /sources)

Уровень доступа	**read**
Request Body	-
Response Body	%source

Возвращает список источников данных, которые удовлетворяют фильтрам:

- `source_rk` : list[int]
- `name` : mask
- `desc` : mask
- `limit` : int
- `offset` : int

Параметры `limit` и `offset` отвечают за размер и сдвиг выборки соответственно. При значении `limit: null` возвращается дефолтное количество объектов, которое определяется параметром `APP__DEFAULT_ITEMS_LIMIT`. Максимальное значение `limit` определяется параметром `APP__MAX_ITEMS_LIMIT`.

Источник данных `source_rk` = -1 - это всегда datastore для API v2 и узлов `constructor`.

3.13.3 Удалить источники данных (DELETE /sources)

Уровень доступа	**admin**
Request Body	list[int]
Response Body	-

Удаляет из meta.v100_source записи по ключам из запроса.

Источник `source_rk` = -1 удалить нельзя.

3.13.4 Получить информацию по конкретному источнику данных (GET /sources/{source_rk})

Уровень доступа	**read**
Request Body	-
Response Body	%source

Возвращает запись из meta.v100_source по ключу.

3.13.5 Изменить источник данных (PATCH /sources/{source_rk})

Уровень доступа	**admin**
Request Body	%source_patch
Response Body	%source

Обновляет в meta.v100_source запись по источнику данных. Изменяются только переданные атрибуты.

3.13.6 Получить список таблиц источника (GET /sources/{source_rk}/tables)

Уровень доступа	**load**
Request Body	-
Response Body	%tables

Возвращает список таблиц в конкретной схеме источника данных.

3.13.7 Получить список столбцов в таблице источника (GET /sources/{source_rk}/columns)

Уровень доступа	**load**
Request Body	-
Response Body	%columns

Возвращается список колонок в таблице. Тип данных колонки обогащается значением `fs_type` через справочник `ref_type_default`, где: - db_name = source.url.db_name (тип СУБД источника, взятый из его URL) - db_type = column.db_type (тип данных колонки)

3.13.8 Провалидировать источник (GET /sources/{source_rk}/health)

Уровень доступа	**admin**
Request Body	-
Response Body	%source_health

Возвращается полный отчет по состоянию источника с учетом валидации заполнения справочников типов данных для `db_name` источника.

Метод валидирует источник данных. Проверяется:

- Возможность подключения по адресу из `url`. Если удалось подключиться, то `ok`, иначе `error` по всем пунктам
- Название используемого в `url` диалекта должно быть таким же, как `db_name`
- Наличие схем `work_schema` и `store_schema` и прав на запись в эти схемы. Если схема присутствует и есть права на запись, то `ok`, иначе `error`
- Корректность заполнения справочников типов данных для источника

Проверка заполнения справочников типов данных

Проверяется корректность заполнения справочников типов данных для `db_name` источника:

- Проверяется наличие имплементации типа данных. Если для `fs_type` отсутствует имплементация `db_type`, то `error`
- Проверяется существование `db_type`. Если выполнить преобразование `null` к типу данных `db_type` на источнике не получается, то `error`
- Проверяется совместимость имплементации `db_type` с его дефолтным `fs_type` по алгоритму:
- Для `fs_type` определяется его имплементация `db_type`
- Для найденного `db_type` определяется дефолтный `fs_type`
- Если `py_type` дефолтного `fs_type` отличается от `py_type` исходного `fs_type`, то `error`
- Иначе Если дефолтный `fs_type` отличается от исходного `fs_type`, то `warning`

Если все проверки выше успешно пройдены, то `ok`.

3.13.9 Получить превью таблицы из источника (GET /sources/{source_rk}/preview)

Уровень доступа	**load**
Request Body	-
Response Body	**dict[str, Any]**

Возвращает построчный список значений из каждого аттрибута таблицы источника.

В параметрах запроса необходимо указать:

- `source_rk` - ключ источника данных
- `db_schema` - схема таблицы
- `db_table` - название таблицы
- `limit` - количество строк для выборки. Значение по-умолчанию берется из переменной окружения `APP__DEFAULT_PREVIEW_LIMIT` и не может быть выше значения переменной окружения `APP__MAX_PREVIEW_LIMIT`

Во время выполнения запроса проверяется:

- Существование источника по значению `source_rk`
- Наличие `db_schema` в списке `schemas` источника
- Существование таблицы `db_table` в источнике
- `limit` не превышает значение переменной окружения `APP__MAX_PREVIEW_LIMIT`

3.14 Stats

3.14.1 Получить список статистик по датасетам (GET /stats)

Уровень доступа	**load**
Request Body	**-**
Response Body	`%stat`

Выводит статистики по всей таблице датасета.

Допустимые параметры фильтрации:

- `stat_rk` : `list[int]` - список суррогатных ключей статистик
- `dataset_rk` : `list[int]` - список суррогатных ключей датасетов
- `indicator` : `list[enum]` - список статистических показателей
- `limit` : `int`
- `offset` : `int`

Параметры `limit` и `offset` отвечают за размер и сдвиг выборки соответственно. При значении `limit: null` возвращается дефолтное количество объектов, которое определяется параметром `APP__DEFAULT_ITEMS_LIMIT`. Максимальное значение `limit` определяется параметром `APP__MAX_ITEMS_LIMIT`.

3.14.2 Удалить статистики по датасетам (DELETE /stats)

Уровень доступа	**load**
Request Body	**list[int]**
Response Body	**-**

Удаляет статистики по всей таблице датасета согласно переданным суррогатным ключам статистик.

3.14.3 Получить список статистик по датасетам в разрезе даты (GET /stats/common)

Уровень доступа	**load**
Request Body	**-**
Response Body	`%stat_common`

Выводит статистики по датасетам в разрезе даты.

Допустимые параметры фильтрации:

- `stat_rk` : `list[int]` - список суррогатных ключей статистик
- `dataset_rk` : `list[int]` - список суррогатных ключей датасетов
- `indicator` : `list[enum]` - список статистических показателей
- `date` : `list[datetime]` - список дат, на которые рассчитаны статистики
- `limit` : `int`
- `offset` : `int`

Параметры `limit` и `offset` отвечают за размер и сдвиг выборки соответственно. При значении `limit: null` возвращается дефолтное количество объектов, которое определяется параметром `APP__DEFAULT_ITEMS_LIMIT`. Максимальное значение `limit` определяется параметром `APP__MAX_ITEMS_LIMIT`.

3.14.4 Удалить статистики по датасетам в разрезе даты (DELETE /stats/common)

Уровень доступа	**load**
Request Body	**list[int]**
Response Body	-

Удаляет статистики по датасетам в разрезе даты согласно переданным суррогатным ключам статистик.

3.14.5 Получить список статистик по экземплрам сущности (GET /stats/instances)

Уровень доступа	**load**
Request Body	-
Response Body	`%stat_instance`

Выводит статистики по экземплярам сущности.

Допустимые параметры фильтрации:

- `stat_rk` : `list[int]` - список суррогатных ключей статистик
- `dataset_rk` : `list[int]` - список суррогатных ключей датасетов
- `instance_rk` : `list[int]` - список суррогатных ключей экземпляров сущностей
- `indicator` : `list[enum]` - список статистических показателей
- `date` : `list[datetime]` - список дат, на которые рассчитаны статистики
- `limit` : `int`
- `offset` : `int`

Параметры `limit` и `offset` отвечают за размер и сдвиг выборки соответственно. При значении `limit: null` возвращается дефолтное количество объектов, которое определяется параметром `APP__DEFAULT_ITEMS_LIMIT`. Максимальное значение `limit` определяется параметром `APP__MAX_ITEMS_LIMIT`.

3.14.6 Удалить статистики по экземплярам сущности (DELETE /stats/instances)

Уровень доступа	**load**
Request Body	**list[int]**
Response Body	-

Удаляет статистики по экземплярам сущности согласно переданным суррогатным ключам статистик.

3.14.7 Получить список статистик по переменным (GET /stats/features)

Уровень доступа	**load**
Request Body	-
Response Body	`%stat_feature`

Выводит статистики по переменным.

Допустимые параметры фильтрации:

- `stat_rk` : `list[int]` - список суррогатных ключей статистик
- `dataset_rk` : `list[int]` - список суррогатных ключей датасетов
- `feature_rk` : `list[int]` - список суррогатных ключей переменных
- `indicator` : `list[enum]` - список статистических показателей
- `date` : `list[datetime]` - список дат, на которые рассчитаны статистики
- `limit` : `int`
- `offset` : `int`

Параметры `limit` и `offset` отвечают за размер и сдвиг выборки соответственно. При значении `limit: null` возвращается дефолтное количество объектов, которое определяется параметром `APP__DEFAULT_ITEMS_LIMIT`. Максимальное значение `limit` определяется параметром `APP__MAX_ITEMS_LIMIT`.

3.14.8 Удалить статистики по переменным (DELETE /stats/features)

Уровень доступа	**load**
Request Body	**list[int]**
Response Body	**-**

Удаляет статистики по переменным согласно переданным суррогатным ключам статистик.

3.15 System Endpoints

Технические эндпоинты, доступные из корня приложения.

3.15.1 Get system info (GET /info)

Уровень доступа	-
Request Body	-
Response Body	%info

Возвращает версию приложения.

3.15.2 Get system health (GET /health)

Уровень доступа	-
Request Body	-
Response Body	%health

Возвращает агрегированный статус доступности системы с разбивкой на основные компоненты:

- БД meta
- БД datastores
- S3
- Executor
- Application Lock

3.15.3 Get workers readiness (GET /health/readiness)

Уровень доступа	-
Request Body	-
Response Body	dict[{worker_host}, %worker_health]

Возвращает информацию о готовности Celery воркеров к работе.

Проверяет соответствие версий (`приложения`, `ядра`, `движка`) между backend'ом и всеми Celery-воркерами.

Response:

- Если версии не совпадают у воркера:
- `state`: `error`
- `detail`: описание ошибки
- Если нет воркеров с совпадающими версиями — `ServiceUnavailableException`

3.15.4 Get system metrics (GET /metrics)

Уровень доступа	-
Request Body	-
Response Body	**str**

Возвращает метрики системы в формате Prometheus

3.15.5 Get system loggers (GET /loggers)

Уровень доступа	-
Request Body	-
Response Body	**list[str]**

Возвращает список доступных логгеров в системе.

3.15.6 Get main logger level (GET /loggers/main)

Уровень доступа	-
Request Body	-
Response Body	**%logger**

Возвращает установленный уровень логирования.

3.15.7 Set main logger level (POST /loggers/main)

Уровень доступа	-
Request Body	-
Response Body	-

Устанавливает уровень логирования по переданному значению аргумента `level` (см. %logger).

3.16 Tag

3.16.1 Получение списка тегов (GET /tag)

Уровень доступа	**read**
Request Body	**-**
Response Body	**list[str]**

Возвращается полный список существующих тегов.

Поддерживает фильтрацию по маске тега (параметр запроса `mask`), в т.ч. исключающую фильтрацию при добавлении символа `^` в начало значения маски фильтрации.

3.16.2 Удаление тегов (DELETE /tag)

Уровень доступа	**load**
Request Body	**list[str]**
Response Body	**-**

Удаляет тег из справочника тегов. Со связанных объектов тег не удаляется.

3.17 Admin

3.17.1 Admin Application Lock

Get Application Lock (GET /admin/app_lock)

Уровень доступа	**admin**
Request Body	-
Response Body	`%app_lock`

Возвращает информацию о текущей блокирующей задаче.

Блокирующие задачи - это `celery`-задачи, которые не могут выполняться параллельно. Запуск такой задачи возможен только в том случае, если на момент её запуска в системе отсутствует информация о блокировке.

Блокировка устанавливается в момент запуска блокирующей `celery`-задачи и может быть снята при выполнении одного из следующих условий:

1. `Celery`-задача завершилась (результат не имеет значения)
2. Превышен таймаут блокировки и запрошено выполнение другой блокирующей задачи (таймаут настраивается через переменную `$APP__LOCK_TIMEOUT`)
3. Блокировка снята администратором через соответствующий эндпоинт

На текущий момент блокирующими задачами являются:

- удаление датасетов v100
- импорт каталога v2.

