Loginom

Синтез сценариев Loginom кодом на Python

Оленчиков Д.М., к.ф.-м.н., ведущий эксперт ООО «Союзгеосервис»

Loginom

Зачем low-code платформе классический код?

Айсберг опыта

Задачи	Срок освоения
Изучение среды разработки или визуального проектирования, синтаксиса языка программирования	Дни, недели
Применение библиотек для решения типовых задач	Недели, месяцы
Построение эффективной архитектуры, понимание алгоритмов и методов, осознание границ применимости	Месяцы, годы

С глубиной меняется уровень задач и исчезает разница между low-code и классическим программированием

Сила и слабость low-code подхода

Достоинства

- Низкий порог входа
- Эффективность решения типовых задач
- Визуализация промежуточных расчетов
- Наглядность и удобство отладки

Недостатки

- Трудности развития готовых сценариев
- Ограниченный набор типов данных
- Недостаточная гибкость
- Проблемы с контролем версий, автоматизацией…

Идея преодоления недостатков

Создание сценариев кодом на Python позволит пользователям умеющим программировать расширить возможности:

1. Повысить гибкость
2. Облегчить решение сложных задач
3. Обеспечить контроль версий
4. Реализовать интеграцию с git
5. Генерировать сценарии, в том числе с помощью ИИ

Low-code пользователи продолжат работать как обычно.

Loginom

Генерация сценариев

Техническая реализация:

- Пакет Loginom – папка с xml-файлами, упакованная zip-ом
- Разработана библиотека loginom.py, содержащая необходимые классы
- Созданный пакет загружается в файловое хранилище Loginom и запускается

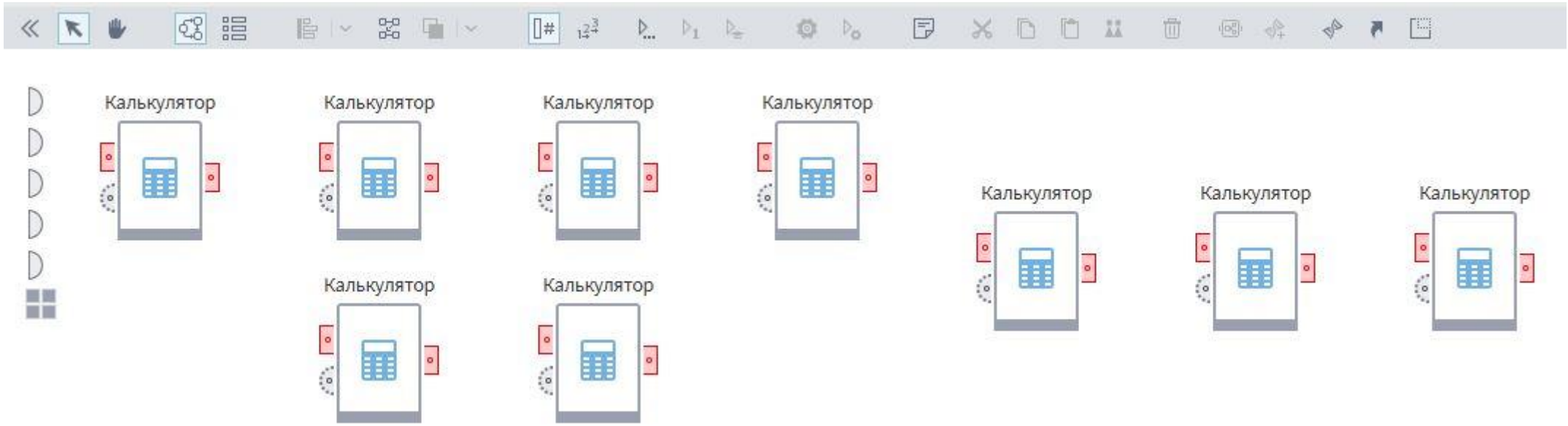
```
from loginom import *
#импорт библиотеки

class MyScenaryBuilder(LGBuilder):
    def build(self, sc):
        ...   # описание сценария

MyScenaryBuilder() #генерация .lgp-файла
```

Узлы располагаются внутри боксов:

- тело сценария
- внутренность подмодели
- горизонтальный HBox
- вертикальный VBox

```
from loginom import *

class BoxBuilder(LGBuilder):
    def build_elems(self, box, n):
        for i in range(n):
            calc = Calc(box)

    def build(self, sc):
        v = VBox(sc)
        h1 = HBox(v)
        h2 = HBox(v)
        h3 = HBox(sc)
        self.build_elems(h1, 4)
        self.build_elems(h2, 2)
        self.build_elems(h3, 3)

BoxBuilder()
```

