Документация

1. Использование Аннотации

Главный класс помечается аннотацией *@RestApiGenerator* с параметром «*jsonPath*», в котором указывается полный путь к JSON файлу. Кроме того, требуется использование обычных аннотаций *Spring-Boot* приложения (листинг 1).

Листинг 1 — Использование аннотации.

```
@SpringBootApplication
@RestApiGenerator(jsonPath="JsonFileFullPath")
public class Main {
   public static void main(String[] args) {
      SpringApplication.run(Main.class,args);
   }
}
```

2. Группы конечных точек

Файл в формате JSON имеет структуру, изображенную в листинге 2.

«*EndpointGroup1*» и «*EndpointGroup2*» — это названия групп конечных точек, на основе которых будут сгенерированы контроллеры в формате: «*EndpointGroupController*» и репозитории в формате: «*EndpointGroupRepository*». «*http*» - место размещения информации о конечных точках.

Листинг 2 — Формат файла JSON.

```
{
  "EndpointGroup1": {
    "http": {
    }
  },
   "EndpointGroup2": {
    "http": {
    }
  }
}
```

http_prefix

Листинг 3 — Использование «*http_prefix*».

```
{
  "EndpointGroup": {
  "http_prfix":"prefix/",
   "http": {
    }
  }
}
```

При использовании параметра «*http_prefix*» (листинг 3), к которому принадлежит *«EndpointGroup*» группа конечных точек, появляется аннотация: *@RequestMapping(“prefix/”).*

3. Конечные точки

Листинг 4 — Формат конечной точки.

```
"http": {
    "EndpointName1": {
      "type":"someType",
      "request": "endpoint"
    },
    "EndpointName2": {
      "type":"someType",
      "request": "endpoint"
    }
```

```
}
```

Конечные точки размещаются в поле «*http*» (листинг 4) «*EndpointName1*» и «*EndpointName2*» — это названия конечных точек, на основе которых будут сгенерированы функции в контроллерах и репозиториях в формате: <Тип><Имя конечной точки>. *«type»* - тип запроса. *«request»* - место размещения самой конечной точки, при парасанге которой будет сгенерирован запрос к РСУБД.

4. Тип запроса

Тип запроса размещается в конечной точке и может иметь следующие форматы (листинг 5–8).

Листинг 5 — Формат типа запроса.

```
"EndpointName1": {
   "get":"entity->limit|sort|fields",
   "post":"entity->entity",
   "put":"entity->entity",
   "patch":"entity->entity",
   "delete":"",
   "request": "endpoint"
}
```

В листинге 5 в *«EndpointName1»* все типы запроса относятся к одной и той же конечной точке. Использование всех пяти типов в конечной точке не обязательно.

Листинг 6 — Формат типа запроса.

```
"EndpointName2": {
   "type":"get:entity",
   "request": "endpoint"
},
"EndpointName3": {
   "type":"post:entity->entity",
   "request": "endpoint"
},
"EndpointName4": {
   "type":"put:entity->entity",
   "request": "endpoint"
},
"EndpointName5": {
   "type":"patch:entity->entity",
   "request": "endpoint"
},
"EndpointName6": {
   "type":"delete",
   "request": "endpoint"
}
```

В «EndpointName2», «EndpointName3», «EndpointName4», «EndpointName5», «EndpointName6» используется один тип запроса на одну конечную точку (листинг 6).

Листинг 7 — Формат типа запроса.

```
"EndpointName7": {
   "types":{
      "get":{},
      "post":{},
      "put":{},
      "delete":{},
      "patch":{}
   },
   "request": "endpoint"
}
```

В *«EndpointName7»* все типы запроса относятся к одной и той же конечной точке. Использование всех пяти типов в конечной точке не обязательно. В этом формате можно указать более расширенную информацию о каждом типе (листинг 7).

Листинг 8 — Формат типа запроса.

```
"EndpointName8": {
  "request": "endpoint"
}
```

В *«EndpointName8»* по умолчанию будет тип запроса *GET* (листинг 8).

Использование двух форматов в одной и той же конечной точке недопустимо, так как будет сформирована ошибка времени компиляции.

Тип запроса *GET*

В типе запроса по умолчанию *GET* (листинг 9–10), в случае успеха, будет возвращен код 200, а метод в контроллере помечен аннотацией: @ResponseStatus (HttpStatus.OK).

В *«EndpointName1»* и *«EndpointName2» «field1|field2|field3»* (листинг 9) — это список возвращаемых полей (для *GET* они не обязательные). Поля перечисляются через «|» с учетом их псевдонимов. Через «->» указываются используемые дополнительные параметры (также не обязательные).

Листинг 9 — Тип запроса *GET*.

```
"EndpointName1": {
    "get":"field1|field2|field3->limit|sort|fields",
    "request": "endpoint"
},
"EndpointName2": {
   "type":"get:field1|field2|field3",
   "request": "endpoint"
}
```

Листинг 10 — Тип запроса *GET*.

```
"EndpointName3": {
   "types":{
      "get":{
      "entity":"field1|field2,
      "return":"limit|sort|fields",
      "httpOk":"OK"
      }
   },
   "request": "endpoint"
}
```

В *«EndpointName3»* параметр *«entity»* — это возвращаемые поля (для *GET* этот параметр не обязательный). Поля перечисляются через «|» с учетом их псевдонимов. В *«return»* указываются используемые дополнительные параметры (также не обязательные). В *«httpOk»* указывается *org.springframework.http.HttpStatus*.

Параметр *«limit»*

В случае указания параметра *«limit»* генерируется код (листинг 11).

Листинг 11 — Код при использовании *«limit»*.

```
@Min(0) @RequestParam(defaultValue = "9223372036854775807")
@Parameter(name = "limit", example = "1") long limit,
@Min(0) @RequestParam(defaultValue = "0") long offset
```

Как видно из листинга 11, оба сгенерированных параметра являются необязательными. В случае обращения к РСУБД, в конце запроса будет добавлен *«limit»* и *«offset»*.

Параметр *«sort»*

В случае указания параметра *«sort»* генерируется код (листинг 12).

Листинг 12 — Код при использовании *«sort»*.

```
@RequestParam @Parameter(name = "sort", example = "fieldName")
String sort,
@RequestParam(defaultValue = "true") @Parameter(name = "asc",
example = "true", description = "asc=true, desc=false") boolean asc
```

Как видно из Листинга 12, сгенерированный параметр *«sort»* является обязательным, а *«asc»* не обязательным. Он имеет тип *boolean,* и, в случае *true* будет *ASC*, а в случае *false DESC*. Параметр *«sort»* — это имя поля для функции *ORDER BY*. В случае обращения к РСУБД, в конце запроса перед *LIMIT* будет добавлен *ORDER BY*.

Параметр *«fields»*

В параметре *«fields»* перечисляются возвращаемые поля. Параметр не обязателен, и, в случае его отсутствия, возвращаются поля, перечисленные в *«entity»*. В случае передачи полей, которых нет в *«entity»,* они не будут возвращены запросом. Если полей в *«entity»* вообще не будет, будут возвращены все поля, переданные в параметре *«fields»*.

Тип запроса *POST*

В типе запроса *POST* по умолчанию (листинг 13–14) в случае успеха, будет возвращен код 201, а метод в контроллере помечен аннотацией: *@ResponseStatus (HttpStatus.CREATED).*

Листинг 13 — Тип запроса *POST*.

```
"EndpointName1": {
 "post":"fieldStr1-s=str|field2-i=23|field3-b=true->field1|field2",
   "request": "endpoint"
},
"EndpointName2": {
   "type":"post:fieldStr1-s=str|field2-i=23|field3-b=true",
   "request": "endpoint"
}
```

В «EndpointName1» и «EndpointName2» (листинг 13) *«fieldStr1-s=str|field2-i=23|field3-b=true»*— это список полей для вставки (для *POST* они обязательные). Поля перечисляются через «|» с учетом их псевдонимов в формате *«fieldType = DEFAULT»*. *«DEFAULT»* по умолчанию *null*; *«DEFAULT»* будет вставлено в поле, если оно будет отсутствовать. Через «->» указываются возвращаемые поля.