Если блокировки нет, то в ответе вернется схема с `null`-ами во всех полях.

Release Application Lock (DELETE /admin/app_lock)

Уровень доступа	**admin**
Request Body	-
Response Body	-

Снимает блокировку, принудительно останавливает блокирующую `celery`-задачу

3.17.2 Admin Executor

Get Task Information From Executor (GET /admin/tasks/{task_id})

Уровень доступа	**admin**
Request Body	**-**
Response Body	**str**

Возвращает статус задачи на основе информации из экзекьютора. Если задача выполняется, то статус = `STARTED`, если зарезервирована, то `PENDING`, иначе `UNKNOWN`.

В качестве параметра пути принимает идентификатор задачи `task_id: uuid`

3.17.3 Admin Migrate

Migrate Metastore (POST /admin/migrate_metastore)

Уровень доступа	**admin**
Request Body	**-**
Response Body	**-**

Обновляет объекты в БД meta до актуальной версии.

3.17.4 Admin Reset

Reset Metastore (POST /admin/reset_metastore)

Уровень доступа	**admin**
Request Body	**-**
Response Body	str

Удаляет и пересоздает все объекты в БД meta в исходном виде.

Reset Airflow History (POST /admin/reset_airflow_history)

Уровень доступа	**admin**
Request Body	**-**
Response Body	str

Удаляет историю запусков дагов в Airflow.

Ref Init (POST /admin/ref_init)

Уровень доступа	**admin**
Request Body	**-**
Response Body	str

Заполняет системные справочники стандартными значениями.

Reset system refs (POST /admin/reset_sys)

Уровень доступа	**admin**
Request Body	**-**
Response Body	str

Удаляет текущие настройки логирования и приводит их к значениям по умолчанию.

Получить информацию о текущем профиле логирования (GET /admin/logger)

Уровень доступа	**admin**
Request Body	**-**
Response Body	**%log_profile**

Возвращаются настройки активного профиля логирования.

Установить профиль логирования (POST /admin/logger/{name})

Уровень доступа	**admin**
Request Body	**%log_profile**
Response Body	**%log_profile**

Устанавливает профиль логирования согласно переданным настройкам. Профиль становится активным, заменяя предыдущий.

Скачать логи (GET /admin/download_logs)

Уровень доступа	**admin**
Request Body	**-**
Response Body	File

Скачать архив логов.

Получить список профилей логирования (GET /sys/logging_profiles)

Уровень доступа	**admin**
Request Body	**-**
Response Body	**%log_profile_summary**

Возвращается список всех профилей логирования с учетом фильтров.

Для фильтрации запроса доступны параметры:

- `name` : mask
- `desc` : mask
- `active_flg` : bool
- `limit` : int
- `offset` : int

Параметры `limit` и `offset` отвечают за размер и сдвиг выборки соответственно. При значении `limit: null` возвращается дефолтное количество объектов, которое определяется параметром `APP__DEFAULT_ITEMS_LIMIT`. Максимальное значение `limit` определяется параметром `APP__MAX_ITEMS_LIMIT`.

Получить профиль логирования (GET /sys/logging_profiles/{name})

Уровень доступа	**admin**
Request Body	**-**
Response Body	**%log_profile**

Возвращается профиль логирования.

Изменить профиль логирования (GET /sys/logging_profiles/{name})

Уровень доступа	**admin**
Request Body	**%log_profile**
Response Body	**%log_profile**

Редактирование существующего профиля логирования.

3.18 Reference tables

3.18.1 Reference Tables

Справочники.

Data types

Внутренние типы данных приложения (`fs_type`) являются логической сущностью, которая объединяет в себе отображения типов данных из разных СУБД (`db_type`).
Основная терминология:

- `fs_type` - тип данных. Логическая сущность.
- `db_type` - тип данных в СУБД. Соответствие между `fs_type` и `db_type` называется имплементацией.

При регистрации/описании объектов используются только `fs_type`. Для корректной работы приложения с данными необходимо настроить имплементации типов данных для диалектов СУБД.

Пример:

fs_type	db_type	db_name
AMOUNT	decimal(26,2)	postgresql
AMOUNT	decimal	spark
FLOAT	float8	postgresql
FLOAT	double	spark
INT	integer	postgresql
INT	int	spark

Таким образом одно значение `fs_type` может объединять в себе несколько имплементаций на разные диалекты, что позволяет работать одновременно с разными СУБД используя один и тот же справочник типов данных.

Базовое наполнение справочника `fs_type`, а так же имплементаций, предоставляется по умолчанию. Доступна пользовательская настройка справочников и имплементаций в разделе Ref API в зависимости от потребностей пользователя.

Помимо справочника имплементаций типов данных `fs_type` на типы данных диалекта СУБД `db_type` существует так же и справочник имплементаций "по умолчанию". Он представляет собой список типов данных диалекта, разбитых на категории. Каждой категории присваивается тип данных по умолчанию `default_fs_type` из справочника `fs_type`. Отличие справочника "по умолчанию" от справочника имплементаций заключается в том, справочник "по умолчанию" содержит в себе расширенный набор типов данных диалекта. Это необходимо для того, чтобы обрабатывать типы данных СУБД, не

указанные в наборе имплементаций, и присваивать им соответсвующий `fs_type`. Пример справочника типов данных по умолчанию:

db_type	default_fs_type	db_name
bigint	BIG_INT	postgresql
bigserial	BIG_INT	postgresql
integer	BIG_INT	postgresql
json	BIG_STRING	postgresql
text	BIG_STRING	postgresql
character	BIG_STRING	postgresql

Справочник типов данных по умолчанию используется в ядре приложения при определении типов данных полей таблиц, когда это в явном виде не сделал пользователь, например, при регистрации сегмента (где `fs_type` для каждой фичи определяется через справочник типов данных по умолчанию через `default_fs_type`).

Заводское наполнение всех справочников типов данных выполняется через эндпоинт Ref init в API.

Создание пользовательского наполнения справочников осуществляется через эндпоинты, описанные в разделе "Create" <ссылка на раздел с описанием эндпоинтов>.

3.18.2 Data Types Implementation (db_types)

Создать реализацию типа данных (POST /ref/db_types)

Уровень доступа	**admin**
Request Body	**%db_type**
Response Body	**%db_type**

Создает новую запись о реализации унифицированного типа данных в справочнике, согласно параметрам из request. Если такая запись по комбинации значений (db_name + fs_type) уже существует, то возвращает ошибку

Получить реализацию типов данных (GET /ref/db_types)

Уровень доступа	**read**
Request Body	-
Response Body	**%db_type**

Возвращает справочник реализации унифицированных типов с учетом фильтров.

Для фильтрации запроса доступны параметры:

- `db_name` : list[str]
- `fs_type` : list[str]
- `limit` : int
- `offset` : int

Параметры `limit` и `offset` отвечают за размер и сдвиг выборки соответственно. При значении `limit: null` возвращается дефолтное количество объектов, которое определяется параметром `APP__DEFAULT_ITEMS_LIMIT`. Максимальное значение `limit` определяется параметром `APP__MAX_ITEMS_LIMIT`.

Пересоздать реализацию типов данных (PUT /ref/db_types)

Уровень доступа	**admin**
Request Body	**%db_type**
Response Body	**%db_type**

Удаляет текущее наполнение справочника и заменяет его значениями из входного списка.

Удалить реализацию типов данных (DELETE /ref/db_types)

Уровень доступа	**admin**
Request Body	list[str]
Response Body	-

Удаляет записи из справочника реализации типов данных по списку из request body.

3.18.3 Execution Profiles

Создать профиль исполнения (POST /ref/exec_profiles)

Уровень доступа	**admin**
Request Body	%exec_profile
Response Body	%exec_profile

В справочник профилей исполнения добавляется новая запись. Для использования сохраненной конфигурации профиля исполнения достаточно указать его имя в `job_options.execution_profile` при создании/запуске цепочки.

Получить профили исполнения (GET /ref/exec_profiles)

Уровень доступа	**read**
Request Body	-
Response Body	%exec_profile

Возвращается список профилей исполнения с учетом фильтров.

Для фильтрации запроса доступны параметры:

- `name` : mask
- `limit` : int
- `offset` : int

Параметры `limit` и `offset` отвечают за размер и сдвиг выборки соответственно. При значении `limit: null` возвращается дефолтное количество объектов, которое определяется параметром `APP__DEFAULT_ITEMS_LIMIT`. Максимальное значение `limit` определяется параметром `APP__MAX_ITEMS_LIMIT`.

Изменить профиль исполнения (PATCH /ref/exec_profiles/{name})

Уровень доступа	**admin**
Request Body	%exec_profile_patch
Response Body	%exec_profile

Метод позволяет изменить как имя профиля исполнения, так и его конфигурацию. При изменении имени профиля исполнения, новое имя должно быть уникальным в разрезе справочника.

Удалить профили исполнения (DELETE /ref/exec_profiles)

Уровень доступа	**admin**
Request Body	list[str]
Response Body	-

Удаляет профили исполнения из справочника по списку их названий из request body.

3.18.4 Data Types (fs_types)

Создать новый тип данных (POST /ref/fs_types)

Уровень доступа	**admin**
Request Body	**%fs_type**
Response Body	**%fs_type**

Создает новый тип данных в справочнике, согласно параметрам из request. Если такой тип с таким же именем (fs_type) уже существует, то возвращает ошибку.

Допустимые значения py_type:

- str
- int
- float
- bool
- date
- datetime
- decimal

Получить список типов данных (GET /ref/fs_types)

Уровень доступа	**read**
Request Body	**-**
Response Body	**%fs_type**

Возвращает справочник унифицированных типов с учетом фильтров.

Для фильтрации запроса доступны параметры:

- `fs_type` : list[str]
- `py_type` : list[str]
- `limit` : int
- `offset` : int

Параметры `limit` и `offset` отвечают за размер и сдвиг выборки соответственно. При значении `limit: null` возвращается дефолтное количество объектов, которое определяется параметром `APP__DEFAULT_ITEMS_LIMIT`. Максимальное значение `limit` определяется параметром `APP__MAX_ITEMS_LIMIT`.

Пересоздать типы данных (PUT /ref/fs_types)

Уровень доступа	**admin**
Request Body	**%fs_type**
Response Body	**%fs_type**

Удаляет текущее наполнение справочника и заменяет его значениями из входного списка, при условии, что для всех существующих ключей в новом справочнике существует `fs_type`.

Удалить типы данных (DELETE /ref/fs_types)

Уровень доступа	**admin**
Request Body	list[str]
Response Body	**-**

Удаляет из справочника унифицированные типы по списку из request body, если они не используются в существующих ключах.

3.18.5 Default Data Type (type_defaults)

Установить тип данных по-умолчанию (POST /ref/type_defaults)

Уровень доступа	**admin**
Request Body	%type_default
Response Body	%type_default

Для пары db_name и db_type устанавливает унифицированный тип по-умолчанию. Данный тип будет использоваться при авто-разметке столбцов. Если такая запись по комбинации значений (db_name + db_type) уже существует, то возвращает ошибку.

Тип данных `db_type` допустимо указывать как с размерностью, так и без нее.

При авто-разметке в первую очередь определяется `default_fs_type` из записи, в которой `db_type` совпадает с учетом размерности, во вторую очередь - без учета размерности.

Получить тип данных по-умолчанию (GET /ref/type_defaults)

Уровень доступа	**read**
Request Body	-
Response Body	%type_default

Возвращает справочник унифицированных типов по-умолчанию с учетом фильтров.

Для фильтрации запроса доступны параметры:

- `db_name` : list[str]
- `limit` : int
- `offset` : int

Параметры `limit` и `offset` отвечают за размер и сдвиг выборки соответственно. При значении `limit: null` возвращается дефолтное количество объектов, которое определяется параметром `APP__DEFAULT_ITEMS_LIMIT`. Максимальное значение `limit` определяется параметром `APP__MAX_ITEMS_LIMIT`.