Размещение узлов в боксах – результат



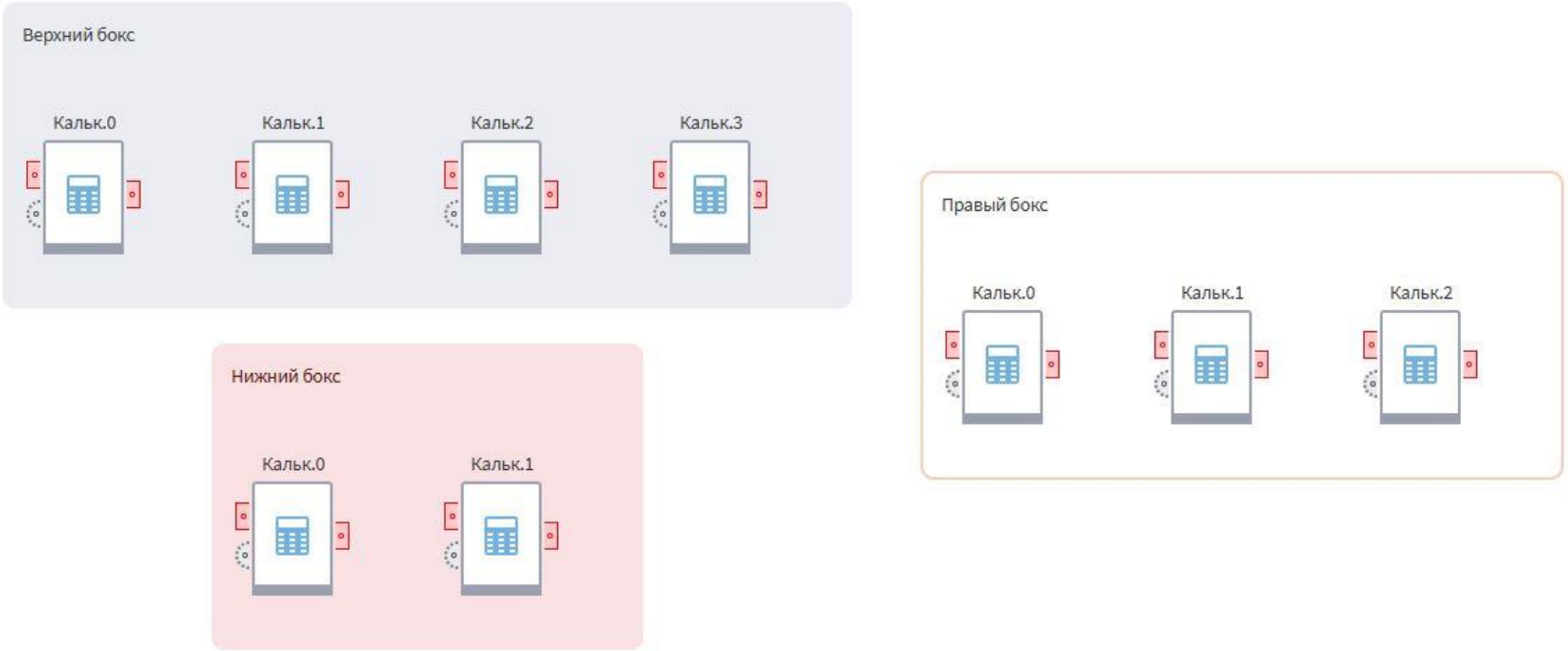
Боксы, аннотация, имена

```
from loginom import *

class BoxBuilder(LGBuilder):
    def build_elems(self, box, n):
        for i in range(n):
            calc = Calc(box, disp_name="Кальк."+str(i))

    def build(self, sc):
        v = VBox(sc)
        h1 = HBox(v, annotation="Верхний бокс")
        h2 = HBox(v, annotation="Нижний бокс", style=0)
        h3 = HBox(sc, annotation="Правый бокс", style=8)
        self.build_elems(h1, 4)
        self.build_elems(h2, 2)
        self.build_elems(h3, 3)

BoxBuilder()
```

Имя узла

Текст заметки и стиль

Размещение заметок в боксах – результат



Тонкая настройка, комментарии

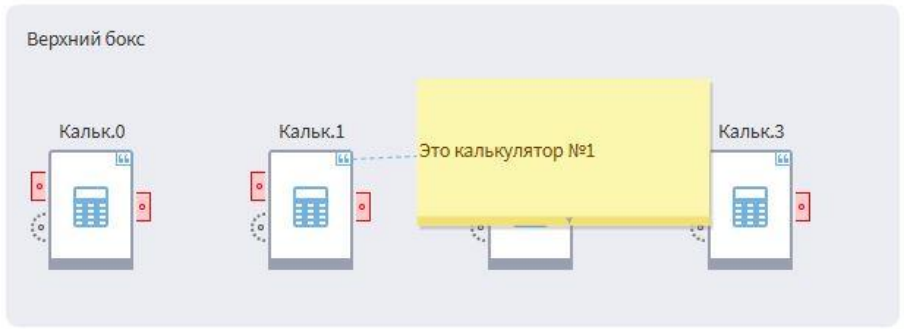
```
from loginom import *

class BoxBuilder(LGBuilder):
    def build_elems(self, box, n):
        for i in range(n):
            calc = Calc(box, disp_name="Кальк."+str(i))
            calc.comment("Это калькулятор №"+str(i))

    def build(self, sc):
        v = VBox(sc, skip_y=250)
        h1 = HBox(v, annotation="Верхний бокс")
        h2 = HBox(v, annotation="Нижний бокс", style=0, margin_x=-150)
        h3 = HBox(sc, annotation="Правый бокс", style=8)
        self.build_elems(h1, 4)
        self.build_elems(h2, 2)
        self.build_elems(h3, 3)

BoxBuilder()
```

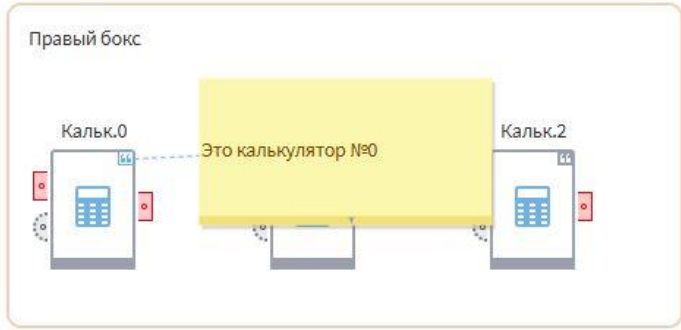
Добавление комментария

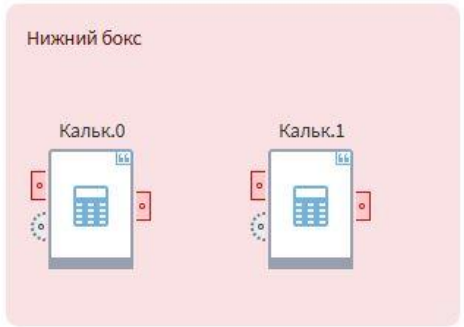
Увеличение зазора

Сдвиг бокса влево

Тонкая настройка и комментарии – результат







Пример создания сценария кодом

Постановка задачи

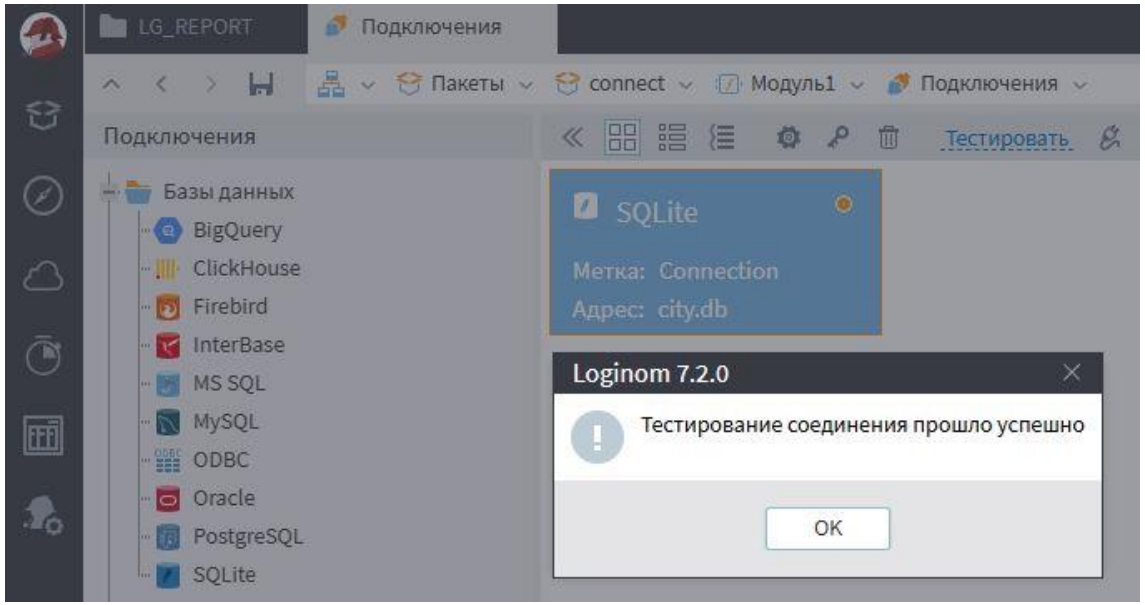
Дана таблица координат 42 городов России (http://ptlab.mccme.ru). Задача – для каждого из городов найти ближайший к данному по широте, долготе, альтитуде (высоте над уровнем моря).