Листинг 14 — Тип запроса *POST*.

```
"EndpointName3": {
   "types":{
      "post":{
      "entity":"field1-s=str|field2,
      "return":"field1|field2",
      "httpOk":"CREATED"
      }
   },
   "request": "endpoint"
}
```

В *«EndpointName3»* параметр *«entity»* (листинг 14) — это вставляемые поля (для *POST* этот параметр обязательный). Поля перечисляются через «|» с учетом их псевдонимов. В *«return»* указываются возвращаемые поля. В *«httpOk»* указывается *«enum»* класса *org.springframework.http.HttpStatus.*

Тип запроса *PUT*

В типе запроса *PUT* по умолчанию (листинг15-16), в случае успеха, будет возвращен код 200, а метод в контроллере помечен аннотацией: *@ResponseStatus (HttpStatus.OK).*

Листинг 15 — Тип запроса *PUT.*

```
"EndpointName1": {
   "put":"fieldStr1-s=str|field2-i=23|field3-b=true->field1|field2",
   "request": "endpoint"
},
"EndpointName2": {
   "type":"put:fieldStr1-s=str|field2-i=23|field3-b=true",
   "request": "endpoint"
}
```

В *«EndpointName1»* и *«EndpointName2»* (листинг 15) *«fieldStr1-s=str|field2-i=23|field3-b=true»* — это список полей для вставки (для *PUT* они обязательные). Поля перечисляются через «|» с учетом их псевдонимов в формате *«fieldType = DEFAULT». «DEFAULT»* по умолчанию *«null»; «DEFAULT»* будет обновлять поле, если оно будет отсутствовать. Через «->» указываются возвращаемые поля.

Листинг 16 — Тип запроса *PUT.*

```
"EndpointName3": {
   "types":{
      "post":{
      "entity":"field1-s=str|field2,
      "return":"field1|field2",
      "httpOk":"OK"
      }
   },
   "request": "endpoint"
}
```

В *«EndpointName3»* параметр *«entity»* (листинг 16) — это обновляемые поля (для *PUT* этот параметр обязательный). Поля перечисляются через «|» с учетом их псевдонимов. В *«return»* указываются возвращаемые поля. В *«httpOk»* указывается *«enum»* класса *org.springframework.http.HttpStatus.*

Тип запроса *PATCH*

В типе запроса *PATCH* по умолчанию (листинг 16–17), в случае успеха, будет возвращен код 200, а метод в контроллере помечен аннотацией: *@ResponseStatus (HttpStatus.OK).*

Листинг 17 — Тип запроса *PATCH.*

```
"EndpointName1": {
   "patch":"field1|field2|field3->field1|field2",
   "request": "endpoint"
},
"EndpointName2": {
   "type":"patch:field1|field2|field3",
   "request": "endpoint"
}
```

В *«EndpointName1»* и *«EndpointName2»* (листинг 17) *«field1|field2|field3»* — это список полей для вставки (для *PATCH* они обязательные). Поля перечисляются через «|», с учетом их псевдонимов. Через «->» указываются возвращаемые поля.

Листинг 18 — Тип запроса *PATCH.*

```
"EndpointName3": {
   "types»: {
      "patch»: {
      "entity":"field1-s=str|field2,
```

```
      "return":"field1|field2",
      "httpOk":"OK"
      }
   },
   "request": "endpoint"
}
```

В *«EndpointName3»* параметр *«entity»* (листинг 18) — это обновляемые поля (для *PATCH* этот параметр обязательный). Поля перечисляются через «|», с учетом их псевдонимов. В *«return»* указываются возвращаемые поля. В *«httpOk»* указывается *«enum»* класса *org.springframework.http.HttpStatus.*

Тип запроса *DELETE*

В типе запроса *DELETE* по умолчанию (листинг 19), в случае успеха, будет возвращен код 204, а метод в контроллере помечен аннотацией: *@ResponseStatus (HttpStatus.NO_CONTENT).*

Листинг 19 — Тип запроса *DELETE.*

```
"EndpointName1": {
  "delete":"",
  "request": "endpoint"
},
"EndpointName2": {
  "type":"delete",
  "request": "endpoint"
},
"EndpointName3": {
  "types":{
    "delete":{
      "httpOk":"NO_CONTENT"
    }
  },
  "request": "endpoint"
}
```

В *«httpOk»* указывается *«enum»* класса *org.springframework.http.HttpStatus.*

5. Тип данных

Поддерживаются 6 типов данных (таблица 1).

Таблица 1 — Типы данных.

Тип.	Обозначение.
String	*-s*
Double	*-d*
Boolean	*-b*
Float	*-f*
Long	*-l*
Integer	*-i*

6. Предикаты

Поддерживаются 12 предикатов (таблица 2).

Таблица 2 — Предикаты.

SQL.	Обозначение.
==	*eq_*
!=	*ne_*
>	*gt_*

<	*lt_*
>=	*ge_*
=<	*le_*
LIKE	*like_*
LIKE REGEX	*reg_*
IN	*in_*
NOT LIKE	*not_like_*
NOT LIKE REGEX	*not_reg_*
NOT IN	*not_in_*

7. Применение предиката к полю

Формат применения предиката имеет вид: <Предикат><Имя поля><Тип> пример представлен в листинге 20.

Листинг 20 — Применение предиката.

```
"like_FieldName-s"
```

Применить *LIKE* к полю с именем *«FieldName»,* у которого тип *String*. Предикаты могут указываться как в фильтрах, так и в запросе. Сгенерированный участок кода примет форму (листинг 21).

Листинг 21— Применение предиката. Сгенерированный код.

```
DSL.field("FieldName").like(like_FieldName)
```

«like_FieldName» - имя переменной в параметрах для фильтра или *«PathVariable»*. По умолчанию предикат это *«eq_»,* а тип *Long*. В случае отсутствия предиката имя переменной будет *«FieldName»*.

8. Запрос к РСУБД

Конечная точка размещается в поле *«request»* и имеет формат:

Листинг 22 — Формат конечной точки.

```
"request":"Table1/{like_F1-s}|{le_F2}&({ne_F3}|{F5})/{F3}"
```

Для листинга 22, в случае типа *GET*, будет сформирован запрос к РСУБД, находящийся в листинге 23.

Листинг 23— Запрос к РСУБД для листинга 22.

```
SELECT * FROM TABLE1
WHERE (F1 LIKE ? OR (F2<=? AND (F3!=? OR F5=?))) AND (F3=?);
```

Внутри фигурных скобок «{}» находятся поля таблицы *«Table1»,* а в квадратных скобках «[]» размещаются фильтры, которые могут быть применены к этим полям. Возможно также составление условий из полей и фильтров. Если фильтр не содержит параметров, он не будет оказывать влияние на остальные условия. Для условий поддерживаются операторы «&»-«и» и «|»-«или», а также возможно использование скобок «()». Косая черта «/» между полями обозначает операцию «и» и имеет самый низкий приоритет.

Адрес

Листинг 24 — *«request»*.

```
"request":"Table1/{f1-s}/Table2/{f2-s}/Table3/{f3-s}/Table4"
```

В запросе может принимать участие любое число таблиц. Библиотека поддерживает следующие виды связей: один ко многим; многие к одному; многие ко многим. Связи между таблицами указывается в параметрах *«joins»*

Часть запроса, относящегося ко всем таблицам, кроме двух последних таблиц для *POST* и последней для остальных, выглядит одинаково по структуре.

Для листинга 24 эта часть будет выглядеть как в листинге 25.

Листинг 25 — Часть запроса к РСУБД, относящейся к 1 части запроса листинга 24.

```
SELECT ID FROM TABLE3 WHERE TABLE2_ID IN (
SELECT ID FROM TABLE2 WHERE TABLE1_ID IN (
SELECT ID FROM TABLE1 WHERE (F1=?)) AND (F2=?)) AND (F3=?)
```

Формируется *SELECT* из *IN*, в котором находится *SELECT* выбирающий поле, которое в свою очередь является связью с предыдущей таблицей в запросе, а также условием выбора, соединенного через «и».