Пересоздать типы данных по-умолчанию (PUT /ref/type_defaults)

Уровень доступа	**admin**
Request Body	%type_default
Response Body	%type_default

Удаляет текущее наполнение справочника и заменяет его значениями из входного списка.

Удалить типы данных по-умолчанию (DELETE /ref/type_defaults)

Уровень доступа	**admin**
Request Body	list[str]
Response Body	-

Удаляет записи из справочника по списку из request body.

3.18.6 Indicators for Data Types (type_indicators)

Установить доступный индикатор для типа данных (POST /ref/type_indicators)

Уровень доступа	**admin**
Request Body	**%type_indicator**
Response Body	**%type_indicator**

Для унифицированного типа устанавливает индикатор, доступный для расчета. Если такая запись по комбинации значений (fs_name + indicator) уже существует, то возвращает ошибку.

Получить доступные индикаторы для типов данных (GET /ref/type_indicators)

Уровень доступа	**read**
Request Body	**-**
Response Body	**%type_indicator**

Возвращает справочник доступных индикаторов для унифицированных типов с учетом фильтров.

Для фильтрации запроса доступны параметры:

- `fs_type` : list[str]
- `limit` : int
- `offset` : int

Параметры `limit` и `offset` отвечают за размер и сдвиг выборки соответственно. При значении `limit: null` возвращается дефолтное количество объектов, которое определяется параметром `APP__DEFAULT_ITEMS_LIMIT`. Максимальное значение `limit` определяется параметром `APP__MAX_ITEMS_LIMIT`.

Пересоздать доступные индикаторы для типов данных (PUT /ref/type_indicators)

Уровень доступа	**admin**
Request Body	**%type_indicator**
Response Body	**%type_indicator**

Удаляет текущее наполнение справочника и заменяет его значениями из входного списка.

Удалить доступные индикаторы для типов данных (DELETE /ref/type_indicators)

Уровень доступа	**admin**
Request Body	list[str]
Response Body	**-**

Удаляет записи из справочника по списку из request body.

3.19 Error codes

3.19.1 /admin

- `/invalid-pass` - При вызове reset метода был неправильно указан passphrase
- `/other-error` - Прочие ошибки, см. detail

3.19.2 /dataset

- `/dataset/invalid-chain` - Некорректная цепочка связей

3.19.3 /fat_constructor

- `/multiple-report-dates` - Dataset must contain exactly one report_dttm value

3.19.4 /fat_merge

- `/invalid-columns` - В поле columns передано некорректное значение
- `/column-not-found` - В поле columns передано несуществующее поля датасета
- `/incompatible-column-category` - В поле columns переданы поля с несовместимыми категориями
- `/multiple-entities` - Сущность датасета из поля code не совпадает с целевой сущностью из поля entity_rk

3.19.5 /fat_split

- `/column-not-found` - Column not found in dataset

3.19.6 /files

- `/invalid-extension` - Invalid file extension in dataframe file
- `/large-file` - The file is too large
- `/invalid-file-path` - Invalid file path
- `/file-broken` - File broken
- `/delimiter-undetected` - Can't detect delimiter for csv file
- `/header-required` - Can't detect header if dataset
- `/count-file-object-exceeded` - Can't create file-like object. Limit exceeded
- `/count-object-closed` - File object closed
- `/file-cannot-be-uploaded` - File can't be uploaded
- `/file-cannot-be-read` - File {} can't be read. Bad encoding

3.19.7 /general

- `/invalid-schema` - Ошибка валидации входных параметров. Либо request body не соответствует схеме, либо параметры в url имеют неправильный тип.
- `/unauthorized` - Ошибка авторизации в API, проблемы с токеном.
- `/forbidden` - У пользователя нет роли доступа.
- `/object-not-found` - Объект с таким rk не найден.
- `/method-not-allowed` - Вызов метода в текущих условиях невозможен, например если производится попытка удалить объект, который используется в других объектах.
- `/object-already-exists` - Объект такого типа и с таким именем уже существует.
- `/internal-server-error` - Неожиданная ошибка сервера. Все доступные детали в самом сообщении.
- `/not-implemented` - Метод не реализован
- `/invalid-name-mask` - Некорректный формат названия
- `/airflow-exception` - Ошибка обращения к Airflow, описание в detail
- `/metastore-unavailable` - Не удалось подключиться к Metastore
- `/metastore-session-refused` - Не удалось установить сессию в рамках соединения с Metastore
- `/legacy-exception` - Тип ошибки для совместимости с функционалом до 1.0

3.19.8 /jobs

- `/invalid-options` - Некорректный формат словаря job_options
- `/job-already-running` - Задание уже запущено, невозможно запустить задание повторно пока предыдущий запуск не завершится
- `/task-not-found` - В задании указана ссылка на несуществующий узел
- `/unexpected-task` - В задании указан узел с неизвестным типом
- `/process-error` - Failed to process job

3.19.9 /ref

- `/cannot-recreate-reference-table` - Не получилось пересоздать справочник. Детали в самом сообщении.
- `/table-is-not-empty` - Ошибка инициирущего наполнения справочника: таблица справочника уже содержит какие-то данные.

3.19.10 /sources

- `/empty-database-name` - Database name can't be empty
- `/empty-schemas-list` - Schemas list can't be empty
- `/invalid-api-url` - API must implement /tables and /columns endpoints
- `/default-fs-type-not-found` - Default fs_type for {}.{} doesn't exist in ref_type_default

4. Schemas

4.1 API

4.1.1 %catalogue_v1

```
{
    *"ref_fs_types": [
        {
            *"fs_type": str,
            "description": str,
            *"py_type": str
        }
    ],
    *"ref_db_types": [
        {
            *"db_name": str,
            *"fs_type": str,
            *"db_type": str
        }
    ],
    *"ref_type_indicators": [
        {
            *"fs_type": str,
            *"indicator": str
        }
    ],
    *"ref_type_defaults": [
        {
            *"db_name": str,
            *"db_type": str,
            *"default_fs_type": str
        }
    ],
    *"features": [
        {
            *"name": str,
            "description": str
        }
    ],
    *"keys": [
        {
            *"name": str,
            "description": str,
            *"data_type_rk": str  # тип данных "{fs_type}"
        }
    ],
    *"entities": [
        {
            *"name": str,
            "description": str,
            *"entity_key_rk": [str],  # список имен ключей ["{key_name}"]
            "tags": [str],
            "options": {
                "storage_key": list[str],  # список имен ключей ["{key_name}"]
                "fat_registry": str  # реестр сущности "{loader_name}"
            }
        }
    ],
    *"loaders": [
        {
            *"name": str,
            "description": str,
            *"loader_kind": "external",
            *"loader_type": "hot" | "entity_link",
            "entity_rk": str,  # имя сущности "{entity_name}"
            *"loader_table": {
                *"datasource_id": -1,  # константа -1
                # имя схемы или параметр
                *"src_schema_name": "{work}" | "{store}" | "{dataset}" | str,
                *"src_table_name": str
            },
            "from_dttm_stg_col_nm": str,
            "to_dttm_stg_col_nm": str,
            "feature_table": {
                "granularity": int,
                "granularity_type": str,
                *"kind": str
            },
            "entity_map": [
                {
                  *"entity_key_rk": str,  # имя ключа сущности "{key_name}",
                  *"stg_column_name": str
                }
            ],
```

```
            "feature_map": [
                {
                    *"feature_column": {
                        *"feature_rk": str,  # имя переменной "{feature_name}"
                        *"data_type_rk": str,  # тип данных "{fs_type}"
                        *"name": str,
                        *"kind": str,
                        "description": str
                    },
                    *"stg_column_name": str
                }
            ],
            "entity_link_map": {
                *"kind": str,
                *"parent": {
                    *"entity_rk": str,  # имя сущности "{entity_name}"
                    *"entity_key": [
                        {
                            *"entity_key": str,  # имя ключа сущности "{key_name}"
                            *"stg_column_name": str
                        }
                    ]
                },
                *"child": {
                    *"entity_rk": str,  # имя сущности "{entity_name}"
                    *"entity_key": [
                        {
                            *"entity_key": str,  # имя ключа сущности "{key_name}"
                            *"stg_column_name": str
                        }
                    ]
                }
            }
        }
    ],
    *"permissions": [
        {
            *"object_rk": str,  # имя загрузчика "{loader_name}"
            *"object_type": "loader",
            *"username": str,
            *"permission": str
        }
    ],
    *"links": [
        {
            "loader_rk": str,  # имя загрузчика "{loader_name}"
            *"name": str,
            "description": str,
            *"parent_entity_rk": str,  # имя родительской сущности "{entity_name}"
            *"child_entity_rk": str,  # имя дочерней сущности "{entity_name}"
            *"kind": "1:1" | "1:M" | "M:N"
        }
    ],
    *"ref_link_chains": [
        {
            *"name": str,
            *"chain": [str],  # цепочка с именами связей ["{link_name}"]
            *"visible_flg": bool,
            "description": str
        }
    ],
    "version": 1
}
```

4.1.2 %catalogue_v2

```
{
    *"ref_fs_types": [
        {
            *"fs_type": str,
            "description": str,
            *"py_type": str
        }
    ],
    *"ref_db_types": [
        {
            *"db_name": str,
            *"fs_type": str,
            *"db_type": str
        }
    ],
    *"ref_type_indicators": [
        {
            *"fs_type": str,
            *"indicator": str
        }
    ],
    *"ref_type_defaults": [
        {
            *"db_name": str,
            *"db_type": str,
            *"default_fs_type": str
        }
    ],
    *"features": [
        {
            *"name": str,
            "description": str
        }
    ],
    *"keys": [
        {
            *"name": str,
            "description": str,
            *"data_type_rk": str  # тип данных "{fs_type}"
        }
    ],
    *"entities": [
        {
            *"name": str,
            "description": str,
            *"entity_key_rk": [str],  # список имен ключей ["{key_name}"]
            "tags": [str],
            "options": {
                "storage_key": list[str],  # список имен ключей ["{key_name}"]
                "fat_registry": str  # реестр сущности "{loader_name}"
            }
        }
    ],
    *"loaders": [
        {
            *"name": str,
            "description": str,
            *"loader_kind": "external",
            *"loader_type": "hot" | "entity_link",
            "entity_rk": str,  # имя сущности "{entity_name}"
            *"loader_table": {
                *"datasource_id": -1,  # константа -1
                # имя схемы или параметр
                *"src_schema_name": "{work}" | "{store}" | "{dataset}" | str,
                *"src_table_name": str
            },
            "from_dttm_stg_col_nm": str,
            "to_dttm_stg_col_nm": str,
            "feature_table": {
                "granularity": int,
                "granularity_type": str,
                *"kind": str
            },
            "entity_map": [
                {
                  *"entity_key_rk": str,  # имя ключа сущности "{key_name}",
                  *"stg_column_name": str
                }
            ],
            "feature_map": [
                {
                    *"feature_column": {
                        *"feature_rk": str,  # имя переменной "{feature_name}"
                        *"data_type_rk": str,  # тип данных "{fs_type}"
                        *"name": str,
                        *"kind": str,
                        "description": str
                    },
                    *"stg_column_name": str
```

```
                }
            ],
            "entity_link_map": {
                *"kind": str,
                *"parent": {
                    *"entity_rk": str,  # имя сущности "{entity_name}"
                    *"entity_key": [
                        {
                            *"entity_key": str,  # имя ключа сущности "{key_name}"
                            *"stg_column_name": str
                        }
                    ]
                },
                *"child": {
                    *"entity_rk": str,  # имя сущности "{entity_name}"
                    *"entity_key": [
                        {
                            *"entity_key": str,  # имя ключа сущности "{key_name}"
                            *"stg_column_name": str
                        }
                    ]
                }
            }
        }
    ],
    *"permissions": [
        {
            *"object_rk": str,  # имя загрузчика "{loader_name}"
            *"object_type": "dataset",
            *"username": str,
            *"permission": str
        }
    ],
    *"links": [  # только автосвязи
        {
            *"name": str,
            "description": str,
            *"parent_entity_rk": str,  # имя родительской сущности "{entity_name}"
            *"child_entity_rk": str,  # имя дочерней сущности "{entity_name}"
            *"kind": "1:1" | "1:M" | "M:N"
        }
    ],
    *"ref_link_chains": [
        {
            *"name": str,
            *"chain": [str],  # цепочка с именами связей ["{link_name}"]
            *"visible_flg": bool,
            "description": str
        }
    ],
    *"version": 2
}
```