G018 **Координаты городов России**

	Широта				Долгота				Высота над уровнем моря
	В десятичных градусах	В градусах, минутах и секундах			В десятичных градусах	В градусах, минутах и секундах			
		Градусы (°)	Минуты (′)	Секунды (″)		Градусы	Минуты	Секунды	
Астрахань	46.35	46	20	58.9	48.04	48	2	26.9	-13
Барнаул	53.36	53	21	38.2	83.76	83	45	49	189
Брянск	53.25	53	15	7.6	34.7	34	22	18.1	204
Владивосток	43.11	43	6	20.2	131.87	131	52	26.4	30
Волгоград	48.72	48	43	9.8	44.5	44	30	6.5	65
Воронеж	51.67	51	40	19.2	39.18	39	11	3.5	155
Екатеринбург	56.85	56	51	6.8	60.61	60	36	43.9	255

Код создания подключение к БД

```
from loginom import *

uuid_connection_str = "my_sqlite_connection_str"

class ConnectBuilder(LGBuilder):
    def build(self, sc):
        self.conn = sc.add_connection(
            conn_class=ConnectionSQLite,
            database="city.db",
            uuid_str = uuid_connection_str,
            visibility="public")

ConnectBuilder()
print(str_to_uuid(uuid_connection_str))
```



Guid `092db398-b169-64d9-4f38-91b177c7f896`

Код создания ссылки на подключение к БД и импорт таблицы

```
from loginom import *

sqlite_str = "my_sqlite_connection_str"
package_path = "connect.lgp"

class CityBuilder(LGBuilder):
    def add_ref(self, package):
        package.add_ref(package_path)

    def build(self, sc):
        box=sc
        ref_db = RefConnection(box, ref_uuid_str=sqlite_str)
        tab_city = ImportDBTable(box, table_name="city",
                                 name="Таблица городов")
        tab_city(ref_db)
        tab_city.set_columns(
            ["scity", "flatitude", "flongtitude", "faltitude"])

CityBuilder()
```

Добавление ссылки на пакет

Ссылка на подключение к БД

Импорт таблицы

Создание связи между узлами

Имена и типы столбцов

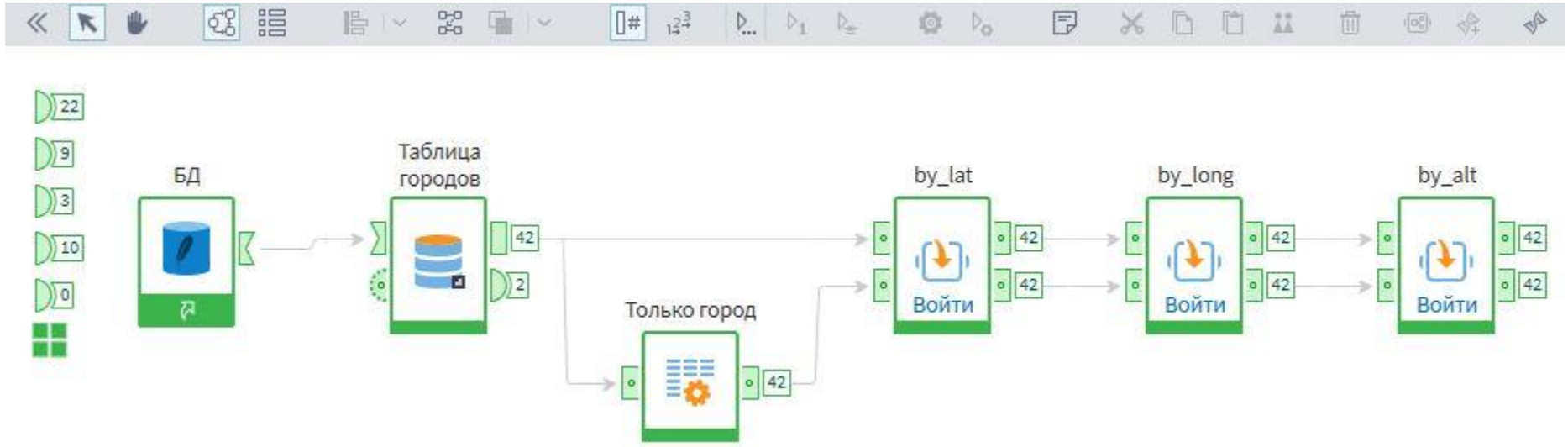
Ссылка на подключение к БД и импорт таблицы – результат

Таблица городов • Быстрый просмотр

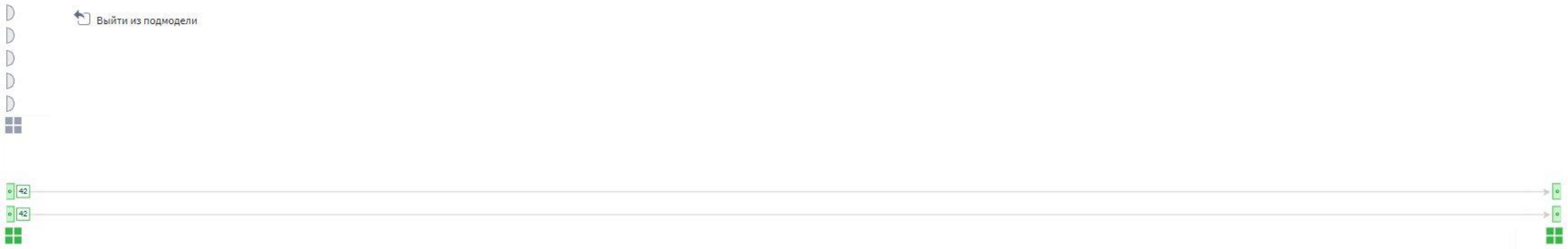
Набор данных | Статус выполнения

#	city	latitude	longtitude	altitude
1	Астрахань	46,35	48,04	-13
2	Барнаул	53,36	83,76	189
3	Брянск	53,25	34,70	204
4	Владивосток	43,11	131,87	30
5	Волгоград	48,72	44,50	65
6	Воронеж	51,67	39,18	155