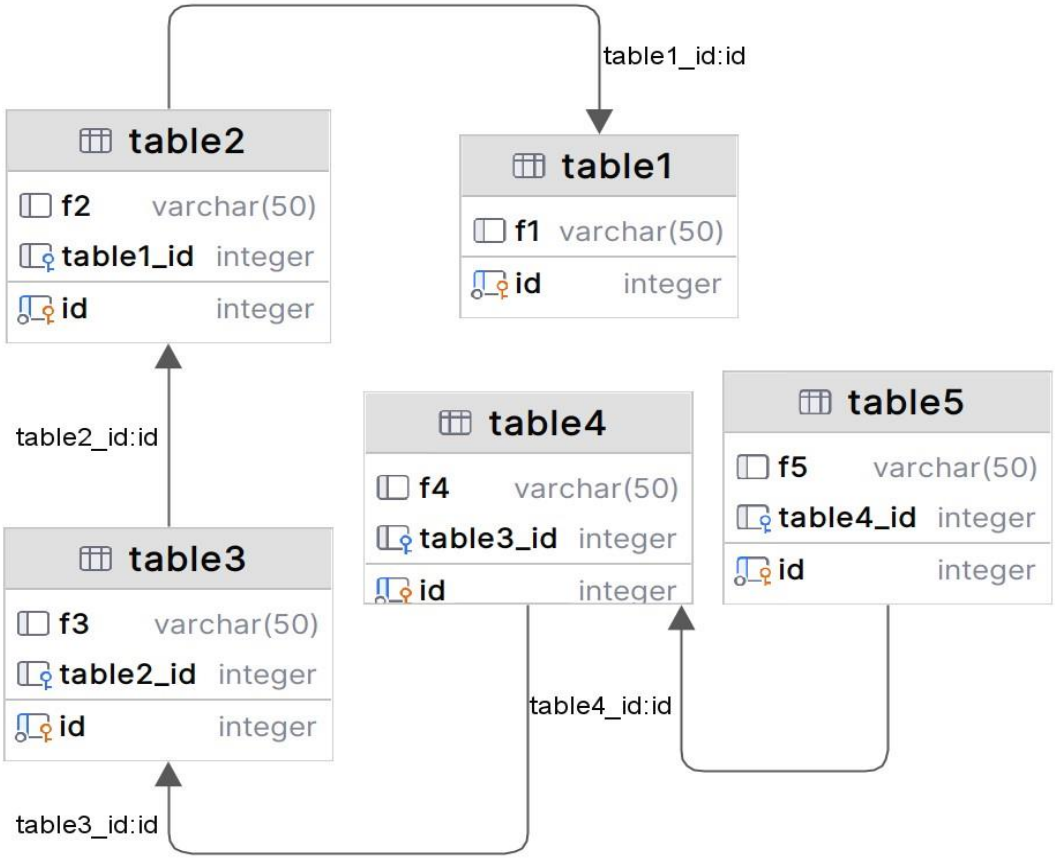

Рисунок 1 — Схема 1.

Запрос *GET*

Листинг 26 — Запрос *GET*.

```
"get":"F5|I5",
"request":"Table2/{f2-s}/Table3/{f3-s}/Table4/{f4-s}/Table5/{f5-s}/"
```

Схема базы данных, к которой относится этот запрос из листинга 26 изображена на рисунке 1.

Листинг 27 — Запроса к РСУБД, относящейся к листингу 26.

```
SELECT F5, I5 FROM TABLE5 WHERE TABLE4_ID IN(
SELECT ID FROM TABLE4 WHERE TABLE3_ID IN (
SELECT ID FROM TABLE3 WHERE TABLE2_ID IN (
SELECT ID FROM TABLE2 WHERE
(F2=?)) AND (F3=?)) AND (F4=?)) AND F5=?;
```

В *GET* запросе, часть, относящаяся к адресу — это все таблицы, кроме последней. К первой таблице относится *SELECT* с полями, перечисленными в типе запроса.

Запрос *POST*

В запросе *POST* можно вернуть поля, но только относящиеся к последней таблице. Вне зависимости от того существует ли выборка первой части или нет все равно будет вставлена хотя бы одна запись.

Одна таблица

Листинг 28 — Запрос *POST*. Случай 1 таблица.

```
"post":"F5|I5",
"request":"Table5/"
```

Схема базы данных, к которой относится этот запрос из листинга 28 изображена на рисунке 1.

Листинг 29 — Запроса к РСУБД, относящейся к листингу 28.

```
INSERT INTO TABLE5 (F5, I5) VALUES(?, ?);
```

Один ко многим

Листинг 30 — Запрос *POST*. Случай один ко многим.

```
"post":"F5|I5",
"request":"Table3/{f3-s}/Table4/{f4-s}/Table5/"
```

Схема базы данных, к которой относится этот запрос из листинга 30 изображена на рисунке 1.

Листинг 31 — Запроса к РСУБД, относящейся к листингу 30.

```
INSERT INTO TABLE5 (TABLE4_ID, F5, I5) (SELECT NULL, ?, ?
WHERE NOT EXISTS (
SELECT ID, ?, ? FROM TABLE4 WHERE TABLE3_ID IN (
SELECT ID FROM TABLE3 WHERE (F3=?)) AND (F4=?))
UNION ALL(
SELECT ID, ?, ? FROM TABLE4 WHERE TABLE3_ID IN (
SELECT ID FROM TABLE3
WHERE (F3=?)) AND (F4=?)));
```

Многие ко многим

Листинг 32 — Запрос *POST*. Случай многие ко многим.

```
"post":"F5|I5",
"request":"Table3/{f3-s}/Table4/{f4-s}/Table5/"
"pseudonyms": {
   "joins": {
     "Table4:Table3":["Table4_has_Table5"]
   }
}
```

Схема базы данных, к которой относится запрос из листинга 32 изображена на рисунке 2. Она состоит из 2-х частей (листинг 33–34).

Листинг 33 — Запроса к РСУБД, относящейся к листингу 32. Первая часть.

```
INSERT INTO TABLE5 (F5, I5) VALUES(?, ?) RETURNING ID as "RESULT_ID";
```

Листинг 34. Запроса к РСУБД, относящейся к листингу 32. Вторая часть.

```
INSERT INTO TABLE4_HAS_TABLE5 (TABLE4_ID, TABLE5_ID) (
SELECT RESULT_ID,? FROM TABLE4 WHERE TABLE3_ID IN (
SELECT ID FROM TABLE3 WHERE (F3=?)) AND (F4=?));
```

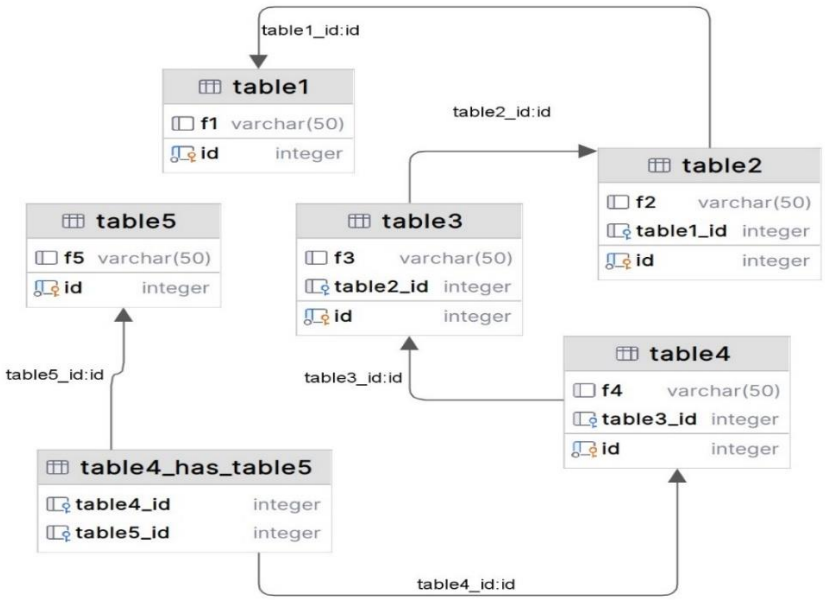

Рисунок 2 — Схема 2.

Многие к одному

Листинг 35 — Запрос *POST*. Случай многие ко одному.

```
"post":"F5|I5",
"request":"Table3/{f3-s}/Table4/{f4-s}/>Table5/"
```

«>» — это отображение типа связи многие к одному. Схема базы данных, к которой относится запрос из листинга 35 изображена на рисунке 3. Он состоит из 2-х частей (листинг 36–37).

Листинг 36 — Запроса к РСУБД, относящейся к листингу 35. Первая часть.

```
INSERT INTO TABLE5 (F5, I5) VALUES(?, ?) RETURNING ID AS "RESULT_ID";
```

Листинг 37. Запроса к РСУБД, относящаяся к листингу 32. Вторая часть.

```
UPDATE TABLE4 SET TABLE5_ID= RESULT_ID
WHERE TABLE3_ID IN (
SELECT ID FROM TABLE3
WHERE (F3=?)) AND (F4=?);
```

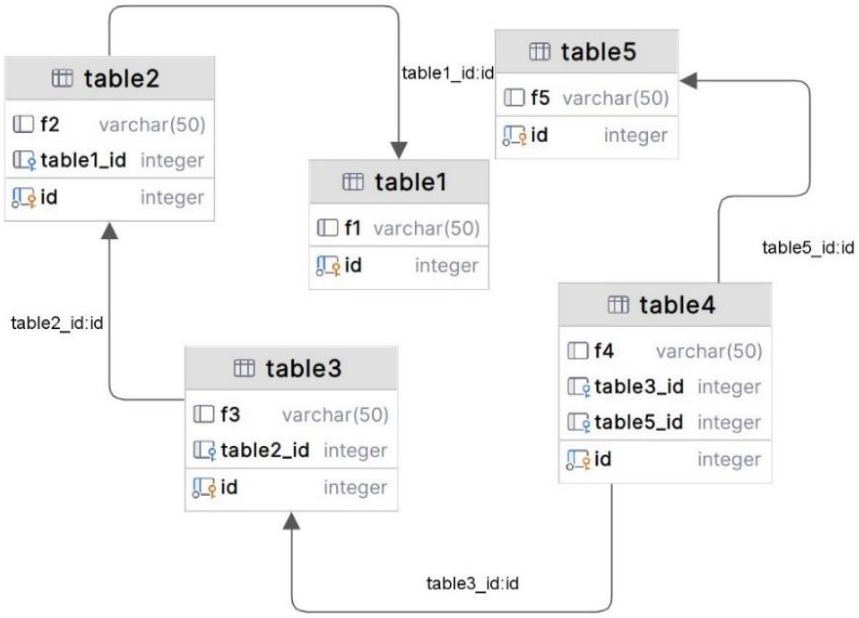

Рисунок 3 — Схема 3.