4.1.3 %catalogue_v3

```
{
    *"ref_fs_types": [
        {
            *"fs_type": str,
            "description": str,
            *"py_type": str
        }
    ],
    *"ref_db_types": [
        {
            *"db_name": str,
            *"fs_type": str,
            *"db_type": str
        }
    ],
    *"ref_type_indicators": [
        {
            *"fs_type": str,
            *"indicator": str
        }
    ],
    *"ref_type_defaults": [
        {
            *"db_name": str,
            *"db_type": str,
            *"default_fs_type": str
        }
    ],
    *"features": [
        {
            *"name": str,
            "description": str
        }
    ],
    *"keys": [
        {
            *"name": str,
            "description": str,
            *"data_type_rk": str  # тип данных "{fs_type}"
        }
    ],
    *"entities": [
        {
            *"name": str,
            "description": str,
            *"entity_key_rk": [str],  # список имен ключей ["{key_name}"]
            "tags": [str],
            "options": {
                "storage_key": list[str],  # список имен ключей ["{key_name}"]
                "fat_registry": str  # реестр сущности "{loader_name}"
            }
        }
    ],
    *"loaders": [
        {
            *"name": str,
            "description": str,
            *"loader_kind": "external",
            *"loader_type": "hot" | "entity_link",
            "entity_rk": str,  # имя сущности "{entity_name}"
            *"loader_table": {
                *"datasource_id": -1,  # константа -1
                # имя схемы или параметр
                *"src_schema_name": "{work}" | "{store}" | "{dataset}" | str,
                *"src_table_name": str
            },
            "from_dttm_stg_col_nm": str,
            "to_dttm_stg_col_nm": str,
            "feature_table": {
                "granularity": int,
                "granularity_type": str,
                *"kind": str
            },
            "entity_map": [
                {
                  *"entity_key_rk": str,  # имя ключа сущности "{key_name}",
                  *"stg_column_name": str
                }
            ],
            "feature_map": [
                {
                    *"feature_column": {
                        *"feature_rk": str,  # имя переменной "{feature_name}"
                        *"data_type_rk": str,  # тип данных "{fs_type}"
                        *"name": str,
                        *"kind": str,
                        "description": str
                    },
                    *"stg_column_name": str
```

```
                }
            ],
            "entity_link_map": {
                *"kind": str,
                *"parent": {
                    *"entity_rk": str,  # имя сущности "{entity_name}"
                    *"entity_key": [
                        {
                            *"entity_key": str,  # имя ключа сущности "{key_name}"
                            *"stg_column_name": str
                        }
                    ]
                },
                *"child": {
                    *"entity_rk": str,  # имя сущности "{entity_name}"
                    *"entity_key": [
                        {
                            *"entity_key": str,  # имя ключа сущности "{key_name}"
                            *"stg_column_name": str
                        }
                    ]
                }
            }
        }
    ],
    *"permissions": [
        {
            *"object_rk": str,  # имя загрузчика "{loader_name}"
            *"object_type": "dataset",
            *"username": str,
            *"permission": str
        }
    ],
    *"links": [  # только автосвязи
        {
            *"name": str,
            "description": str,
            *"parent_entity_rk": str,  # имя родительской сущности "{entity_name}"
            *"child_entity_rk": str,  # имя дочерней сущности "{entity_name}"
            *"kind": "1:1" | "1:M" | "M:N"
        }
    ],
    *"ref_link_chains": [
        {
            *"name": str,
            *"chain": [str],  # цепочка с именами связей ["{link_name}"]
            *"visible_flg": bool,
            "description": str
        }
    ],
    *"version": 3
}
```

4.1.4 %catalogue_v4

```
{
    *"ref_fs_types": [
        {
            *"fs_type": str,
            "description": str,
            *"py_type": str
        }
    ],
    *"ref_db_types": [
        {
            *"db_name": str,
            *"fs_type": str,
            *"db_type": str
        }
    ],
    *"ref_type_indicators": [
        {
            *"fs_type": str,
            *"indicator": str
        }
    ],
    *"ref_type_defaults": [
        {
            *"db_name": str,
            *"db_type": str,
            *"default_fs_type": str
        }
    ],
    *"features": [
        {
            *"name": str,
            "description": str
        }
    ],
    *"keys": [
        {
            *"name": str,
            "description": str,
            *"data_type_rk": str  # тип данных "{fs_type}"
        }
    ],
    *"entities": [
        {
            *"name": str,
            "description": str,
            *"entity_key_rk": [str],  # список имен ключей ["{key_name}"]
            "tags": [str],
            "options": {
                "storage_key": list[str],  # список имен ключей ["{key_name}"]
                "fat_registry": str  # реестр сущности "{dataset_name}"
            }
            "rank": int
        }
    ],
    *"datasets": [
        {
            *"name": str,
            "desc": str,
            *"origin": (
                "loader/final/external" | "loader/common_preagg/external"
                | "loader/trans_preagg/external" | "loader/link/external" | "loader/final/file"
                | "loader/common_preagg/file" | "loader/trans_preagg/file" | "loader/link/file" | "link/auto"
            ),
            *"db_schema": "{work}" | "{store}" | "{dataset}" | str,
            *"db_table": str,
            "db_date": str,
            "db_date_end": str,
            "config": {  # справочная информация, которая влияет только на вывод в get-методах api v2
                "LOAD_MODE": str,
                "LOADER_FILE": {
                    "FILE_PATH": str,
                    "FILE_TYPE": str,
                    "DELIMITER": str,
                    "HEADER_FLG": bool,
                },
                "LOADER_TABLE": {
                    "DATASOURCE_ID": int,
                    "SRC_SCHEMA_NAME": "{work}" | "{store}" | "{dataset}" | str,
                    "SRC_TABLE_NAME": str,
                },
                "FEATURE_TABLE_HIST": {
                    "GRANULARITY": int,
                    "GRANULARITY_TYPE": str,
                },
                "MAPPING": dict[str, str],  # mapping {target_column: source_column}
            },
            *"features": [
                {
                    *"name": str,
```

```
                    "desc": str,
                    *"db_column": str,
                    *"fs_type": str,  # тип данных "{fs_type}"
                    "tags": list[str],
                    "rank": int
                }
            ],
            *"instances": [
                {
                    *"name": str,
                    "desc": str,
                    *"unique": bool,
                    *"entity_rk": str,  # имя сущности "{entity_name}",
                    *"columns": [
                        {
                            *"key_rk": str,  # имя ключа "{key_name}"
                            *"db_column": str
                        }
                    ]
                    "rank": int
                }
            ]
            "rank": int,
            "tags": list[str]
        }
    ],
    *"jobs": [
        {
            *"datasets": list[str],  # список имен датасетов ["{dataset_name}"]
            "job_options": <%job_options>,
            *"body": {  # оригинальный граф задач
                *"tasks": [<%job_task>, ...]  # список задач
                *"graph": {                   # зависимости между задачами
                    "<task id>": [<task id>, ...],
                    ...
                }
            },
            "desc": str,
            "resource": str,
            "dispose": "on_success" | "manual" | "always" | "smart"
        }
    ],
    *"permissions": [
        {
            *"object_rk": str,  # имя датасета "{dataset_name}"
            *"object_type": "dataset",
            *"username": str,
            *"permission": "no_access" | "read" | "full"
        }
    ],
    *"ref_link_chains": [
        {
            *"name": str,
            *"chain": [str],  # цепочка с именами связей ["{dataset_name}"]
            *"visible_flg": bool,
            "description": str
        }
    ],
    *"version": 4
}
```

- %job_options
- %job_task

4.1.5 %import_task

```
{
    *"TASK_ID": uuid,           # Идентификатор задачи в экзекьюторе
    *"LAST_FLG": bool,          # Признак последнией задачи импорта
    *"START_DTTM": datetime,    # Дата-время начала импорта
    "FINISH_DTTM": datetime,    # Дата-время завершения импорта
    *"STATUS": enum,             # Статус задачи: "new" | "queued" | "running" | "success" | "failed"
    "DETAIL": str,              # Информация об ошибке
    *"CREATED_USER": str,       # Пользователь, создавший запись
    *"CREATED_DTTM": datetime   # Дата-время создания записи
    *"UPDATED_DTTM": datetime   # Дата-время последнего обновления записи
}
```

4.1.6 %loader_entity_link

```
{
    *"NAME": str,  # название загрузчика, уникально
    "DESCRIPTION": str,  # описание загрузчика
    *"LOADER_KIND": str,  # возможные источники: ['file', 'table', 'external', 'sql']
    *"LOADER_TYPE": str,  # назначение загрузчика: ['hot' (для фичей), 'keys', 'entity_link]'
    *"ENTITY_RK": int,  # суррогатный ключ сущности
    *"LOAD_MODE": str,  # режим загрузки данных: ['full', 'inc']
    "LOADER_TABLE": {  # описание источника table/external (заполняется для LOADER_KIND=table/external)
        "DATASOURCE_ID": int,  # ключ источника (доступно только -1)
        "SRC_SCHEMA_NAME": str,  # название схемы таблицы источника
        "SRC_TABLE_NAME": str  # название таблицы источника
    },
    "LOADER_SQL": {  # описание источника table (заполняется для LOADER_KIND=table)
        "DATASOURCE_ID": int,  # ключ источника (доступно только -1)
        "SQL_PATH": str  # путь до файла в хранилище
    },
    "FILE_DESC": {  # описание источника file (заполняется для LOADER_KIND=file)
        "FILE_PATH": str,  # путь до файла в хранилище
        "DELIMITER": str,  # разделитель только для FILE_TYPE=csv,
                           # автоматически определяется по первой строке, если не передан
        "FILE_TYPE": str  # тип файла (csv, xlsx)
    },
    "FROM_DTTM_STG_COL_NM": str,  # столбец в источнике с датой начала периода актуальности данных
    "FROM_DTTM_VALUE": str,  # значение даты начала периода актуальности данных (1.)
    "TO_DTTM_STG_COL_NM": str,  # столбец в источнике с датой окончания периода актуальности данных
    "ENTITY_LINK_MAP": {  # маппинг полей таблицы-источника на связанные сущности
    "SCHEDULE": str,    # расписание в формате крон-строки
    "TIMEZONE": float,  # разница с UTC, от -12.0 до 14.0
                        # дробная часть указывается в долях часа (например, 3.5 для UTC+3:30)
    *"KIND": str,  # мощность связи "1:1" / "1:M" / "M:N"
    *"PARENT": {  # маппинг сущности-родителя
      *"ENTITY_RK": int,  # суррогатный ключ сущности-родителя
      *"ENTITY_KEY": [  # маппинг полей таблицы-источника на ключи родительской сущности
        {
          *"ENTITY_KEY_RK": int,  # суррогатный ключ ключа сущности
          *"STG_COLUMN_NAME": str  # название столбца в источнике, соответствующее выбранному ключу
        }
      ]
    },
    *"CHILD": {  # маппинг сущности-потомка
      *"ENTITY_RK": int,  # суррогатный ключ сущности-потомка
      *"ENTITY_KEY": [  # маппинг полей таблицы-источника на ключи дочерней сущности
        {
          *"ENTITY_KEY_RK": int,  # суррогатный ключ ключа сущности
          *"STG_COLUMN_NAME": str  # название столбца в источнике, соответствующее выбранному ключу
        }
      ]
    }
    }
}
```