Ожидаемый результат



В подмоделях пока что входы сразу идут на выход



```
def build(self, sc):
    box=sc
    ref_db = RefConnection(box, ref_uuid_str=sqlite_str)
    tab_city = ImportDBTable(box, table_name="city",
                             name="Таблица городов")
    tab_city(ref_db)
    tab_city.set_columns(
        ["scity", "flatitude", "flongtitude", "faltitude"])

    city_only = FieldParam(
        box, name="Только город", margin_y=80)
    city_only(tab_city)
    city_only.remove_columns(
        ["latitude", "longtitude", "altitude"])

    self.pred = (tab_city, city_only)

    self.calc_nearest(box, "latitude", "by_lat", "Ближ. по широте")
    self.calc_nearest(box, "longtitude", "by_long", "Ближ. по долготе")
    self.calc_nearest(box, "altitude", "by_alt", "Ближ. по высоте")


def calc_nearest(self, box, column, res_col, res_disp):
    sm = multitab_submodel(box, name=res_col, n_in=2, n_out=2)
    sm(self.pred[0], self.pred[1])
    self.pred = (sm[0], sm[1])

    box = sm.inner
    box(box[0], box[1])
```

Запоминаем узлы для создания связей

Создание подмоделей

Собственно создание подмодели

Запомнить выходы подмодели для последующих связей

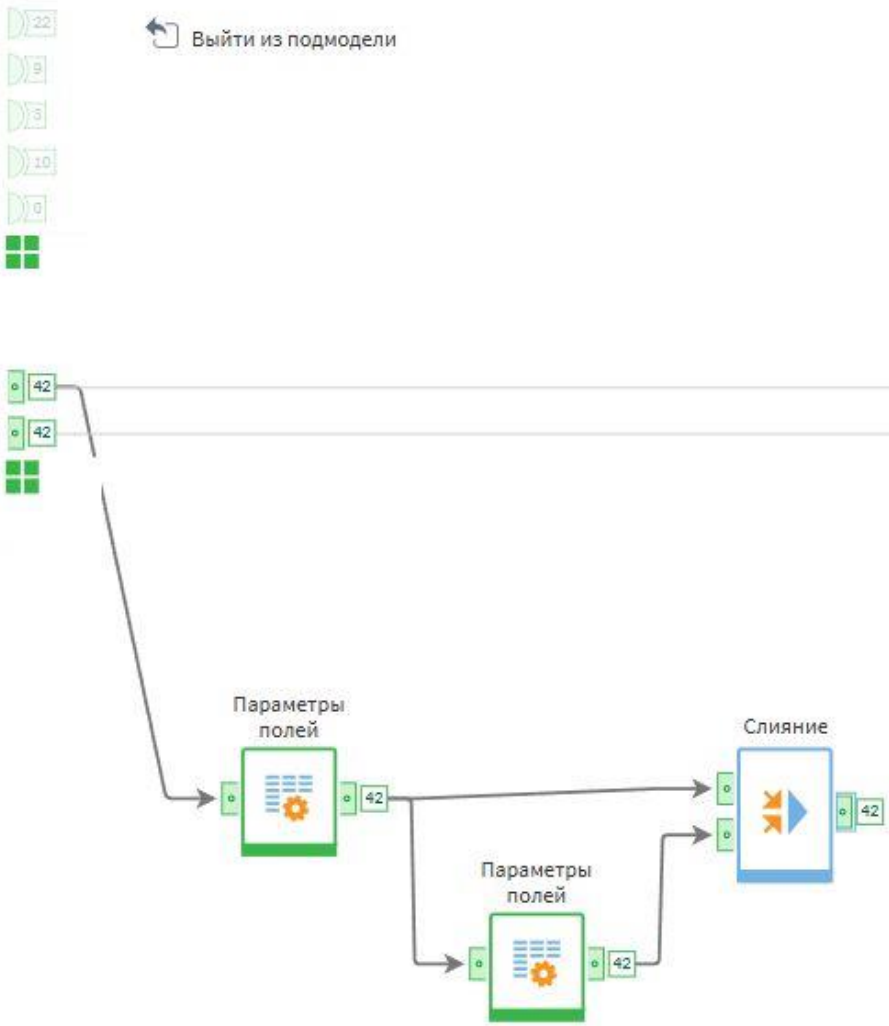
Внутренность подмодели

Соединить выходы со входами

Код создания подмодели и слияния

```
def calc_nearest(self, box, val_name, res_col, res_disp):
    sm = multitab_submodel(box, name=res_col, n_in=2, n_out=2)
    sm(self.pred[0], self.pred[1])
    self.pred = (sm[0], sm[1])

    box = sm.inner
    box(box[0], box[1])
    h = HBox(box, margin_y=200)

    tab_col = FieldParam(h)
    tab_col(box[0])
    # оставить только колонки "city" и val_name
    tab_col.remove_other_columns(["city", val_name])

    tab2 = FieldParam(h, margin_y=100)
    tab2(tab_col)
    # дописать к именам колонок "2"
    tab2.rename_columns([("city", "city2"),
                         (val_name, val_name+"2")])

    join = Join(h, join_type="Full")
    join(tab_col, tab2)
```