Запрос *PUT*

Листинг 38 — Запрос *PUT*.

```
"put":"F5=str|I5=10",
"request":"Table2/{f2-s}/Table3/{f3-s}/Table4/{f4-s}/Table5/{f5-s}"
```

Схема базы данных, к которой относится запрос из листинга 38 изображена на рисунке 1. Запрос к РСУБД находится в листинг 39.

Листинг 39 — Запроса к РСУБД, относящейся к листингу 38.

```
UPDATE TABLE5 SET F5=?, I5=?
WHERE TABLE4_ID IN (
SELECT ID FROM TABLE4 WHERE TABLE3_ID IN (
SELECT ID FROM TABLE3 WHERE TABLE2_ID IN (
SELECT ID FROM TABLE2 WHERE
(F2=?)) AND (F3=?)) AND (F4=?)) AND F5=?;
```

В *PUT* запросе, часть, относящаяся к адресу — это все таблицы, кроме последней. К первой таблице относится *PUT* с *IN* вместе *SELECT* к предыдущем таблицам и условиям выбора. В запросе *PUT*, в случае не передачи какого-то значения, будет вставлено значение по умолчанию. Если его нет, то null. В данном случае *«F5=“str”», «I5=10»*.

Запрос *PATCH*

Листинг 40 — Запрос *PATCH*.

```
"patch":"F5|I5",
"request":"Table2/{f2-s}/Table3/{f3-s}/Table4/{f4-s}/Table5/{f5-s}"
```

Схема базы данных, к которой относится запрос из листинга 40 изображена на рисунке 13. Запрос к РСУБД изображен в листинг 41–42.

Листинг 41— Запроса к РСУБД, относящейся к листингу 40. Все параметры есть.

```
UPDATE TABLE5 SET F5=?, I5=?
WHERE TABLE4_ID IN (
SELECT ID FROM TABLE4 WHERE TABLE3_ID IN (
SELECT ID FROM TABLE3 WHERE TABLE2_ID IN (
SELECT ID FROM TABLE2 WHERE
(F2=?)) AND (F3=?)) AND (F4=?)) AND F5=?;
```

В *PATCH* запросе, часть, относящаяся к адресу — это все таблицы, кроме последней. К первой таблице относится *PATCH* с *IN* вместе с *SELECT* к предыдущим таблицам и

условиям выбора. В запросе *PATCH*, в случае не передачи какого-то значения поля, будет сформирован запрос, находящийся в листинге 42 (не передано *«F5»*).

Листинг 42 — Запроса к РСУБД, относящейся к листингу 40. Нет *«F5»*.

```
UPDATE TABLE5 SET F5=F5, I5=?
WHERE TABLE4_ID IN (
SELECT ID FROM TABLE4 WHERE TABLE3_ID IN (
SELECT ID FROM TABLE3 WHERE TABLE2_ID IN (
SELECT ID FROM TABLE2 WHERE
(F2=?)) AND (F3=?)) AND (F4=?)) AND F5=?;
```

Запрос *DELETE*

Листинг 43 — Запрос *DELETE*.

```
"type":"delete",
"request":"Table3/{f3-s}/Table4/{f4-s}/Table5/{f5-s}"
```

Схема базы данных, к которой относится запрос из листинга 43, изображена на рисунке 1.

Листинг 44 — Запроса к РСУБД, относящейся к листингу 43.

```
DELETE FROM TABLE5 WHERE TABLE4_ID IN (
SELECT ID FROM TABLE4 WHERE TABLE3_ID IN (
SELECT ID FROM TABLE3 WHERE
(F3=?)) AND (F4=?)) AND F5=?;
```

В *DELETE* запросе, часть, относящаяся к адресу — это все таблицы, кроме последней. К первой таблице относится *DELETE* с *IN* вместе с *SELECT* предыдущей таблице и условием выбора.

9. Псевдонимы

«joins»

По умолчанию поля связаны типом связи один ко многим.

Листинг 45 — Обозначение связи для таблиц в запросе.

```
{
"GroupDefault":{
 "http":{
      "endpointTable2ToTable1":{
       "type":"get",
       "request":"table1/table2"
      },
      "endpointTable1ToTable2":{
       "type":"get",
       "request":"table1/table2"
      },
      "endpointManyToOne":{
       "type":"get",
       "request":"table1/>table2"
      },
       "endpointOneToMany":{
       "type":"get",
       "request":"table1/<table2"
      }
    }
  }
}
```

Для *«endpointTable2ToTable1»* будет сгенерирован запрос к РСУБД (листинг 46).

Листинг 46. Запроса к РСУБД, относящейся к листингу 45. Для *“endpointTable2ToTable1”*.

```
SELECT * FROM TABLE2
WHERE TABLE1_ID IN (
SELECT ID FROM TABLE1);
```

Однако и для «*endpointTable1ToTable2»* будет сгенерирован запрос к РСУБД (листинг 47).

Листинг 47 — Запроса к РСУБД, относящейся к листингу 45. Для «endpointTable1ToTable2».

```
SELECT * FROM TABLE1
WHERE TABLE2_ID IN (
SELECT ID FROM TABLE2);
```

Однако обычно 2 таблицы так не связанны. Тип связи один ко многим или многие к одному можно указать в самом запросе «<» или «>» - этот знак указывается перед таблицей и обозначает тип связи (в сгенерированной конечной точке он будет отсутствовать).

Так для *«endpointManyToOne»,* код обращения к РСУБД представлен в листинге 48.

Листинг 48 — Запроса к РСУБД, относящейся к листингу 45. Для «endpointManyToOne».

```
SELECT * FROM TABLE1
WHERE TABLE2_ID IN(
SELECT ID FROM TABLE2);
```

А для *«endpointOneToMany»,* код обращения к РСУБД представлен в листинге 49.

Листинг 49 — Запроса к РСУБД, относящейся к листингу 45. Для *«endpointOneToMany»*.

```
SELECT * FROM TABLE2
WHERE TABLE1_ID IN (
SELECT ID FROM TABLE1);
```

Поля связи таблиц

Схема базы данных, к которой относится запрос из листинга 50 изображена рисунке 5. Если поля связи таблицы имеют нестандартные имена, то они указываются в *«joins»*. Запросы к РСУБД, относящиеся к этим конечным точкам, находятся в листингах 51–52.

Листинг 50 — Пример указания связей между таблицами.

```
{
"Table1Group":{
   "pseudonyms": {
     "joins": {
       "table1:table2": ["ref",table1_ref]
     }
   },
   "http":{
      "endpointGetTable1RefTable2":{
       "type":"get",
       "request":"table1/table2"
      },
       "endpointGetTable2RefTable1":{
       "type":"get",
       "request":"table2/table1"
     }
   }
  }
}
```

Листинг 51 — Запроса к РСУБД, относящейся к листингу 50. Для *«endpointGetTable1RefTable2»*.

```
SELECT *FROM table2
WHERE table1_ref IN (SELECT ref FROM table1);
```

Листинг 52 — Запроса к РСУБД, относящейся к листингу 50. Для *«endpointGetTable2RefTable1»*

```
SELECT *FROM table1
WHERE ref IN (SELECT table1_ref FROM table2);
```

Тип связи не важен для всех типов запросов, кроме *POST*. В случае связи многие к одному, знак связи: «>» для *«POST»*, следует указывать в последней таблице в конечной точке.

Многие к Одному

Схема базы данных, к которой относится следующий запрос, изображена на схеме рисунка 4. Тип связи многие к одному можно указывать и в *«joins»*, однако, в этом случае нельзя дать полям связи иные имена, кроме тех, что будут даны по умолчанию (листинг 53). Запрос к РСУБД, находятся в листингах 54–55.