Комментарии к заполнению полей

1. FROM_DTTM_STG_COL_NM / FROM_DTTM_VALUE / TO_DTTM_STG_COL_NM

- LOADER_KIND=table/file/sql:
- TO_DTTM_STG_COL_NM - запрещено
- FROM_DTTM_STG_COL_NM / FROM_DTTM_VALUE - заполняется либо одно из двух, либо ни одно (в таком случае подставится значение текущей даты в FROM_DTTM_VALUE)
- LOADER_KIND=external:
- FROM_DTTM_STG_COL_NM / FROM_DTTM_VALUE - заполняется либо одно из двух, либо ни одно
- TO_DTTM_STG_COL_NM - заполняется опционально, но только при наличии заполненного FROM_DTTM_STG_COL_NM или FROM_DTTM_VALUE

4.1.7 %loader_features

```
{
    *"NAME": str,  # название загрузчика, уникально
    "DESCRIPTION": str,  # описание загрузчика
    *"LOADER_KIND": str,  # возможные источники: ['file', 'table', 'external', 'sql']
    *"LOADER_TYPE": str,  # назначение загрузчика: ['hot' (для фичей), 'keys', 'entity_link]'
    *"ENTITY_RK": int,  # суррогатный ключ сущности
    *"LOAD_MODE": str,  # режим загрузки данных: ['full', 'replace']
    "LOADER_TABLE": {  # описание источника table/external (заполняется для LOADER_KIND=table/external)
        "DATASOURCE_ID": int,  # ключ источника (доступно только -1)
        "SRC_SCHEMA_NAME": str,  # название схемы таблицы источника
        "SRC_TABLE_NAME": str  # название таблицы источника
    },
    "LOADER_SQL": {  # описание источника table (заполняется для LOADER_KIND=table)
        "DATASOURCE_ID": int,  # ключ источника (доступно только -1)
        "SQL_PATH": str  # путь до файла в хранилище
    },
    "FILE_DESC": {  # описание источника file (заполняется для LOADER_KIND=file)
        *"FILE_PATH": str,  # путь до файла в хранилище
        "HEADER_FLG": bool, # флаг наличия названий столбцов в первой строке файла, всегда true
        "DELIMITER": str,  # разделитель только для FILE_TYPE=csv,
                           # автоматически определяется по первой строке, если не передан
        "FILE_TYPE": str  # тип файла (csv, xlsx)
    },
    "FROM_DTTM_STG_COL_NM": str,  # столбец в источнике с датой начала периода актуальности данных
    "FROM_DTTM_VALUE": str,  # значение даты начала периода актуальности данных
    "TO_DTTM_STG_COL_NM": str,  # столбец в источнике с датой окончания периода актуальности данных
    "SCHEDULE": str,    # расписание в формате крон-строки
    "TIMEZONE": float,  # разница с UTC, от -12.0 до 14.0
                        # дробная часть указывается в долях часа (например, 3.5 для UTC+3:30)
    *"FEATURE_TABLE": {  # описание фичей
        "GRANULARITY": int,  # значение гранулярности
        "GRANULARITY_TYPE": str,  # тип гранулярности: ['hour', 'day', 'week', 'month', 'year', 'calendar_month']
        *"KIND": str  # тип фичей: ['final', 'trans_preagg', 'common_preagg']
    },
    *"ENTITY_MAP": [  # маппинг ключей сущности
        {
            *"ENTITY_KEY_RK": int,  # суррогатный ключ ключа сущности
            *"STG_COLUMN_NAME": str  # название столбца в источнике, соответствующее выбранному ключу
        }
    ],
    *"FEATURE_MAP": [  # маппинг фичей
        {
            "FEATURE_COLUMN": {  # конфигурация версии фичи
                *"FEATURE_RK": int,  # суррогатный ключ фичи
                *"DATA_TYPE_RK": int,  # суррогатный ключ типа данных версии фичи
                *"NAME": str,  # название версии фичи
                *"KIND": str,  # тип версии, доступные значения: ['feature']
                "DESCRIPTION": str  # описание версии фичи
                "TAGS": list[str]  # теги версии фичи
            },
            *"STG_COLUMN_NAME": str  # название столбца в источнике, соответствующее выбранной фиче
        }
    ]
}
```

Комментарии к заполнению полей

1. FROM_DTTM_STG_COL_NM / FROM_DTTM_VALUE / TO_DTTM_STG_COL_NM

- LOADER_KIND=table/file/sql:
- TO_DTTM_STG_COL_NM - запрещено
- FROM_DTTM_STG_COL_NM / FROM_DTTM_VALUE - заполняется либо одно из двух, либо ни одно (в таком случае подставится значение текущей даты в FROM_DTTM_VALUE)
- LOADER_KIND=external:
- FEATURE_TABLE.KIND=final/common_preagg:
- FROM_DTTM_STG_COL_NM / FROM_DTTM_VALUE - заполняется либо одно из двух, либо ни одно
- TO_DTTM_STG_COL_NM - заполняется опционально, но только при наличии заполненного FROM_DTTM_STG_COL_NM или FROM_DTTM_VALUE
- FEATURE_TABLE.KIND=trans_preagg:
- FROM_DTTM_STG_COL_NM / FROM_DTTM_VALUE - заполняется только одно из двух обязательно
- TO_DTTM_STG_COL_NM - заполняется обязательно

2. LOADER_KIND/LOADER_TYPE:

- Комбинация LOADER_TYPE='keys' + LOADER_KIND='sql' недоступна

4.1.8 %loader_keys

```
{
    *"NAME": str,  # название загрузчика, уникально
    "DESCRIPTION": str,  # описание загрузчика
    *"LOADER_KIND": str,  # возможные источники: ['file', 'table', 'external']
    *"LOADER_TYPE": str,  # назначение загрузчика: 'keys'
    *"ENTITY_RK": int,  # суррогатный ключ сущности
    "LOADER_TABLE": {  # описание источника table/external (заполняется для LOADER_KIND=table/external)
        "DATASOURCE_ID": int,  # ключ источника (доступно только -1)
        "SRC_SCHEMA_NAME": str,  # название схемы таблицы источника
        "SRC_TABLE_NAME": str  # название таблицы источника
    },
    "FILE_DESC": {  # описание источника file (заполняется для LOADER_KIND=file)
        "FILE_PATH": str,  # путь до файла в хранилище
        "DELIMITER": str,  # разделитель только для FILE_TYPE=csv
                           # автоматически определяется по первой строке, если не передан
        "FILE_TYPE": str  # тип файла (csv, xlsx)
    },
    "SCHEDULE": str,    # расписание в формате крон-строки
    "TIMEZONE": float,  # разница с UTC, от -12.0 до 14.0
                        # дробная часть указывается в долях часа (например, 3.5 для UTC+3:30)
    *"ENTITY_MAP": [  # маппинг ключей сущности
        {
            *"ENTITY_KEY_RK": int,  # суррогатный ключ ключа сущности
            *"STG_COLUMN_NAME": str  # название столбца в источнике, соответствующее выбранному ключу
        }
    ]
}
```

Комментарии к заполнению полей

ENTITY_MAP: для источника external поля ключей (`STG_COLUMN_NAME`) в таблице-источнике должны называться точно так же, как и сами ключи.

4.2 API 1.0

4.2.1 OpenAPI Schemas

Схема ответа при коде отличном от 2xx всегда одна: %failure

В описании схем используются следующие модификаторы для атрибутов:

```
{
    *"required": обязательный атрибут
    :"readonly": автогенерируемый атрибут, игнорируется в POST запросах, всегда заполнен в get запросах
    ~"semiauto": если не заполнен в POST запросе, то сгенерируется автоматически.
    +"expanded": атрибут заполняется со связанных сущностей, если в GET запрос передан expanded=true
}
```

В *%patch схемах null значения допустимы для _необязательных* параметров. В таком случае значение параметра будет выставлено в null.

Если в %_patch схеме параметр не передается, то он не будет изменен.

4.2.2 %automap_file

```
{
   *"path": str,
    "format": "csv" | "xlsx",
    "encoding": str,
    "delimiter": str
    "entities": list[int]
}
```

4.2.3 %automap_table

```
{
   *"db_schema": str,       # имя схемы в БД
   *"db_table": str,        # имя таблицы в схеме
    "entities": list[int],  # список сущностей
}
```

4.2.4 %column

```
{
   :"column_num" int,  # порядковый номер столбца
   :"db_column": str,  # имя столбца
   :"db_type": str,    # имя типа в БД
   :"fs_type": str     # тип данных по-умолчанию
}
```

4.2.5 %dataset

```
{
   :"dataset_rk": int,       # идентификатор датасета
    "source_rk": int,        # источник датасета
    "origin": str,           # происхождение датасета
    "virtual": bool,         # признак виртуального датасета, default=False
    "name": str,             # название датасета, default=db_table
   *"db_schema": str,        # имя схемы в БД
   *"db_table": str,         # имя таблицы в схеме
    "db_date": str,          # колонка, содержащая дату актуальности
    "db_date_end": str,      # колонка, содержащая дату окончания актуальности
    "features": [<%feature>, ...],     # маппинг на переменные
    "instances": [<%instance>, ...],   # маппинг на сущности
    "config": dict,          # конфигурация датасета
    "rank": int            # ранг датасета. Используется для сортировки в get-list-методах
    "tags": [str, ...],    # список тегов датасета
   :"created_dttm": str,  # дата создания
   :"updated_dttm": str,   # дата последнего изменения
   :"author": str,        # пользователь, создавший датасет
   :"permission": enum    # PermissionLevelEnum # уровень доступа пользователя
}
```

4.2.6 %dataset_health

```
{
   :"source": {       # статус
      "state": "ok" | "warning" | "error",
      "detail": str
    },
   :"db_date": {      # статус перемменой "дата актуальности"
      "state": "ok" | "warning" | "error",
      "detail": str
    },
   :"db_date_end": {  # статус перемменой "дата окончания актуальности"
      "state": "ok" | "warning" | "error",
      "detail": str
    },
   :"features": {     # статус описанных переменных в датасете
      <feature_rk>: {
        "name": str,
        "state": "ok" | "warning" | "error",
        "detail": str  # например колонка не существует
      },
      ...
    },
   :"instances": {    # статус экземпляров сущности
      <instance_rk>: {
        "name": str,
        "state": "ok" | "warning" | "error",
        "detail": str  # например сущность не существует или не хватает ключей
      },
      ...
    }
}
```

4.2.7 dataset_job_summary

```
{
   :"job_rk": int,                  # идентификатор задания
   :"dataset_rk": int,              # идентификатор датасета
   :"role": str,                    # роль джоба для датасета
   :"desc": str,                    # описание задания
   :"start_dttm": str,              # дата последнего запуска
   :"finish_dttm": str,             # дата завершения последнего запуска
   :"state": str,                   # статус последнего запуска
   :"detail": str,                  # описание ошибки, если state=failed
   :"resource": str,                # ссылка на создаваемый ресурс
   :"schedule": str                 # расписание запуска
   :"dispose": enum,                # статегия автоматического удаления job-а с запусками
   :"created_dttm": str,            # дата создания задания
   :"udated_dttm": str,             # дата последнего изменения задание
   :"author": str                   # пользователь, создавший задание
   :"permission": enum              # уровень доступа к датасету
}
```

4.2.8 %dataset_patch

```
{
    "name": str,         # название датасета
    "desc": str,         # описание датасета
    "db_schema": str,    # имя схемы в БД
    "db_table": str,     # имя таблицы в схеме
    "db_date": str,      # колонка, содержащая дату актуальности
    "db_date_end": str,  # колонка, содержащая дату окончания актуальности
    "features": [<%feature>, ...],     # маппинг на переменные
    "instances": [<%instances>, ...],  # маппинг на сущности
    "tags": [str, ...],  # список тегов датасета
    "rank": int          # ранг датасета. Используется для сортировки в get-list-методах
}
```