Создание подмодели и слияние – результат



Слияние • Быстрый просмотр

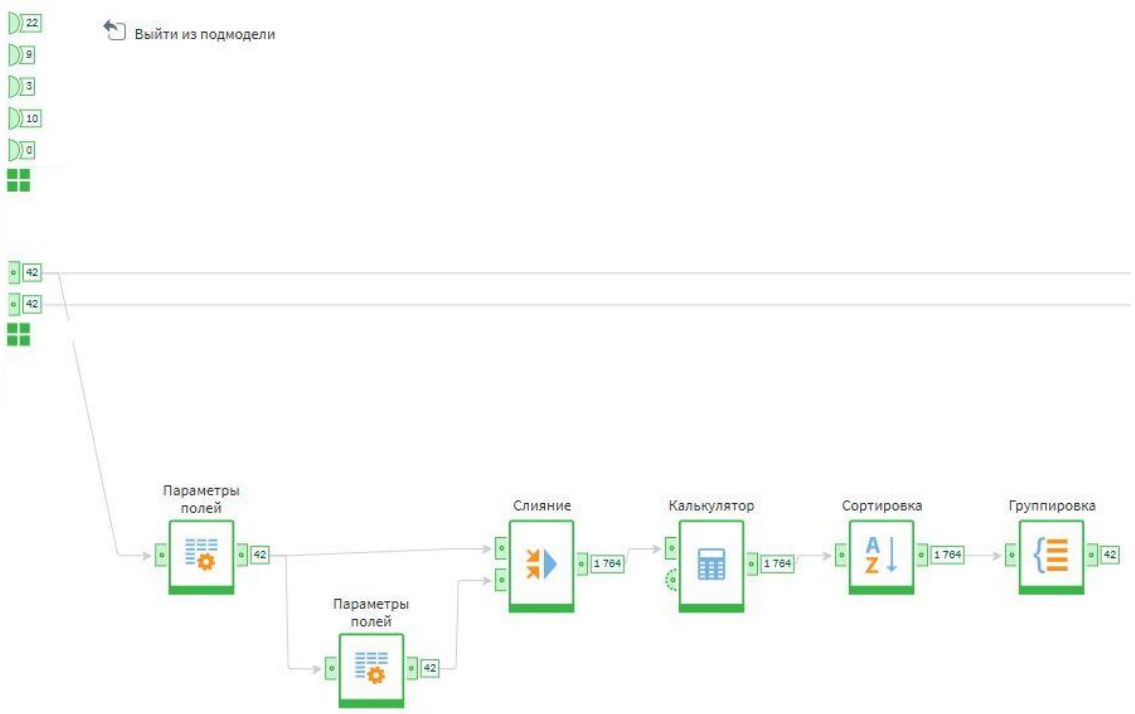
Выходной набор данных

#	city	latitude	city2	latitude2
1	Астрахань	46,35	Астрахань	46,35
2	Астрахань	46,35	Барнаул	53,36
3	Астрахань	46,35	Брянск	53,25
4	Астрахань	46,35	Владивосток	43,11
5	Астрахань	46,35	Волгоград	48,72
6	Астрахань	46,35	Воронеж	51,67
7	Астрахань	46,35	Екатеринбург	56,85
8	Астрахань	46,35	Иваново	57,00
9	Астрахань	46,35	Ижевск	56,85
10	Астрахань	46,35	Иркутск	52,30
11	Астрахань	46,35	Казань	55,79
12	Астрахань	46,35	Калининград	54,71
13	Астрахань	46,35	Кемерово	55,33
14	Астрахань	46,35	Киров	58,60

Код создания подмодели и группировка.

```
tab2 = FieldParam(h, margin_y=100)
tab2(tab_col)
# дописать к именам колонок "2"
tab2.rename_columns([("city", "city2"),
                     (val_name, val_name+"2")])

join = Join(h, join_type="Full")
join(tab_col, tab2)

calc = Calc(h)
calc(join)
calc.add("delta", dt_float, "abs({0:} - {0:}2)".format(val_name))

sort = Sort(h)
sort(calc)
sort.set_sort(["city", "delta"])

gr = Group(h)
gr(sort)
gr.add_group_names("city")
gr.add_value_names("city2", "First")
```

Создание подмодели и группировка – результат



Группировка • Быстрый просмотр

Выходной набор данных

#	city	city2_First
1	Астрахань	Астрахань
2	Барнаул	Барнаул
3	Брянск	Брянск
4	Владивосток	Владивосток
5	Волгоград	Волгоград
6	Воронеж	Воронеж
7	Екатеринбург	Екатеринбург
8	Иваново	Иваново

Получили тривиальный ответ. Нужна фильтрация.

Код добавления фильтрации

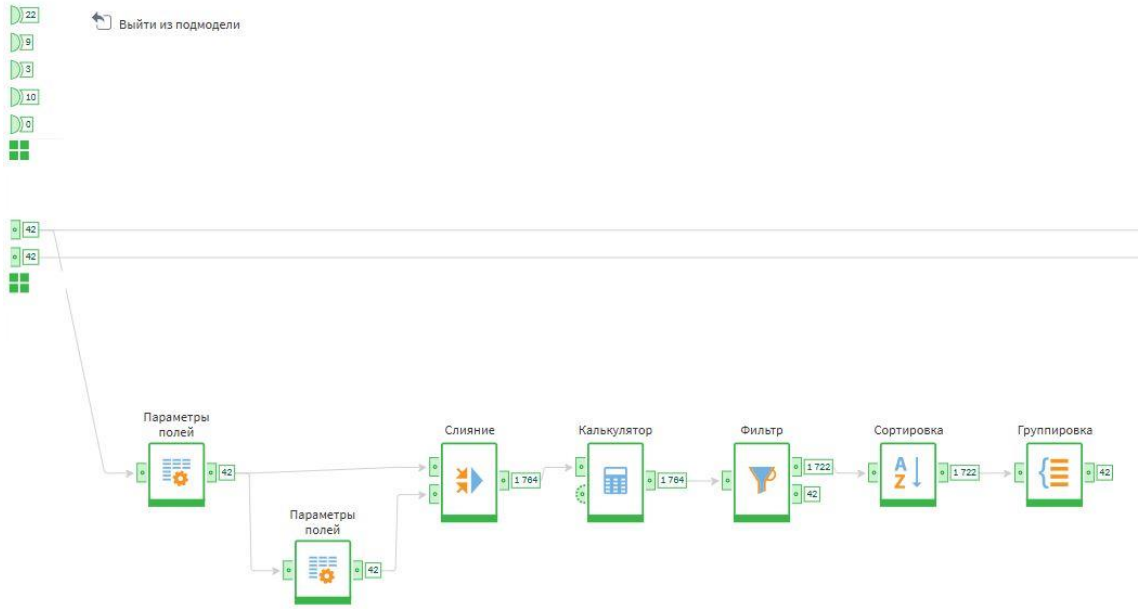
```
join = Join(h, join_type="Full")
join(tab_col, tab2)

calc = Calc(h)
calc(join)
calc.add("delta", dt_float, "abs({0:} - {0:}2)".format(val_name))
calc.add("b_diff", dt_bool, "city <> city2")

flt = Filter(h)
flt(calc)
flt.add("b_diff", "True")

sort = Sort(h)
sort(flt)
sort.set_sort(["city", "delta"])

gr = Group(h)
gr(sort)
gr.add_group_names("city")
gr.add_value_names("city2", "First")
gr.remove_suffix()
```