Листинг 53 — Пример указания связей между таблицами.

```
{
"Table1Group":{
    "pseudonyms": {
      "joins": {
        "table1:table2": ["<"]
      }
    },
    "http":{
       "endpointGetTable1ManyToOneTable2":{
        "type":"get",
        "request":"table1/table2"
      },
       "endpointGetTable2ManyToOneTable1":{
        "type":"get",
        "request":"table2/table1"
      }
    }
  }
}
```

Листинг 54 — Запроса к РСУБД, относящейся к листингу 53. Для *«endpointGetTable1ManyToOneTable2».*

```
SELECT * FROM table2
WHERE TABLE1_ID IN (
SELECT ID FROM table1);
```

Листинг 55 — Запроса к РСУБД, относящейся к листингу 53. Для *«endpointGetTable2ManyToOneTable1».*

```
SELECT * FROM table1
WHERE ID IN (
SELECT TABLE1_ID FROM table2);
```

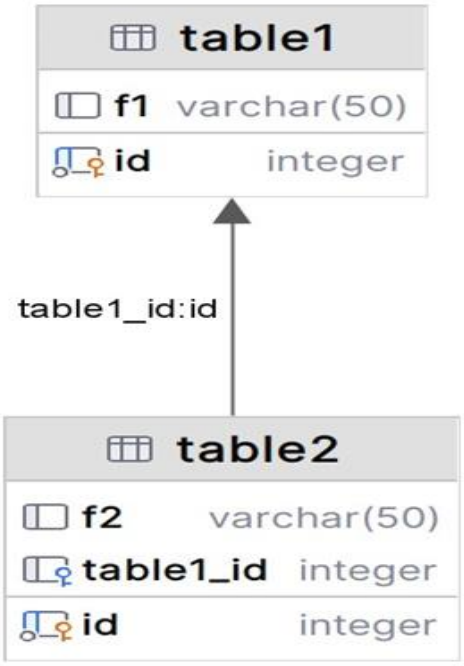

Рисунок 4 — Схема 4.

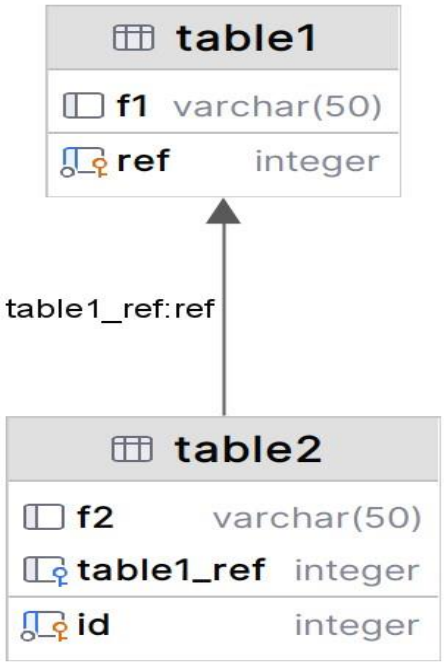

Рисунок 5 — Схема 5.

Один ко многим

Схема базы данных, к которой относится запрос из листинга 56, изображена на схеме рисунка 4. Тип связи один ко многим можно указывать и в *«joins»*. Однако, в этом случае нельзя дать полям связи иные имена, кроме тех, что будут даны по умолчанию. Запрос к РСУБД (листинг 57–58).

Листинг 56 — Пример указания связей между таблицами.

```
{
"Table1Group": {
"pseudonyms": {
"joins": {
"table2:table1": [">"]
}
},
"http":{      "endpointGetTable1OneToManyTable2": {
        "type":"get",
        "request":"table1/table2"
      },
       "endpointGetTable2OneToManyTable1": {
        "type":"get",
        "request":"table2/table1"
      }
    }
  }
}
```

Листинг 57 — Запроса к РСУБД, относящейся к листингу 56 для *«endpointGetTable1OneToManyTable2»*.

```
SELECT * FROM table2
WHERE TABLE1_ID IN (
SELECT ID FROM table1);
```

Листинг 58 — Запроса к РСУБД, относящейся к листингу 56 для *«endpointGetTable2OneToManyTable1»*.

```
SELECT * FROM table1
```

```
WHERE ID IN (
SELECT TABLE1_ID FROM table2);
```

Многие ко многим

Схема базы данных, к которой относится запрос из листинга 59, изображена на рисунке 19. Запрос к РСУБД (листинг 60–61).

Листинг 59 — Пример указания связей между таблицами.

```
{
"Group":{
   "pseudonyms": {
     "joins": {
       "table1:table2": ["table2_has_table1"]
     }
   },
   "http":{
      "endpointGetTable1ManyToManyTable2":{
       "type":"get",
       "request":"table1/table2" },
       "endpointGetTable2ManyToManyTable1":{
       "type":"get",
       "request":"table2/table1" }
   }
  }
}
```

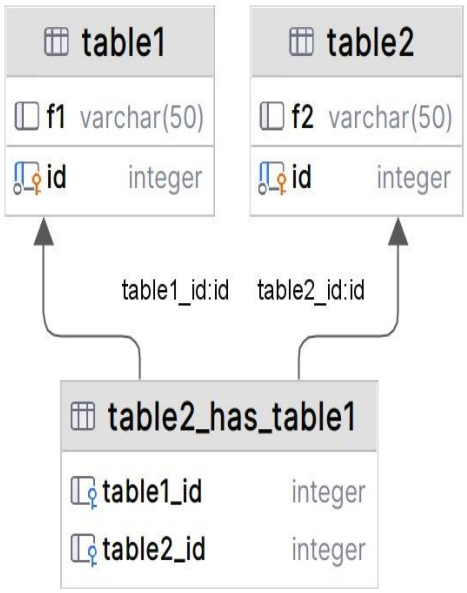

Рисунок 7 — Схема 6.

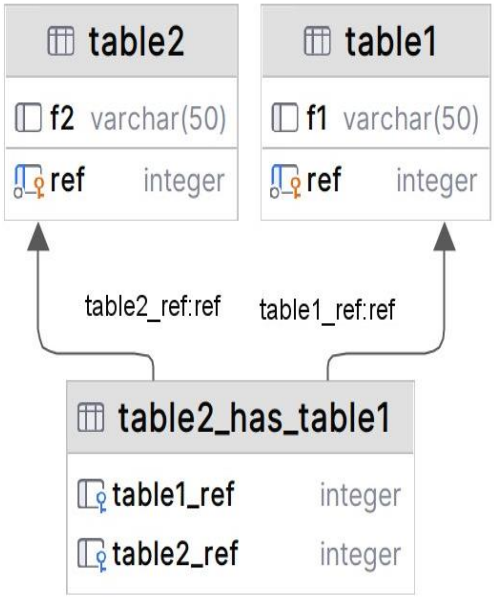

Рисунок 6 — Схема 7.

Листинг 60 — Запроса к РСУБД, относящейся к листингу 59 для *«endpointGetTable1ManyToManyTable2».*

```
SELECT *FROM table2 WHERE ID IN (
SELECT TABLE2_ID FROM table2_has_table1
WHERE TABLE1_ID IN (SELECT ID FROM table1));
```

Листинг 61 — Запроса к РСУБД, относящейся к листингу 59 для *«endpointGetTable2ManyToManyTable1».*

```
SELECT *FROM table1 WHERE ID IN (
SELECT TABLE1_ID FROM table2_has_table1
WHERE TABLE2_ID IN (SELECT ID FROM table2));
```

Схема базы данных, к которой относится запрос из листинга 62, изображена на рисунке 6. Запрос к РСУБД (листинг 63–64).

Листинг 62 — Пример указания связей между таблицами.

```
{
"Table1Group":{
   "pseudonyms": {
    "joins": {
      "table1:table2": ["table2_has_table1"],
      "table2:table1_has_table2": ["ref","table2_ref"],
      "table1:table1_has_table2": ["ref","table1_ref"]
    }
   },
   "http":{
      "endpointGetTable1ManyToManyTable2Ref":{
      "type":"get",
      "request":"table1/table2"
    },
      "endpointGetTable2ManyToManyTable1Ref":{
      "type":"get",
      "request":"table2/table1"
    }
   }
  }
}
```

Листинг 63 — Запроса к РСУБД, относящейся к листингу 62. Для *«endpointGetTable1ManyToManyTable2Ref»*.

```
SELECT *FROM table2 WHERE ref IN (
SELECT TABLE2_ref FROM table2_has_table1
WHERE TABLE1_ref IN (SELECT ref FROM table1));
```

Листинг 64 — Запроса к РСУБД, относящейся к листингу 62. Для *«endpointGetTable2ManyToManyTable1Ref»*.

```
SELECT *FROM table1 WHERE ref IN (
SELECT TABLE1_ref FROM table2_has_table1
WHERE TABLE2_ref IN (SELECT ref FROM table2));
```

Связь по умолчанию

Если связь между таблицами по умолчанию, или одно из полей по умолчанию, то следует обозначить лишь факт ее наличия «:».