4.2.9 %dataset_summary

```
{
   :"dataset_rk": int,       # идентификатор датасета
   :"origin": str,           # происхождение датасета
   :"virtual": bool,         # признак виртуального датасета
   :"source_rk": int,        # источник датасета
   :"name": str,             # название датасета
   :"desc": str,             # описание датасета
   :"db_schema": str,        # имя схемы в БД
   :"db_table": str,         # имя таблицы в схеме
   :"db_date": str,          # колонка, содержащая дату актуальности
   :"db_date_end": str,      # колонка, содержащая дату окончания актуальности
   :"config": dict,          # конфигурация датасета
   :"rank": int              # ранг датасета. Используется для сортировки в get-list-методах
   :"tags": [str, ...],      # список тегов датасета
   :"created_dttm": str,     # дата создания
   :"updated_dttm": str,     # дата последнего изменения
   :"author": str,           # пользователь, создавший датасет
   :"permission": enum       # PermissionLevelEnum # уровень доступа пользователя
}
```

4.2.10 %link

```
{
    :"link_rk": int,       # Идентификатор связи
    *"dataset_rk": int,    # Идентификатор дочернего датасета
    *"parent_dataset_rk": int,  # Идентификатор родительского датасета
    *"name": str,          # Название связи
     "desc": str,          # Описание связи
    :"author": str,        # Пользователь, создавший связь
    :"created_dttm": datetime,  # Дата создания
    :"updated_dttm": datetime   # Дата последнего изменения
}
```

4.2.11 %link_patch

```
{
    "dataset_rk": int,    # Идентификатор дочернего датасета
    "parent_dataset_rk": int,    # Идентификатор родительского датасета
    "name": str,          # Название связи
    "desc": str           # Описание связи
}
```

4.2.12 %entity

```
{
   :"entity_rk": int,              # id зерна
   *"name": str,                   # имя, должно быть уникальным для всех сущностей
    "desc": str,                   # описание
   *"keys": [int, ...],            # набор ключей в сущности
   +"keys_info": [<%key>, ...],    # детальная информация по ключам в сущности
    "tags": [str, ...],            # список тегов
    "options": <%entity_options>,  # опции сущности
    "rank": int                     # ранг сущности. Используется для сортировки в get-list-методах
   :"author": str,        # пользователь
   :"created_dttm": str,  # дата создания
   :"updated_dttm": str   # дата изменения

}
```

4.2.13 %entity_options

```
{
   "fat_registry": int,  # Ссылка на загрузчик реестра сущности
   "storage_key": [int]  # Список key_rk ключей сущности, входящих в ключ распределения
}
```

4.2.14 %entity_patch

```
{
    "name": str,         # имя
    "description": str,  # описание
    "tags": [str, ...],  # теги
    "options": <%entity_options>,  # опции
    "rank": int           # ранг сущности. Используется для сортировки в get-list-методах
}
```

4.2.15 %exec_profile

```
{
    *"name": str,  # имя профиля исполнения
    ~"job_options": <%job_options>,  # глобальные параметры цепочки
    :"created_dttm": str,  # дата создания профиля
    :"updated_dttm": str,  # дата последнего изменения профиля
    :"author": str         # пользователь, создавший профиль
}
```

- %job_options

4.2.16 %exec_profile_patch

```
{
    "name": str,  # имя профиля исполнения
    "job_options": <%job_options>,  # глобальные параметры цепочки
    :"created_dttm": str,  # дата создания профиля
    :"updated_dttm": str,  # дата последнего изменения профиля
    :"author": str         # пользователь, создавший профиль
}
```

- %job_options

4.2.17 %failure

```
{
   *"type": str,       # URI [RFC3986] ошибки
   *"title": str,      # короткое читаемое описание типа ошибки
   *"status": int,     # http статус-код
    "detail": str,     # читаемое описание конкретной ошибки
    "traceback": str,  # traceback
    "instance": str,   # URI на описание ошибки, может не нести дополнительной полезной информации
}
```

example

```
{
    "type": "/error-codes/invalid-link-chain",
    "title": "Invalid entity link chain",
    "status": 422,
    "detail": "'!el_1!el_2': First entity must contain entity of main dataset",
    "traceback": "...",
    "instance": "/entities/7"
}
```

4.2.18 %feature

```
{
   :"feature_rk": int,   # идентификатор переменной
   :"dataset_rk": int,   # идентификатор датасета
   *"name": str,         # имя переменной
    "desc": str,         # описание переменной
    "tags": [str, ...],  # список тегов
   *"db_column": str,    # имя столбца в таблице в БД
   :"role": enum,        # "feature" | "key" | "date" | "date_end" # роль в датасете
   *"fs_type": str,              # идентификатор типа данных
   +"fs_type_info": <%fs_type>,   # детальная информация по типу данных
    "rank": int            # ранг переменной. Используется для сортировки в get-list-методах
   :"author": str,        # пользователь
   :"created_dttm": str,  # дата создания
   :"updated_dttm": str,  # дата изменения
   :"permission": enum    # PermissionLevelEnum # уровень доступа пользователя
}
```

4.2.19 %feature_patch

```
{
    "name": str,     # имя переменной
    "desc": str,     # описание переменной
    "fs_type": str,  # тип данных
    "rank": int      # ранг переменной. Используется для сортировки в get-list-методах
}
```

4.2.20 %feature_summary

```
{
   :"feature_rk": int,   # идентификатор переменной
   :"dataset_rk": int,   # идентификатор датасета
   :"dataset_name": str, # имя датасета
   :"name": str,         # имя переменной
   :"desc": str,         # описание переменной
   :"db_schema": str,    # имя схемы в БД
   :"db_table": str,     # имя таблицы в БД
   :"db_column": str,    # имя столбца в таблице в БД
   :"role": enum,        # "feature" | "key" | "date" | "date_end" # роль в датасете
   :"fs_type": str,      # идентификатор типа данных
   :"rank": int           # ранг переменной. Используется для сортировки в get-list-методах
   :"author": str,        # пользователь
   :"created_dttm": str,  # дата создания
   :"updated_dttm": str,  # дата изменения
   :"permission": enum    # PermissionLevelEnum # уровень доступа пользователя
}
```

4.2.21 %file

```
{
    :"path": str,              # путь к файлу
    :"size": str,              # размер в байтах
    :"extension": str,         # расширение файла
    :"delimiter": str,         # разделитель в файле (только для csv)
    :"columns": list[%column]  # Список стобцов
}
```

- %column

4.2.22 %file_handler

```
{
    "class": str,       # класс хендлера
    "level": str,       # уровень логирования "DEBUG" | "INFO" | "WARNING" | "ERROR" | "CRITICAL"
    "formatter": str,   # название форматтера из секции formatters
    "filename ": str,   # абсолютный путь до файла с логом
    "when": str,        # когда осуществляется закрытие файла и открытие нового (по умолчанию midnight)
    "backupCount": int  # количество файлов с логами
}
```

4.2.23 %health

```
{
    "metastore": {
        "uri": str,       # адрес компонента
        "state": enum,    # "up" | "down" # статус
        "revision": str,  # имя alembic ревизии
        "detail": str     # описание ошибки
    },
    "executor": {
        "state": enum,              # "up" | "down" # статус
        "broker_url": str,          # адрес брокера сообщений
        "result_backend_url": str,  # адрес движка, где celery хранит результаты задач
        "queue": str,               # имя очереди в брокере
        "stats": {
            "{hostname}": { stat info },  # статистики воркера Celery в формате as-is
            ...
        },
        "detail": str               # детали ошибки
    },
    "sources": {
        "{source_rk}:{source_name}": {
            "url": str,     # адрес источника
            "connection": {
                "state": enum,  # "ok" | "error" # статус
                "detail": str   # описание ошибки
            }
        }
    },
    "s3": {
        "state": enum,     # "up" | "down" | "off" # статус или отключено
        "url": str,        # адрес компонента
        "auth_type": str,  # способ авторизации в keycloak
        "bucket": str,     # имя бакета
        "detail": str      # описание ошибки
    },
    "app_lock": {                # информация о текущей блокирующей задаче
        "task_id": UUID,         # идентификатор задачи в Celery
        "task_name": str,        # имя задачи (e.g. 'catalogue_importer', 'dataset_deleter')
        "start_dttm": datetime   # дата и время запуска задачи
    }

}
```

4.2.24 %info

```
{
    "version": str  # версия приложения
}
```

4.2.25 %ingest_file

```
{
   *"path": str
   *"format": "csv" | "xlsx"
    "encoding": str  # mandatory for csv
    "delimiter": str  # mandatory for csv
   *"history": "replace" | "insert" | "snapshot"  # default: replace
    "db_date": str
    "db_date_end": str
    "instances": [%instance]  # instance schema from dataset model
    "features": [%feature]  # feature schema from dataset model
}
```

4.2.26 %ingest_sql

```
{
   *"statement": str,  # исполняемая SELECT инструкция на языке SQL
   *"history": "replace" | "insert" | "snapshot"  # default: replace
    "db_date": str
    "db_date_end": str
    "instances": [%instance]  # instance schema from dataset model
    "features": [%feature]  # feature schema from dataset model
}
```

4.2.27 %ingest_table

```
{
   *"db_schema": str,  # имя схемы в БД (откуда)
   *"db_table": str,   # имя таблицы в БД (откуда)
   *"history": "replace" | "insert" | "snapshot"  # default: replace
    "db_date": str
    "db_date_end": str
    "instances": [%instance]  # instance schema from dataset model
    "features": [%feature]  # feature schema from dataset model
}
```

4.2.28 %instance

```
{
   :"instance_rk": int,        # идентификатор экземпляра сущности
   :"dataset_rk": int,         # идентификатор датасета
   *"name": str,               # имя экземпляра сущности
    "desc": str,               # описание экземпляра сущности
    "unique": bool,            # значения экземпляра сущности уникальны в датасете. default: false
   *"entity_rk": int,          # идентификатор сущности
   +"entity_info": <%entity>,  # детальная информация по сущности
   *"columns": [{
      *"db_column": str,      # имя столбца
      *"key_rk": int,         # идентификатор ключа сущности
      :"feature_rk": int,     # индентификатор переменной
      :"instance_rk": int,    # индентификатор экземпляра сущности
      :"author": str,         # пользователь
      :"created_dttm": str,   # дата создания
      :"updated_dttm": str    # дата изменения
   }]
    "rank": int           # ранг экземпляра сущности. Используется для сортировки в get-list-методах
   :"author": str,        # пользователь
   :"created_dttm": str,  # дата создания
   :"updated_dttm": str,  # дата изменения
   :"permission": enum    # PermissionLevelEnum # уровень доступа пользователя
}
```

4.2.29 %instance_patch

```
{
    "name": str,     # имя экземпляра сущности
    "desc": str,     # описание экземпляра сущности
    "unique": bool,  # значения экземпляра сущности уникальны в датасете
    "rank": int      # ранг экземпляра сущности. Используется для сортировки в get-list-методах
}
```

4.2.30 %instance_summary

```
{
   :"instance_rk": int,        # идентификатор экземпляра сущности
   :"dataset_rk": int,         # идентификатор датасета
   :"dataset_name": str        # имя датасета
   :"name": str,               # имя экземпляра сущности
   :"desc": str,               # описание экземпляра сущности
   :"unique": bool,            # значения экземпляра сущности уникальны в датасете
   :"entity_rk": int,          # идентификатор сущности
   :"entity_name": str,        # имя сущности
   :"rank": int                # ранг экземпляра сущности. Используется для сортировки в get-list-методах
   :"author": str,        # пользователь
   :"created_dttm": str,  # дата создания
   :"updated_dttm": str,  # дата изменения
   :"permission": enum    # PermissionLevelEnum # уровень доступа пользователя
}
```