Признак для фильтрации

Добавляем фильтр

Убираем суффикс «_First»

Добавление фильтрации – результат



Группировка • Быстрый просмотр

Выходной набор данных

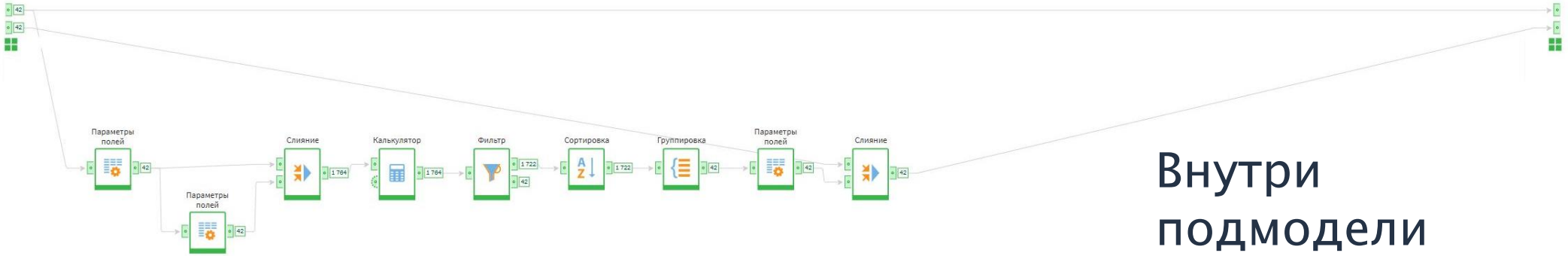
#	city	city2
1	Астрахань	Ростов-на-Дону
2	Барнаул	Брянск
3	Брянск	Пенза
4	Владивосток	Махачкала
5	Волгоград	Хабаровск
6	Воронеж	Оренбург
7	Екатеринбург	Ижевск
8	Иваново	Екатеринбург
9	Ижевск	Екатеринбург

Окончательный код создания подмодели

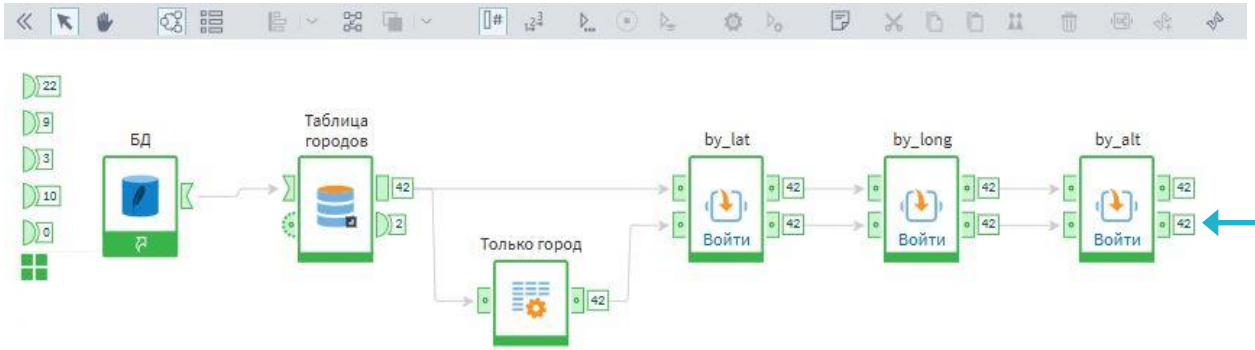
```
def calc_nearest(self, box, val_name, res_col, res_disp):
    sm = multitab_submodel(box, name=res_col, n_in=2, n_out=2)
    sm(self.pred[0], self.pred[1])
    self.pred = (sm[0], sm[1])

    box = sm.inner
    h = HBox(box, margin_y=200)

    tab_col = FieldParam(h)
    tab_col(box[0])
    # оставить только колонки "city" и val_name
    tab_col.remove_other_columns(["city", val_name])

    tab2 = FieldParam(h, margin_y=100)
    tab2(tab_col)
    # дописать к именам колонок "2"
    tab2.rename_columns([("city", "city2"),
                         (val_name, val_name+"2")])

    join = Join(h, join_type="Full")
    join(tab_col, tab2)

    calc = Calc(h)
    calc(join)
    calc.add("delta", dt_float, "abs({0:} - {0:}2)".format(val_name))
    calc.add("b_diff", dt_bool, "city <> city2")
```

```
flt = Filter(h)
flt(calc)
flt.add("b_diff", "True")

sort = Sort(h)
sort(flt)
sort.set_sort(["city", "delta"])

gr = Group(h)
gr(sort)
gr.add_group_names("city")
gr.add_value_names("city2", "First")
gr.remove_suffix()

param = FieldParam(h)
param(gr)
param.set_map([{
    original_name_key: "city2", #старое имя
    name_key: res_col,          #новое имя
    disp_name_key: res_disp     #подпись
    }])

res_join = Join(h)    # По умолчанию левое слияние
res_join(box[1], param)
res_join.set_links(["city"])

box(box[0], res_join)
```

Финальный вариант подмодели – результат



Внутри подмодели



Основной сценарий

Порт с результатом

Финальный вариант подмодели – итоговая таблица

#	city	Ближ. по широте	Ближ. по долготе	Ближ. по высоте
1	Астрахань	Ростов-на-Дону	Ульяновск	Калининград
2	Барнаул	Брянск	Новосибирск	Ульяновск
3	Брянск	Пенза	Москва	Новокузнецк
4	Владивосток	Махачкала	Хабаровск	Краснодар
5	Волгоград	Хабаровск	Нижний Новгород	Казань
6	Воронеж	Оренбург	Краснодар	Липецк
7	Екатеринбург	Ижевск	Челябинск	Челябинск
8	Иваново	Екатеринбург	Ярославль	Кемерово
9	Ижевск	Екатеринбург	Набережные Челны	Новосибирск
10	Иркутск	Липецк	Красноярск	Екатеринбург
11	Казань	Москва	Тольятти	Волгоград
12	Калининград	Уфа	Санкт-Петербург	Махачкала
13	Кемерово	Челябинск	Новокузнецк	Нижний Новгород
14	Киров	Пермь	Тольятти	Ульяновск
15	Краснодар	Астрахань	Воронеж	Владивосток
16	Красноярск	Чебоксары	Новокузнецк	Нижний Новгород
17	Липецк	Иркутск	Рязань	Уфа
18	Махачкала	Владивосток	Чебоксары	Калининград
19	Москва	Набережные Челны	Тула	Красноярск
20	Набережные Челны	Москва	Ижевск	Рязань
21	Нижний Новгород	Томск	Волгоград	Кемерово