Листинг 65 — Пример указания связей между таблицами.

```
{
"Table1Group":{
   "pseudonyms": {
    "joins": {
      "table1:table2": [":",":"]
    }
   },
   "http":{
     "endpointGetRef":{
      "type":"get",
      "request":"table1/table2"
    }
   }
  }
}
```

Дейкстра

Листинг 66 — Поиск связей.

```
{
"Table1Group":{
   "pseudonyms": {
    "joins": {
```

```
        "table1:table2": [">"],
        "table2:table3": ["<"],
        "table3:table4": ["table3_has_table4"],
        "table4:table5": [">"]
      }
    },
    "http":{
      "endpointGetDeicstra":{
        "type":"get",
        "request":"table1/table5"
      },
      "endpointGetNoDeicstra":{
        "type":"get",
        "request":"table1/table2/table3/table4/table5"
      }
    }
  }
}
```

Для удобства написания конечных точек был реализован алгоритм Дейкстры (описание находится в источнике: [20]). В случае указания и наличия связи в «*joins*» через различные “пропущенные таблицы” будет запущен алгоритм Дейкстры. В листинг 66 запрос к РСУБД в *«endpointGetDeicstra»* с *«endpointGetNoDeicstra»* будет одинаковым.

Односторонняя связь

По умолчанию связь между таблицами является двусторонней. Однако, если необходимо установить одностороннюю связь или связь, когда таблицы связаны через разные поля в разных направлениях, следует использовать символ «!» перед связью (листинг 67). Запрос к РСУБД для листинга 67 находится в листингах 68–69.

Листинг 67 — Пример указания связей между таблицами в одну сторону.

```
{
"Table1Group":{
    "pseudonyms": {
      "joins": {
        "!table1:table2": ["ref", "table1_ref"]
      }
    },
    "http":{
      "endpointGetOneRef":{
        "type":"get",
        "request":"table1/table2"
      },
      "endpointGetDefaultRef":{
        "type":"get",
        "request":"table2/table1"
      }
    }
  }
}
```

Листинг 68 — Запроса к РСУБД, относящейся к листингу 67. Для *«endpointGetOneRef».*

```
SELECT *FROM table2
WHERE table1_ref IN (
SELECT ref FROM table1);
```

Листинг 69 — *Запроса к РСУБД, относящейся к листингу 67. Для «endpointGetDafaultRef».*

```
SELECT *FROM table1
WHERE table2_id IN (
SELECT id FROM table2);
```

Связь через псевдонимы таблиц

Схема базы данных, к которой относится это запрос, изображена на рисунке 8. Связь можно указывать ни только через реальные имена таблицы, но и через её псевдонимы, задаваемые в параметрах *«tables»* (листинг 70) . Запрос к РСУБД для листинга 70 находится в листинге 71.

Однако стоит учитывать, что поиск связей или пропущенных таблиц будет вестись сначала через псевдонимы или псевдонима и реального имени, используемые в конечной точке. В случае неуспеха все этапы поиска связей пойдут через реальные имена таблицы.

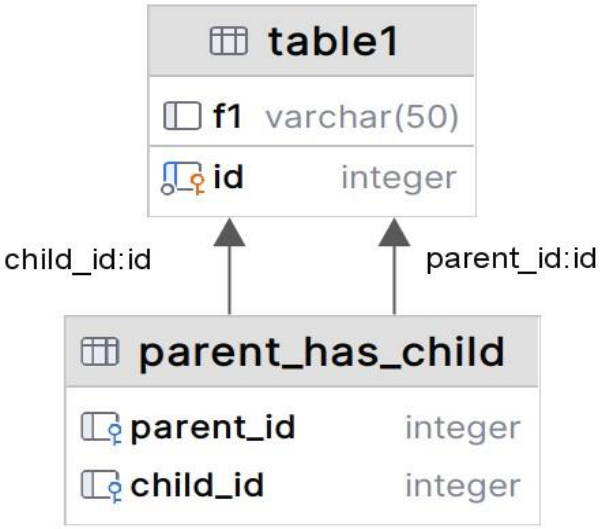

Рисунок 8 — Схема 8.

Листинг 70 — Пример указания связей между таблицами через их псевдонимы.

```
{
"Table1Group":{
   "pseudonyms": {
    "tables": {
      "table1": ["parent","child"]
    },
    "joins":{
      "parent:child":["parent_has_child"],
      "parent:parent_has_child":["id","parent_id"],
      "child:parent_has_child":["id","child_id"]
    }
   },
   "http":{
    "endpointGetPseudonymsRef":{
      "type":"get",
      "request":"parent/child"
    }
   }
  }
}
```

Листинг 71 — Запроса к РСУБД, относящейся к листингу 70. Для *«endpointGetPseudonymsRef»*.

```
SELECT *FROM table1 as “child”
WHERE ID IN (
SELECT chield_id FROM
parent_has_chield
WHERE parent_id IN (
```

```
SELECT ID FROM table1 as “parent”));
```

«tables»

При использовании восклицательного знака перед псевдонимом, для таблицы все поля, в случае если они заданы по умолчанию, генерируются в формате <Псевдоним>_id. Таким образом вместо «*Table1_id»* сгенерировалось *«Table_id»;*

Листинг 72 — Пример использования псевдонимов для таблицы.

```
{
"Table1Group":{
   "pseudonyms": {
    "tables": {
      "table1": ["t1","!Table"]
    }
   },
   "http":{
    "endpointGet":{
      "type":"get",
      "request":"t1/Table2"
    },
     "endpointGetRef":{
      "type":"get",
      "request":"Table/Table2"
    }
   }
 }
}
```

Листинг 73 — Запроса к РСУБД, относящейся к листингу 72. Для *«endpointGet».*

```
SELECT * FROM TABLE2
WHERE Table1_id IN (
SELECT ID FROM Table1 as 'T1');
```

Листинг 74 — Запроса к РСУБД, относящейся к листингу 72. Для *«endpointGetRef».*

```
SELECT * FROM TABLE2
WHERE Table_id IN
(SELECT ID FROM Table1 as 'Table');
```

«fields»

Листинг 75 — Пример использования псевдонимов для полей.

```
{
"Table1Group":{
   "pseudonyms": {
    "fields": {
      "Field1": ["F1","someF"]
    }
   },
   "http":{
    "endpointGet":{
      "get":"someF",
      "request":"Table1/{F1-s}"
    },
     "endpointPost":{
      "post":"someF->F1",
      "request":"Table1"
    }
   }
    }
}
```

Листинг 76 — *Запроса к РСУБД, относящейся к листингу 75. Для «endpointGet».*

```
SELECT FIELD1 as 'someF'
FROM TABLE1 WHERE FIELD1=?;
```

Листинг *77— Запроса к РСУБД, относящейся к листингу 75. Для «endpointPost».*

```
INSERT INTO TABLE1 (Field1) VALUES(?)
```

```
RETURNING Field1 as 'F1';
```

При этом *«Field1»* передается в теле запроса как *«someF»*. Псевдонимы для полей можно использовать в фильтрах, теле запроса, в конечной точке. Однако нельзя использовать в *«joins»*. В них можно указывать только настоящие названия полей.

«entity»

Код, приведенный в листингах 78 и 79 является равноценным.

Листинг 78 — Пример использования *«entity»*.

```
 {
"Table5Group":{
   "pseudonyms": {
    "entity": {
      "entity1": ["F1-s=str","F2-i=6","F3-i=10"],
      "entity2": ["F4-i=13"]
    }
   },
   "http":{
    "endpoint":{
      "patch":"entity1|entuty2|F5-i=19->entity2|id",
      "request":"Table5/{F5-s}"
    }
   }
  }
}
```

Листинг 79 — Пример отсутствия *«entity»*.

```
{
"Table5Group":{
  "http":{
    "endpoint":{
      "patch":"F1-s=str|F2-i=6|F3-i=10|F4-i=13|F5-i=19->F4|id",
      "request":"Table5/{F5-s}"
    }
   }
  }
}
```

«entity»- можно использовать в качестве тела запроса для: *POST, PATCH, PUT* и возвращаемых значений для *POST, PATCH, PUT, GET*. В качестве дополнительных параметров (*«limit|sort|fields»*) для *GET, «entity»* использовать нельзя.