4.2.31 %job

```
{
   :"job_rk": int,       # идентификатор задания
    "desc": str,         # описание задания
    "resource": str,     # ссылка на создаваемый ресурс; например /datasets/123 или /files/link?path=aalfs
   *"body": {  # оригинальный граф задач
       *"tasks": [<%job_task>, ...]  # список задач
       *"graph": {                   # зависимости между задачами
            "<task id>": [<task id>, ...],
            ...
        }
    },
    "dispose": enum,            # "smart" | "on_success" | "always" | "manual"
                                #   default: manual
                                #   статегия автоматического удаления job-а с запусками
                                #   * smart - джоб удалится по внутренним правилам приложения:
                                #       * если создан через POST /jobs, то удаляется вручную
                                #       * если создан через POST /transform и не регистрирует в мете
                                #       датасеты/загрузчики, то удалится при успешном завершении
                                #       * иначе удалится при удалении последнего связанного
                                #       датасета/загрузчика
                                #   * on_success - джоб удалится только при успешном завершении
                                #   * always - джоб удалится после завершения
                                #   * manual - джоб удаляется только вручную
    "job_options": <%job_options>,  # глобальные параметры цепочки
   :"schedule": str,     # расписание запуска в формате cron-строки
   :"created_dttm": str,  # дата создания цепочки
   :"updated_dttm": str,   # дата последнего изменения цепочки
   :"author": str         # пользователь, создавший цепочку
}
```

- %job_options
- %job_task

4.2.32 %job_options

```
{
    "execution_profile": str,  # имя профиля исполнения
    "async_factor": int,       # максимальное количество задач в параллель
    "params": {                # значения переменных для подстановки
        "default_to": str,                # "table" | "view" | "cte" | "subquery"
                                          #   способ материализации по-умолчанию в узле select
        "default_schema": str             # схема в БД по-умолчанию для создание таблиц/вью
        "default_storage_key": [str, ...]  # значение storage_key по-умолчанию в узле select
        "default_analyze": bool,          # значение analyze по-умолчанию в узле select
        "default_create_index": bool,     # значение create_index по умолчанию в узле select
        ...  # любые другие пользовательские переменные
    }
    "advanced": {
        "postgresql": [str, ...],  # специфичные параметры для диалекта postgresql, без ведущего SET
        "greenplum": [str, ...]    # специфичные параметры для диалекта greenplum, без ведущего SET
    }

}
```

4.2.33 %job_patch

```
{
    "desc": str,    # описание цепочки
    "body": {  # оригинальный граф задач
       "tasks": [<%job_task>, ...]  # список задач
       "graph": {                   # зависимости между задачами
            "<task id>": [<task id>, ...],
            ...
        }
    },
    "job_options": <%job_options>  # изменения параметров
}
```

- %job_options

4.2.34 %job_run

```
{
   :"job_rk": int,              # идентификатор задания
   :"run_rk": int,              # идентификатор запуска
   :"source_rk": int,           # идентификатор источника
   :"start_dttm": str,          # дата запуска
   :"finish_dttm": str,         # дата завершения
   :"state": enum,              # "new" | "queued" | "running" | "success" | "failed" # статус
   :"detail": str,              # описание ошибки, если state=failed
   :"resource": str,            # ссылка на создаваемый ресурс; например /datasets/123 или /files/link?path=aalfs
   :"run_info": <%run_info>,      # детальная информация о запуске задания
   ~"job_options": <%job_options>,  # override для параметров цепочки, в тч переменные запуска
                                    # если не заполнено - копируется с job
   :"latest_flg": bool,         # признак последнего запуска
   :"author": str,              # пользователь, сделавший запуск.
                                # Для запусков на расписании заполняется "_scheduler"
   :"created_dttm": str,        # дата создания
   :"updated_dttm": str         # дата обновления
}
```

- %job_options
- %run_info

4.2.35 %job_summary

```
{
   :"job_rk": int,                  # идентификатор задания
   :"desc": str,                    # описание задания
   :"start_dttm": str,              # дата последнего запуска
   :"finish_dttm": str,             # дата завершения последнего запуска
   :"state": str,                   # enum: "new" | "queued" | "running" | "success" | "failed"
                                      # статус последнего запуска
   :"detail": str,                  # описание ошибки, если state=failed
   :"resource": str,                # ссылка на создаваемый ресурс; например /datasets/123 или /files/link?path=aalfs
   :"schedule": str                 # расписание запуска
   :"created_dttm": str,            # дата создания задания
   :"udated_dttm": str,             # дата последнего изменения задание
   :"author": str                   # пользователь, создавший задание
}
```

4.2.36 %job_task

```
{
   ~"id": str,                          # автогенерация внутри блоков par и seq, если не указан явно
   *"type": str,                        # тип задачи
   *"body": dict | [<%job_task>, ...],  # тело задачи, зависит от type
    "options": dict,                    # параметры задачи
    "params": dict,                     # локальные переменные, видные только внутри таски
    "advanced": dict                    # словарь продвинутых опций СУБД, в формате
                                        # { "dialect name": ["option", "opetion", ...] }
}
```

4.2.37 %json_formatter

```
{
    *"()": str,           # класс форматтера
    *"len_limit": int,    # ограничение длины лога
    "fmt_keys": dict,     # json с переименованием имен полей в логе
    "builtin_flds": bool  # флаг расширенного логирования
}
```

4.2.38 %key

```
{
   :"key_rk": int,            # id ключа
   *"name": str,              # имя, должно быть уникальное среди ключей
    "desc": str,       # описание ключа
   *"fs_type": str,           # id типа данных
   +"fs_type_info": <%type>,  # детальная информация по типу данных
   :"author": str,        # пользователь
   :"created_dttm": str,  # дата создания
   :"updated_dttm": str   # дата изменения
}
```

4.2.39 %key_create

```
{
   *"name": str,              # имя, должно быть уникальное среди ключей
    "description": str,       # описание ключа
   *"fs_type": str           # id типа данных
}
```

4.2.40 %key_patch

```
{
   "name": str,         # имя, должно быть уникальное среди ключей
   "description": str  # описание ключа
}
```

4.2.41 %log_formatter

```
{
    "text": <%text_formatter>,  # настройка вывода логов в текстовом формате
    "json": <%json_formatter>   # настройка вывода логов в формате json
}
```

Название конфигураций (`text` и `json` в схеме) задается пользователем.

- %text_formatter
- %json_formatter

4.2.42 %log_handler

```
{
    "stdout": <%stdout_handler>,  # параметры вывода логов в stdout
    "file": <%file_handler>       # параметры вывода логов в файл
}
```

- %stdout_handler
- %file_handler

4.2.43 %log_profile

```
{
    *"name": str,       # название профиля логирования
    "desc": str,        # описание профиля логирования
    *"profile": <%log_profile_config>,  # json конфигурация профиля логирования
    :"active_flg": bool  # флаг активного профиля логирования
}
```

- %log_profile_config

4.2.44 %log_profile_config

```
{
    *"version": int,                    # значение версии всегда указывается 1
    "disable_existing_loggers": bool,  # флаг отключения существующих логгеров, по умолчанию False
    *"formatters": <%log_formatter>,   # форматеры логгера
    *"handlers": <%log_handler>,       # хендлеры логгера
    *"loggers": <%loggers>             # логгеры компонентов
}
```

- %log_formatter
- %log_handler
- %loggers

4.2.45 %log_profile_summary

```
{
    "name": str,          # название профиля логирования
    "desc": str,          # описание профиля логирования
    :"active_flg": bool  # флаг активного статуса профиля логирования
}
```

4.2.46 %logger

```
{
    "configuredLevel": "DEBUG" | "INFO" | "WARNING" | "ERROR" | "CRITICAL",
    "effectiveLevel": "DEBUG" | "INFO" | "WARNING" | "ERROR" | "CRITICAL"
}
```

4.2.47 %logger_config

```
{
    "level": str,          # уровень логирования "DEBUG" | "INFO" | "WARNING" | "ERROR" | "CRITICAL"
    "handlers": List[str]  # список хендлеров из секции handlers
}
```

4.2.48 %loggers

```
{
    *"root": <%logger_config>,               # базовый логгер
    *"sqlalchemy.engine": <%logger_config>,  # логгер sqlalchemy
    "fastapi": <%logger_config>              # логгер fastapi
}
```

Уровень логирования и хендлер может быть настроен отдельно для используемых в приложении библиотек.

- %logger_config

4.2.49 %permission

```
{
    :"permission_rk": int,  # идентификатор права доступа
    *"object_rk": int,      # идентификатор объекта права доступа
    *"object_type": str,    # тип объекта права доступа: source, dataset
    *"username": str,       # имя пользователя права доступа
    *"permission": str,     # уровень доступа к объекту
    "name": str             # имя объекта в каталоге
}
```

4.2.50 %db_type

```
{
   *"db_name": str,       # имя БД
   *"fs_type": str,       # тип данных feature store
   *"db_type": str,       # тип данных в БД
   :"author": str,       # пользователь
   :"created_dttm": str  # дата создания
}
```

4.2.51 %fs_type

```
{
   *"fs_type": str,        # тип данных в feature store
    "desc": str,           # описание типа данных
   *"py_type": enum,       # "int" | "float" | "decimal" | "str" | "bool" | "date" | "datetime"
                             # категория типа данных
   :"author": str,        # пользователь
   :"created_dttm": str   # дата создания
}
```

4.2.52 %type_default

```
{
   *"db_name": str,           # имя базы данных
   *"db_type": str,           # тип в базе данных
   *"default_fs_type": str,  # идентификатор типа данных
   :"author": str,           # пользователь
   :"created_dttm": str      # дата создания
}
```

4.2.53 %type_indicator

```
{
   *"fs_type": str,       # тип данных feature store
   *"indicator": str,     # имя метрики
   :"author": str,       # пользователь
   :"created_dttm": str  # дата создания
}
```

4.2.54 %run_info

```
{
  :"runner": str,           # среда исполнения задания
  :"id": str                # идентификатор запуска в среде исполнения
}
```

4.2.55 %schedule

```
{
   :"schedule_rk": int,             # суррогатный ключ расписания
   *"job_rk": int,                  # суррогатный ключ задания
   *"source_rk": int,               # суррогатный ключ источника
   *"schedule": str,                # расписание задания в формате пятизначной cron-строки
    "timezone": float,              # разница с UTC, от -12.0 до 14.0
                                      # дробная часть указывается в долях часа (например, 3.5 для UTC+3:30)
   ~"job_options": <%job_options>,  # опции запуска задания
   :"author": str,                  # пользователь
   :"last_run_dttm": str,           # дата последнего запуска
   :"created_dttm": str,            # дата создания
   :"updated_dttm": str             # дата изменения
}
```

- %job_options

4.2.56 %source

```
{
   :"source_rk": int,          # идентификатор источника
   *"name": str,               # имя
    "desc": str,               # описание
   *"url": str,                # URL источника для подключения или переменная окружения через ${}
   *"work_schema": str,        # work-схема для временных объектов
   *"store_schema": str,       # store-схема для хранения целевых (постоянных) объектов
   *"schemas": list[str],      # список доступных схем
    "options": dict,           # дополнительные опции движка SQLAlchemy в формате JSON,
                               # подробнее см. "Подключение к данным"
   :"permission": str          # уровень доступа к источнику
   :"author": str,             # пользователь, создавший источник
   :"created_dttm": str,       # дата создания
   :"updated_dttm": str        # дата изменения
}
```

4.2.57 %source_health

```
{
    :"url": str,             # URL источника с маскированием логина/пароля
    :"connection": {
        :"state": enum,      # статус подключения - "ok" | "error"
        :"detail": str       # текст ошибки
    },
    :"work_schema": {
        :"state": enum,      # статус проверки work-схемы - "ok" | "error"
        :"detail": str,      # текст ошибки (нет схемы | нет прав на запись и т.д.)
    },
    :"store_schema": {
        :"state": enum,      # статус проверки store-схемы - "ok" | "error"
        :"detail": str,      # текст ошибки (нет схемы | нет прав на запись и т.д.)
    },
    :"fs_types": {
        "{fs_type}": {       # рассчитывается для каждого fs_type из ref_fs_type
            :"state": enum,  # статус проверки типа данных - "ok" | "warning" | "error"
            :"detail": str   # текст ошибки или предупреждения
        }
    }
}
```