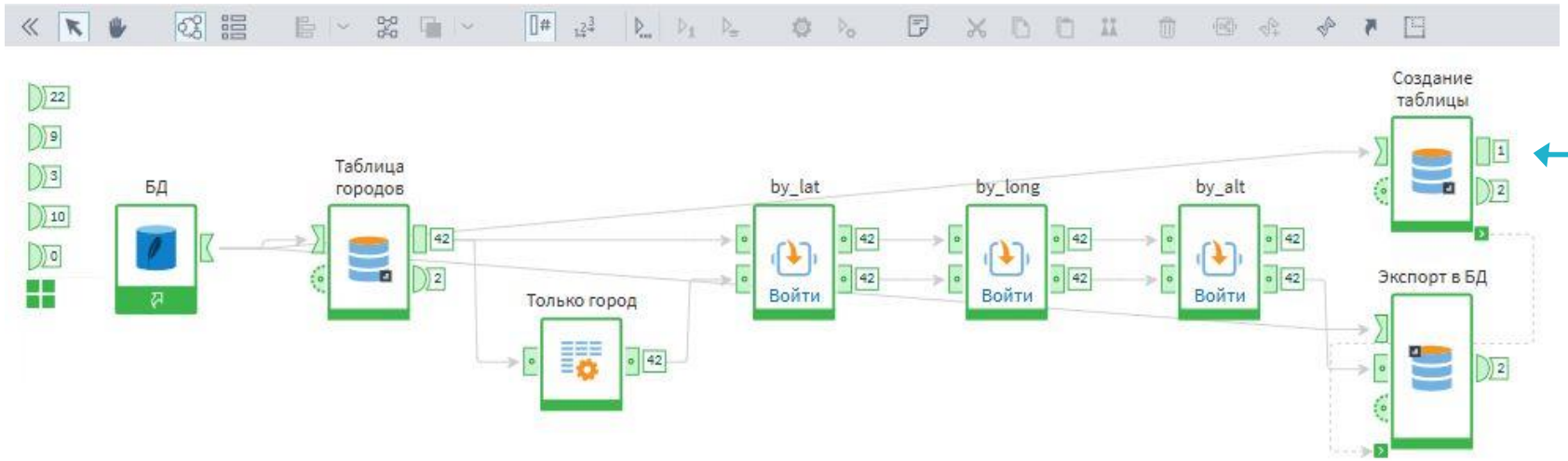
city	Ближ. по широте	Ближ. по долготе	Ближ. по высоте
Нижний Новгород	Томск	Волгоград	Кемерово
Новокузнецк	Тольятти	Кемерово	Брянск
Новосибирск	Омск	Барнаул	Ижевск
Омск	Новосибирск	Тюмень	Тольятти
Оренбург	Воронеж	Уфа	Чебоксары
Пенза	Самара	Волгоград	Пермь
Пермь	Ярославль	Уфа	Пенза
Ростов-на-Дону	Астрахань	Рязань	Саратов
Рязань	Калининград	Ростов-на-Дону	Набережные Челны
Самара	Пенза	Киров	Томск
Санкт-Петербург	Киров	Брянск	Махачкала
Саратов	Воронеж	Пенза	Ростов-на-Дону
Тольятти	Барнаул	Казань	Омск
Томск	Нижний Новгород	Кемерово	Самара
Тула	Ульяновск	Москва	Уфа
Тюмень	Иваново	Челябинск	Хабаровск
Ульяновск	Тула	Астрахань	Киров
Уфа	Калининград	Пермь	Липецк
Хабаровск	Волгоград	Владивосток	Тюмень
Чебоксары	Красноярск	Махачкала	Оренбург
Челябинск	Новосибирск	Екатеринбург	Новокузнецк
Ярославль	Пермь	Ростов-на-Дону	Набережные Челны

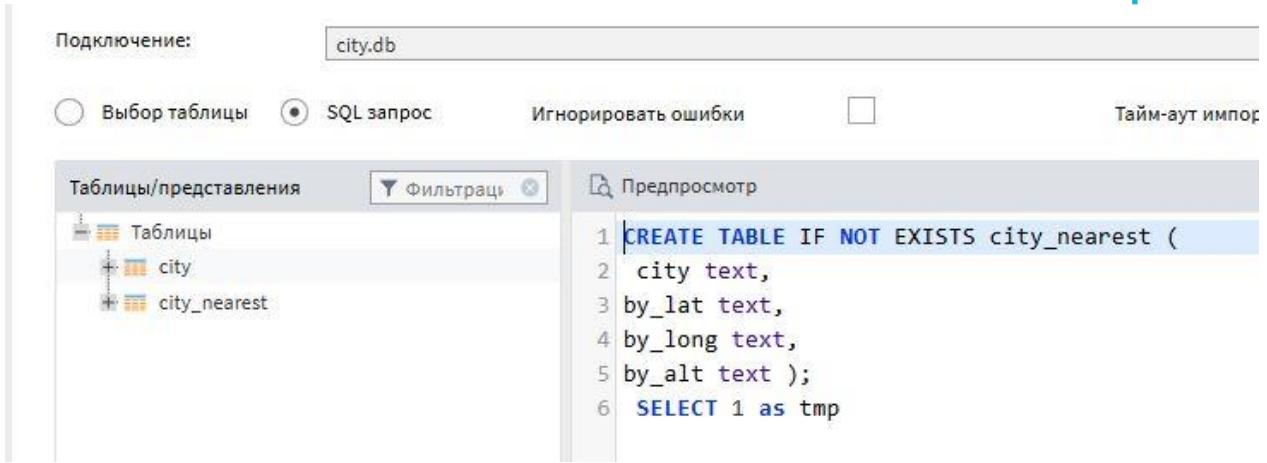
Код сохранения результатов в БД

```
def build(self, sc):
    box=sc
    ref_db = RefConnection(box, ref_uuid_str=sqlite_str)
    tab_city = ImportDBTable(box, table_name="city",
                             name="Таблица городов")
    tab_city(ref_db)
    tab_city.set_columns(
        ["scity", "flatitude", "flongtitude", "faltitude"])

    city_only = FieldParam(
        box, name="Только город", margin_y=80)
    city_only(tab_city)
    city_only.remove_columns(
        ["latitude", "longtitude", "altitude"])

    self.pred = (tab_city, city_only)

    self.calc_nearest(box, "latitude", "by_lat", "Ближ. по широте")
    self.calc_nearest(box, "longtitude", "by_long", "Ближ. по долготе")
    self.calc_nearest(box, "altitude", "by_alt", "Ближ. по высоте")

    export_db = ExportCreateDBTable(box, ref_db, self.pred[1],
                table="city_nearest", export_type="delete")
```