10. Фильтры

Фильтры могут быть либо на уровне контроллера, либо на уровне конечной точки. При одном и том же имени, фильтр, находящийся в конечной точке, будет иметь преимущество. Код функции фильтра будет сгенерирован в том случае, если фильтр будет присутствовать в *«request»*.

Фильтр имеет формат (листинг 80).

Листинг 80 — Формат фильтра.

```
 "FilterName:type=Value":"someInfo"
```

«=Value» - не обязательно и в случае его присутствия оно будет в *«request»* вместо *«[filterName]»*.

Листинг 81 — Пример использования фильтра.

```
"GroupEndpoint": {
   "filters": {
     "filterCall:call": "org.example.filter#CallClass#make1",
     "filterAnd:and": "field1|field2",
     "filterOr:or": "field1|field2",
     "filterNotAnd:!and": "field1|field2",
```

```
      "filterNotOr:!or": "field1|field2"
    },
    "http": {
      "Endpoint": {
        "request": "table/[filterCall]|[filterAnd]/[filterOrOne]",
        "filters": {
            "filterCall:call": "org.example.filter#CallClass#make2",
            "filterAndOne:1-and": "field1|field2",
            "filterOrOne:1-or": "field1|field2",
            "filterNotAndOne:!1-and": "field1|field2",
            "filterNotOrOne:!1-or": "field1|field2"
        }
      }
    }
}
```

Фильтр «:call»

Листинг 82 — Пример фильтра *«:call»*.

```
"filters":{
"filterCall:call":"org.example.filter#CallClass#make"
}
```

В нужном месте запроса будет вызвана статическая функция у класса, прототип которой находится в листинге 83.

Листинг 83 — Прототип функции для фильтра из листинга 82.

```
package org.example.filter;
public record CallClass() {
public static Condition make(
MultiValueMap<String,String> filterCall,
String table){
  return //
   }
}
```

В запросе параметр *«MultiValueMap<String, String> filterCall»* будет иметь имя фильтра. В данном случае *«filterCall»*.

Фильтр «:and»

Листинг 84 — Пример фильтра *«:and»*.

```
"filters": {
    "filterAnd:and": "like_field1-s|field2"
}
```

Для листинга 84 будет сгенерирована и вызвана в нужном месте функция, которая находится в листинге 85.

Листинг 85 — Прототип функции для фильтра из листинга 84.

```
public Condition EndpointFilterAndOfEndpoint(
MultiValueMap<String, String> returnFields,
      String table, Condition defaultCondition) {
    List<Condition> conditions=new ArrayList<>();
    if (returnFields.containsKey("like_field1")) {
      conditions.add(DSL.field("field1")
      .like(DSL.val(returnFields.getFirst("like_field1"),
      String.class)));
    }
    if (returnFields.containsKey("field2")) {
      conditions.add(DSL.field("field2")
      .eq(DSL.val(returnFields.getFirst("field2"),
      Long.class)));
    }
    return conditions.stream().reduce(Condition::and)
        .ofNullable(defaultCondition).get();
  }
```

Фильтр *«:or»*

Листинг 86 — Пример фильтра *«:or»*.

```
 "filters": {
     "filterOr:or": "field1|field2"
}
```

Для листинга 86 будет сгенерирована и вызвана в нужном месте функция, которая находится в листинге 87.

Листинг 87 — Прототип функции для фильтра из листинга 86.

```
public Condition EndpointFilterOrOfEndpoint(
 MultiValueMap<String, String> returnFields,
     String table, Condition defaultCondition) {
   List<Condition> conditions=new ArrayList<>();
   if (returnFields.containsKey("field1")) {
    conditions.add(DSL.field("field1")
    .eq(DSL.val(returnFields.getFirst("field1"),
    String.class)));
   }
   if (returnFields.containsKey("field2")) {
    conditions.add(DSL.field("field2")
    .eq(DSL.val(returnFields.getFirst("field2"),
    Long.class)));
   }
   return conditions.stream().reduce(Condition::or)
       .ofNullable(defaultCondition).get();
 }
```

Фильтр «:!and»

Листинг 88 — Пример фильтра *«:!and»*.

```
"filters": {
     "filterNotAnd:!and": "field1|field2"
}
```

Для листинга 88 будет сгенерирована и вызвана в нужном месте функция, которая находится в листинге 89.

Листинг 89 — Прототип функции для фильтра из листинга 88.

```
public Condition EndpointFilterNotAndOfEndpoint(
 MultiValueMap<String, String> returnFields,
     String table, Condition defaultCondition) {
   List<Condition> conditions=new ArrayList<>();
   if (returnFields.containsKey("field1")) {
    conditions.add(DSL.field("field1")
    .eq(DSL.val(returnFields.getFirst("field1"), String.class)));
   }
   if (returnFields.containsKey("field2")) {
    conditions.add(DSL.field("field2")
    .eq(DSL.val(returnFields.getFirst("field2"),  Long.class)));
   }
   if(conditions.isEmpty()) {
    return defaultCondition;
   }
   return DSL.not(conditions.stream().reduce(Condition::and)
       .ofNullable(defaultCondition).get());
}
```

Фильтр *«:!or»*

Листинг 90 — Пример фильтра *«:!or»*.

```
"filters": {
     "filterOr:!or": "field1|field2"
}
```

Для листинга 90 будет сгенерирована и вызвана в нужном месте функция, которая находится в листинге 91.

Листинг 91 — Прототип функции для фильтра из листинга 90.

```
public Condition EndpointFilterNotOrOfEndpoint(
 MultiValueMap<String, String> returnFields,
      String table, Condition defaultCondition) {
    List<Condition> conditions=new ArrayList<>();
    if (returnFields.containsKey("field1")) {
      conditions.add(DSL.field("field1")
      .eq(DSL.val(returnFields.getFirst("field1"),
     String.class)));
    }
    if (returnFields.containsKey("field2")) {
      conditions.add(DSL.field("field2")
      .eq(DSL.val(returnFields.getFirst("field2"),
      Long.class)));
    }
    if(conditions.isEmpty()) {
      return defaultCondition;
    }
    return DSL.not(conditions.stream().reduce(Condition::or)
        .ofNullable(defaultCondition).get());
  }
```

Фильтр «:1-and»

Листинг 92 — Пример фильтра *«:1-and»*.

```
"filters": {
      "filterAndOne:1-and": "field1|field2"
}
```

Для листинга 92 будет сгенерирована и вызвана в нужном месте функция, которая находится в листинге 93.

Листинг 93 — Прототип функции для фильтра из листинга 92.

```
public Condition EndpointFilterAndOneOfEndpoint(
 MultiValueMap<String, String> returnFields,
      String table, Condition defaultCondition) {
    List<Condition> conditions=new ArrayList<>();
    if(returnFields.containsKey("filterAndOne")) {
    conditions.add(DSL.field("field1")
    .eq(DSL.val(
    returnFields.getFirst("filterAndOne"),
    Long.class)));
      conditions.add(DSL.field("field2")
      .eq(DSL.val(
      returnFields.getFirst("filterAndOne"),
      Long.class)));
    }
    return conditions.stream().reduce(Condition::and)
        .ofNullable(defaultCondition).get();
  }
```

Фильтр «:1-or»

Листинг 94 — Пример фильтра *«:1-or»*.

```
"filters": {
      "filterOrOne:1-or": "field1|field2"
}
```

Для листинга 94 будет сгенерирована и вызвана в нужном месте функция, которая находится в листинге 95.

Листинг 95 — Прототип функции для фильтра из листинга 94.

```
  public Condition EndpointFilterOrOneOfEndpoint(
  MultiValueMap<String, String> returnFields,
      String table, Condition defaultCondition) {
    List<Condition> conditions=new ArrayList<>();
    if(returnFields.containsKey("filterOrOne")) {
```

```
      conditions.add(DSL.field("field1")
      .eq(DSL.val(
      returnFields.getFirst("filterOrOne"),
      Long.class)));
      conditions.add(DSL.field("field2")
      .eq(DSL.val(
      returnFields.getFirst("filterOrOne"),
      Long.class)));
    }
    return conditions.stream().reduce(Condition::or)
        .ofNullable(defaultCondition).get();
  }
```

Фильтр «:!1-and»

Листинг 96 — Пример фильтра *«:!1-and»*.

```
"filters": {
    "filterNotAndOne:!1-and": "field1|field2"
}
```

Для листинга 96 будет сгенерирована и вызвана в нужном месте функция, которая находится в листинге 97.