4.2.58 %source_patch

```
{
    "name": str,              # имя
    "description": str,       # описание
    "url": str,               # URL источника для подключения или переменная окружения через ${}
    "work_schema": str,       # work-схема для временных объектов
    "store_schema": str,      # store-схема для хранения целевых (постоянных) объектов
    "schemas": list[str],     # список доступных схем
    "options": dict          # дополнительные опции движка SQLAlchemy в формате JSON,
                              # подробнее см. "Подключение к данным"
}
```

4.2.59 %stat

```
{
    :"stat_rk": int,        # Суррогатный ключ статистики
    :"dataset_rk": int,     # Суррогатный ключ датасета
    :"indincator": enum,    # Статистический показатель
    :"value": json,         # Значение статистики
    :"run_rk": int,         # Идентификатор запуска задания расчета статистик
    :"author": str,         # Пользователь, запустивший задание расчета статистик
    :"created_dttm": str,   # Дата создания записи
    :"updated_dttm": str    # Дата последнего изменения записи
}
```

4.2.60 %stat_common

```
{
    :"stat_rk": int,        # Суррогатный ключ статистики
    :"dataset_rk": int,     # Суррогатный ключ датасета
    :"date": datetime,      # Дата, на которую рассчитана статистика
    :"indincator": enum,    # Статистический показатель
    :"value": json,         # Значение статистики
    :"run_rk": int,         # Идентификатор запуска задания расчета статистик
    :"author": str,         # Пользователь, запустивший задание расчета статистик
    :"created_dttm": str,   # Дата создания записи
    :"updated_dttm": str    # Дата последнего изменения записи
}
```

4.2.61 %stat_feature

```
{
    :"stat_rk": int,        # Суррогатный ключ статистики
    :"dataset_rk": int,     # Суррогатный ключ датасета
    :"feature_rk": int,     # Суррогатный ключ переменной
    :"date": datetime,      # Дата, на которую рассчитана статистика
    :"indincator": enum,    # Статистический показатель
    :"value": json,         # Значение статистики
    :"run_rk": int,         # Идентификатор запуска задания расчета статистик
    :"author": str,         # Пользователь, запустивший задание расчета статистик
    :"created_dttm": str,   # Дата создания записи
    :"updated_dttm": str    # Дата последнего изменения записи
}
```

4.2.62 %stat_instance

```
{
    :"stat_rk": int,        # Суррогатный ключ статистики
    :"dataset_rk": int,     # Суррогатный ключ датасета
    :"instance_rk": int,    # Суррогатный ключ экземпляра сущности
    :"date": datetime,      # Дата, на которую рассчитана статистика
    :"indincator": enum,    # Статистический показатель
    :"value": json,         # Значение статистики
    :"run_rk": int,         # Идентификатор запуска задания расчета статистик
    :"author": str,         # Пользователь, запустивший задание расчета статистик
    :"created_dttm": str,   # Дата создания записи
    :"updated_dttm": str    # Дата последнего изменения записи
}
```

4.2.63 %stdout_handler

```
{
    "class": str,      # класс хендлера
    "level": str,      # уровень логирования "DEBUG" | "INFO" | "WARNING" | "ERROR" | "CRITICAL"
    "formatter": str,  # название форматтера из секции formatters
    "stream": str      # стрим (по умолчанию ext://sys.stdout)
}
```

4.2.64 %table

```
{
   :"db_schema": str,  # схема в БД
   :"db_table": str    # имя таблицы в БД
}
```

4.2.65 %extract:file

```
{
  *"type": "extract:file",
  *"body": {
     *"dataset": int | str,  # dataset_rk | "dataset name" | "{task_id}"
     *"path": str            # адрес файла, например 's3://foo.txt' или просто 'foo.txt'
   }
}
```

4.2.66 %text_formatter

```
{
    *"()": str,         # класс форматтера
    *"len_limit": int  # ограничение длины лога
}
```

4.2.67 %transform

```
{
    "desc": str,                      # описание задания
    "schedule": str,                  # cron-строка расписания
    "timezone": float,                # разница с UTC, от -12.0 до 14.0
                                        # дробная часть указывается в долях часа (например, 3.5 для UTC+3:30)
    "body": {                         # граф задач
       :"tasks": [<%job_task>, ...]     # список задач
       :"graph": {                      # зависимости между задачами
           :"<task id>": [<task id>, ...],
            ...
        }
    },
    "dispose": enum,            # "smart" | "on_success" | "always" | "manual"
                                #   default: smart
                                #   статегия автоматического удаления job-а с запусками
                                #   * smart - джоб удалится по внутренним правилам приложения:
                                #       * если создан через POST /jobs, то удаляется вручную
                                #       * если создан через POST /transform и не регистрирует в мете
                                #      датасеты/загрузчики, то удалится при успешном завершении
                                #       * иначе удалится при удалении последнего связанного
                                #      датасета/загрузчика
                                #   * on_success - джоб удалится только при успешном завершении
                                #   * always - джоб удалится после завершения
                                #   * manual - джоб удаляется только вручную
    "job_options": <%job_options>
}
```

- job_task
- job_options

4.2.68 %transform_summary

```
{
   :"job_rk": int,
   :"job_run": <%job_run>,
   :"datasets": [<%dataset>]
}
```

- job_run
- dataset

4.2.69 %user

```
{
   :"username": str,  # имя текущего пользователя
   :"roles": List[str]    # доступные роли
}
```

4.2.70 %worker_health

```
{
    "state": enum,    # "ok" | "error" # статус готовности воркера
    "detail": str     # описание ошибки, если статус "error"
    "app_version": str,  # версия приложения воркера
    "core_version": str, # версия ядра воркера
    "engine_version": str, # версия движка воркера
}
```

5. Синтаксис в формулах

Ниже описаны функции, синтаксис и поведение которых гарантируется на всех поддерживаемых data-движках. Помимо этого списка можно использовать любые функции конкретного движка, если их синтаксис укладывается в *func* (*arg*, *arg*, ...)

5.1 Поддерживаемые функции в выражениях

5.1.1 Базовые операторы

	Описание	**Пример**
`+`, `-`, `*`, `/`, `%`	стандартно как в математике	`A + B`
`=`, `!=`, `>`, `>=`, `<`, `<=`	стандартно как в математике	`A < B`
`not` ...	логическое отрицание	`not(A = B)`
`and`, `or`	логические операторы	`A and B`
`and` (x, y, ...), `or` (x, y, ...)	логические операторы	`or(A, B, C)`
x `in` (...)	x имеет значение из списка (...)	`A in (1, 2, 3)`
`in` (x, ...)	то же самое что: x `in` (...)	`in (A, 1, 2, 3)`
x `like` *string*	соответсвие текстовой маске	`A like '%retail%'`
`like` (x, *string*)	то же самое что: x `like` *string*	`like(A, '%retail%')`
x `between` y `and` z	значение x в интервале (y, z]	`A between D1 and D2`
x `not between` y `and` z	значение x не в интервале (y, z]	`A not between D1 and D2`
`between` (x, y, z)	то же самое что: x `between` y `and` z	`A between D1 and D2`
`coalesce` (x, ...)	первое значение отличное от null	`coalesce(A, B, C)`
`cast` (x `as` *type*)	приведение к *type* (fs_type)	`cast(A as AMOUNT)`
type (x)	тоже самое что: `cast` (x `as` *type*)	`AMOUNT(A)`
`case` {`when` x `then` y}+ [`else` z] `end`	оператор выбора	`case when A then B else C end`
`case` ({x, y}+ , [z])	то же самое что: `case when`	`case (A, B, C)`

5.1.2 Функции арифметики

	Описание	Пример
`abs` (*number*)	модуль	`abs (-1)`
`ceil` (*number*)	округление вверх	`ceil(0.1)`
`floor` (*number*)	округление вниз	`floor(1.1)`
`round` (*number*, [*digits*])	математическое округление	`round(1.3)`, `round(1.12345, 2)`
`trunc` (*number*, [*digits*])	оставляет *digits* знаков после запятой	`trunc(0.1234, 2)`
`ln` (*number*)	натуральный логарифм	`ln(1)`
`log` (*base*, *number*)	логарифм, *base* - основание логарифма	`log(10)`, `log(3, 10)`
`sqrt` (*number*)	квадратный корень	`sqrt(4)`
`power` (*number*, *number*)	возведение в степень	`power(2, 2)`

5.1.3 Функции работы со строками

	Описание	Пример
`concat` (*text*, ...)	объединяет строки в одну	`concat('a', 'b')`
`concat_ws` (*sep*, *text*, ...)	объединяет через *sep*	`concat_ws(';', 'a', 'b')`
`substr` (*text*, *start*, [*length*])	подстрока	`substr('abc', 1)`
`length` (*text*)	длина	`length('abc')`
`replace` (*text*, *from*, *to*)	замена подстроки	`replace('abc', 'a', 'b')`
`reverse` (*text*)	разворот	`reverse('abc')`
`trim` (*text*, [*chars*])	удаление символов из *chars*	`trim('__abc__', '_')`
`ltrim` (*text*, [*chars*])	удаление символов слева	`ltrim('__abc', '_')`
`rtrim` (*text*, [*chars*])	удаление символов справа	`rtrim('abc__', '_')`
`strpos` (*text*, *what*)	поиск подстроки	`strpos('abc','a')`
`translate` (*text*, *from*, *into*)	замена символов по словарю	`translate('abc', 'ab', '12')`
`upper` (*text*)	в верхний регистр	`upper('abc')`
`lower` (*text*)	в нижний регистр	`lower('ABC')`

5.1.4 Функции работы с датами

	Описание	Пример
`d` '*string*'	константа типа `дата`	`d'2022-01-13'`
`dt` '*string*'	константа типа `дата+время`	`dt'13.01.2023 00:30:59'`
`interval` {*number span*}*	константа типа интервал. `span`	`interval 1 day`
`date_add` (*date*, *interval*)	добавление интервала к дате	`date_add(DATE_1, interval 1 day)`
`date_sub` (*date*, *interval*)	вычитание интервала из даты	`date_sub(DATE_2, interval 1 month)`
`date_part` (*span*, *date*)	вычленение части даты.	`date_trunc('month', d'1900-01-01')`
`date_trunc` (*span*, *date*)	округление даты.	`date_part('day', d'1900-01-01')`
`today` ()	текущая дата	`today()`
`now` ()	текущие дата и время	`now()`
`day_start` (*date*)	начало дня в формате datetime	`day_start(now())`
`day_end` (*date*)	конец дня в формате datetime	`day_end(today())`
x in `LAST_itrv` (y)	Дата x в интервале (y - `itrv` ; y]	`issue_dt in last_3d(report_dt)`
x in `BEFORE_itrv` (y)	Дата x в интервале (-inf; y - `itrv`]	`issue_dt in before_1y3w(report_dt)`

5.1.5 Агрегирующие функции

	Описание	Пример
`min` (x)	минимум	`min(customer_age)`
`max` (x)	максимум	`max(customer_age)`
`count` (x)	количество	`count(customer_id)`
`mean` (x)	среднее арифметическое	`mean(customer_age)`
`median` (x)	медиана – серединное значение	`median(customer_age)`
`mode` (x)	мо́да — наиболее часто встречающееся значение	`mode(customer_age)`

5.1.6 Аналитические функции

func `over` ([`partition by` x, ...] [`order by` z [*acs* | *desc*] [*nulls_last* | *nulls_first*], ...])

func:

- `row_number` ()
- `min` (...)
- `max` (...)
- `mean` (...)
- `count` (...)

5.1.7 Appendix

Форматы дат

Для составляющей даты поддерживаются форматы: `YYYY-MM-DD`, `DD.MM.YYYY`, `MM/DD/YYYY`

Для составляющей времени формат: `hh:mm:ss` – который присоединяется к составляющей даты через пробел: '1999-01-02 03:04:05', '01.02.1999 03:04:05', '02/01/1999 03:04:05'

Форматы `itrv`

Литера	Описание
`y`	year
`m`	month
`w`	week
`d`	day
`h`	hour
`mi`	minute
`s`	second

Примеры: 1y2m3w, 12h30mi

Возможные значение `span`

- second
- minute
- hour
- day
- week
- month
- year