Листинг 97 — Прототип функции для фильтра из листинга 96.

```
   public Condition EndpointFilterNotAndOneOfEndpoint(
   MultiValueMap<String, String> returnFields,
      String table, Condition defaultCondition) {
    List<Condition> conditions=new ArrayList<>();
    if(returnFields.containsKey("filterNotAndOne")) {
      conditions.add(DSL.field("field1")
      .eq(DSL.val(
      returnFields.getFirst("filterNotAndOne"),
      Long.class)));
      conditions.add(DSL.field("field2")
      .eq(DSL.val(
      returnFields.getFirst("filterNotAndOne"),
      Long.class)));
    }
    if(conditions.isEmpty()) {
      return defaultCondition;
    }
    return DSL.not(conditions.stream().reduce(Condition::and)
        .ofNullable(defaultCondition).get());
  }
```

Фильтр «:!1-or»

Листинг 98 — Пример фильтра *«:!1-or»*.

```
"filters": {
    "filterNotOrOne:!1-or": "field1|field2"
}
```

Для листинга 98 будет сгенерирована и вызвана в нужном месте функция, которая находится в листинге 99.

Листинг 99 — Прототип функции для фильтра из листинга 98.

```
   public Condition EndpointFilterNotOrOneOfEndpoint(
  MultiValueMap<String, String> returnFields,
      String table, Condition defaultCondition) {
    List<Condition> conditions=new ArrayList<>();
    if(returnFields.containsKey("filterNotOrOne")) {
      conditions.add(DSL.field("field1")
     .eq(DSL.val(
      returnFields.getFirst("filterNotOrOne"),
      Long.class)));
      conditions.add(DSL.field("field2")
      .eq(DSL.val(
```

```
      returnFields.getFirst("filterNotOrOne"),
      Long.class)));
    }
    if(conditions.isEmpty()) {
      return defaultCondition;
    }
    return DSL.not(conditions.stream().reduce(Condition::or)
        .ofNullable(defaultCondition).get());
  }
```

11. Пути к JSON файлу

Для удобного формата файла JSON, была реализована подгрузка информации из других файлов, используемых в таких местах, как группа конечных точек поле *«filters», «http», «pseudonyms2*. Благодаря этому один большой файл (листинг 100) можно разделить на файлы меньшего размера (листинг 101–108).

Листинг 100 — Пример не разделенного файла.

```
{
    "Group1": {
      "pseudonyms": {
        "tables": {
          "t1": ["table1"]
        },
        "fields": {
          "ref": ["id"]
  }
      },
      "filters": {
        "f1:call": "some#ClassFilter#callFunc"
      },
      "http": {
        "endpoint1": {
          "request": "table1/{ref}/[f1]"
        }
      }
    },
    "Group2": {
      "pseudonyms": {
        "tables": {
          "t1": ["table1"]
        },
        "fields": {
          "ref": ["id"]
        }
      },
      "filters": {
        "f1:call": "some#ClassFilter#callFunc"
      },
      "http": {
        "endpoint1": {
          "request": "table2/[f1]",
          "filters": {
            "f1:call": "some#ClassFilter#callFunc2"
          }
        }
      }
    }
}
```

Листинг 101 — «*MainFile.json».*

```
{
    "Group1": "Group1.json",
    "Group2": "Group2.json"
```

```
}
```

Листинг 102 — Group1.json.

```
{
     "pseudonyms": "Pseudonyms.json",
     "filters": "filtersInGroup.json",
     "http":  "httpOfGroup1.json"
 }
```

Листинг 103 — *«Group2.json»*.

```
{
     "pseudonyms":"Pseudonyms.json",
     "filters":"filtersInGroup.json",
     "http":"httpOfGroup2.json"
}
```

Листинг 104 — *«Pseudonyms.json»*.

```
{
   "tables": {
   "t1": ["table1"]
   },
   "fields": {
    "ref": ["id"]
   }
}
```

Листинг 105 — *«httpOfGroup1.json»*.

```
{
   "endpoint1": {
   "request": "table1/{ref}/[f1]"
   }
}
```

Листинг 106 — *«httpOfGroup2.json»*.

```
{
   "endpoint1": {
   "request": "table2/[f1]",
   "filters":filtersInEndpontOfGroup2.json
   }
}
```

Листинг 107 — *«filtersInGroup.json»*.

```
{
   "f1:call": "some#ClassFilter#callFunc"
}
```

Листинг 108 — *«filtersInEndpontOfGroup2.json»*.

```
{
   "f1:call": "some#ClassFilter#callFunc2"
}
```

Необходимо отметить, что в приведенном примере не указаны полные пути к файлам. Однако в реальной практике они должны быть указаны полными.

12. Файл: *«application.properties»*

Помимо стандартных параметров *Spring-Boot* приложения файла *«application.properties»* (листинг 109).

Листинг 109 — *«application.properties»*. Стандартные настройки.

```
spring.application.name=RestApiApplication
server.port=8080
spring.datasource.url=jdbc:postgresql://localLost:5433/postgres
spring.datasource.username=postures
spring.datasource.password=password
spring.jooq.sql-dialect=POSTGRES
```

Были добавлены дополнительные настройки, используемые в сгенерированном коде (листинг 110).

Листинг 110 — *«application.properties».* Дополнительные настройки.

```
restApi.openApi.title=RestApi
restApi.openApi.description=Manager
restApi.openApi.version=1.0
restApi.showSql=true
```

Параметр из листинга 111 отвечает за показ кода, отправляемого к базе данных.

Листинг 111 — Показ SQL.

```
restApi.showSql
```

Параметры из листинга 112 используются при конфигурации *Swagger*.

Листинг 112 — Конфигурации *Swagger*.

```
restApi.openApi.title
restApi.openApi.description
restApi.openApi.version
```

13. Configuration

Код сгенерированного класса конфигурации находится в листинге 113.

Листинг 113 — *«Configuration».*

```
@Configuration
public class ConfigRest {
  @Bean
  public OpenAPI usersMicroserviceOpenAPI
 (@Value("${restApi.openApi.title:}") String title,
  @Value("${restApi.openApi.description:}") String description,
  @Value("${restApi.openApi.version:}") String version) {
    return new OpenAPI()
    .info(new Info()
    .title(title)
    .description(description)
    .version(version));
  }
}
```

«Operation»

Для более красивого обозначения в конечных точках для каждого метода в контроллере добавляется аннотация (листинг 114).

Листинг 114 — Operation.

```
@Operation(summary="")
```

Была реализована возможность добавления таких параметров как *«tags»* и «summary».

«tags» и «summary»

Параметры «tags» и «summary» (листинг 115–116).

Листинг 115 — «tags» и «summary».

```
"EndpointName7": {
    "types":{
        "get":{
        "summary":"Описание",
        "tags":"tag1|tag2"
        }
    },
    "request": "endpoint"
}
```

Листинг 116 — «tags» и «summary».

```
"EndpointName7": {
    "type":"get",
```

```
    "summary":"Описание",
    "tags":"tag1|tag2",
    "request": "endpoint"
}
```

Сгенерированный код для листинга 115 и 116 будет представлен в листинге 117.

Листинг 117 — Код для листинга 115 или 116.

```
@Operation (summary="Описание", tags={"tag1","tag2"})
```

В случае следующего формата данных для листинга 118 невозможно указать *«summary»,* но возможно указать в *«tags».*

Листинг 118 — *«tags”.*

```
"EndpointName7": {
    "get":"",
    "delete":"",
    "tags":"tag1|tag2",
    "request": "endpoint"
}
```

Сгенерированный код для листинга 115–116 и 118 представлен в листинг 119.

Листинг 119 — Код для листинга 118.

```
@Operation(summary="", tags={"tag1","tag2"})
```

14. Компиляция

В случае успеха компиляции, будет выведена информация о сгенерированных файлах (рисунок 10).

В случае некорректного JSON файла, будет выведена информация о месте предполагаемой ошибки (рисунок 9).

Рисунок 9 — Не корректный JSON файл.

```
-GENERATE REPOSITORY-org.example.repository.AuthorizationRepository
-GENERATE CONTROLLER-org.example.controller.AuthorizationController
-GENERATE REPOSITORY-org.example.repository.AuthorRepository
-GENERATE CONTROLLER-org.example.controller.AuthorController
-GENERATE REPOSITORY-org.example.repository.CommitsRepository
-GENERATE CONTROLLER-org.example.controller.CommitsController
-GENERATE REPOSITORY-org.example.repository.ReaderRepository
-GENERATE CONTROLLER-org.example.controller.ReaderController
-GENERATE SWAGGER CONFIG-org.example.config.ConfigRest
Compiling.....
Compiling.....
Compiling.....
Compiling.....
Compiling.....
Compiling.....
```

Рисунок 10 — Компиляция прошла успешно.