Сохранение результатов в БД (с созданием таблицы при необходимости)

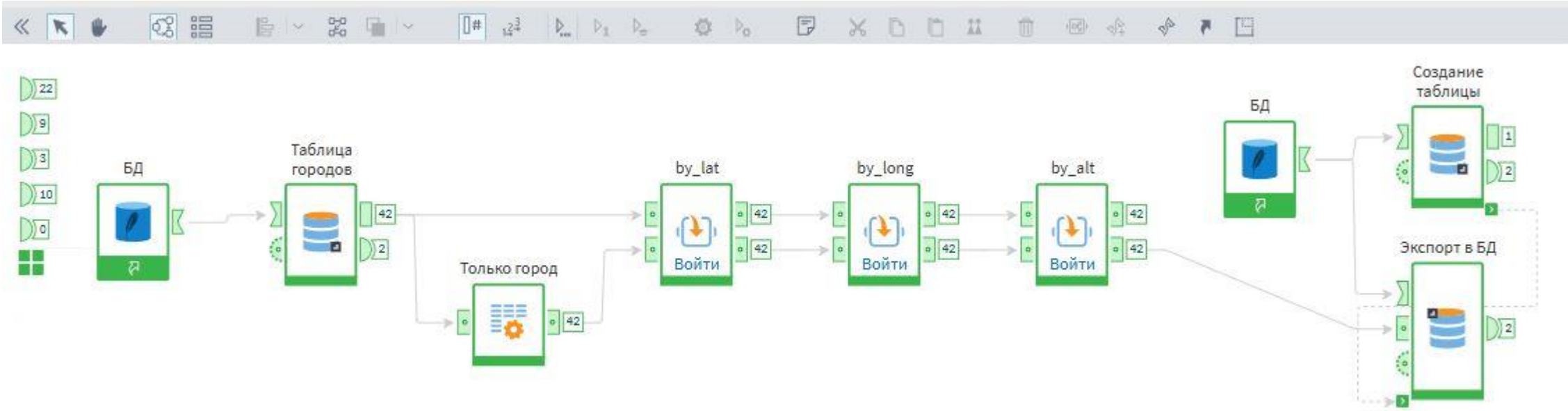
Сохранение итогов в БД – результат





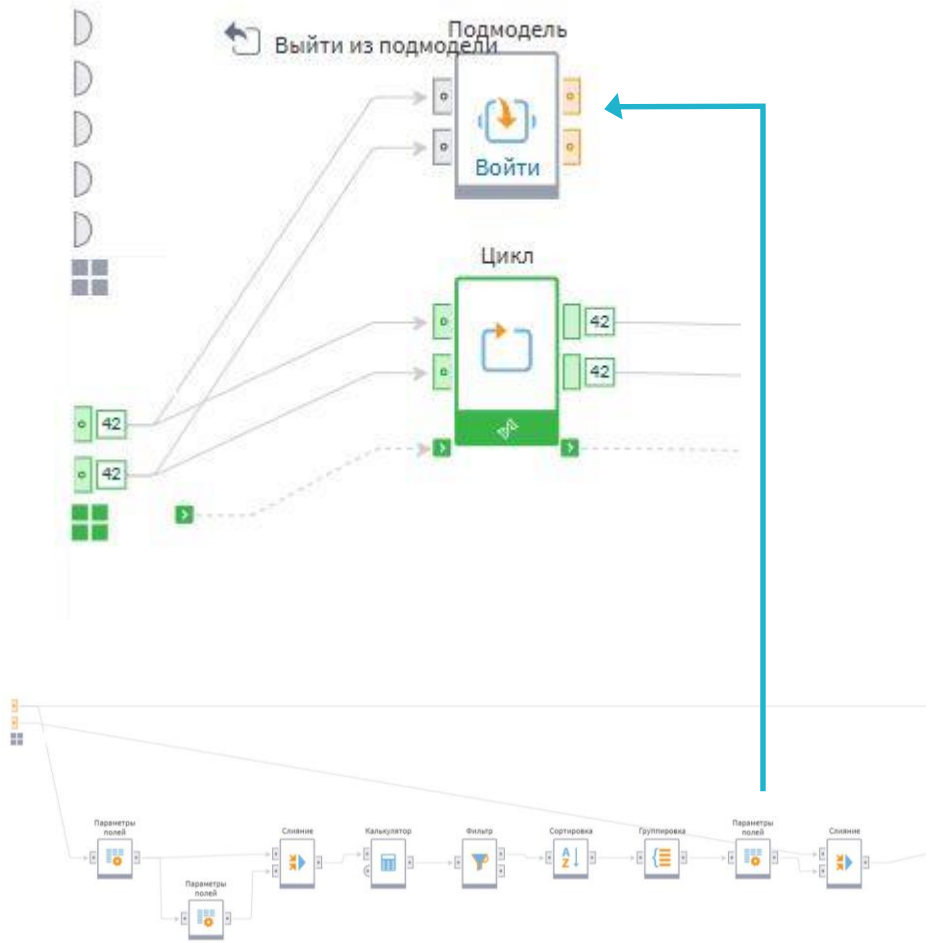
Код – заключительные штрихи

```
self.calc_nearest(box, "latitude", "by_lat", "Ближ. по широте")
self.calc_nearest(box, "longtitude", "by_long", "Ближ. по долготе")
self.calc_nearest(box, "altitude", "by_alt", "Ближ. по высоте")

ref_db2 = RefConnection(box, ref_uuid_str=sqlite_str, margin_y=-50)

export_db = ExportCreateDBTable(box, ref_db2, self.pred[1],
            table="city_nearest", export_type="delete")
```



Код – лайфхак (экономия памяти)

```
param = FieldParam(h)
param(gr)
param.set_map([{
    original_name_key: "city2", #старое имя
    name_key: res_col,          #новое имя
    disp_name_key: res_disp     #подпись
    }])

res_join = Join(h)    # По умолчанию левое слияние
res_join(box[1], param)
res_join.set_links(["city"])

box(box[0], res_join)

sm.loop()
```



Код создания подмодели. Метод calc_nearest. Добавляем в самом конце после завершения создания подмодели

Предложенный подход эффективно генерирует сценарии Loginom из кода на Python, что позволяет:

1. Повысить гибкость за счет возможностей Python
2. Облегчить решение сложных задач, требующих кодирования
3. Обеспечить контроль версий за счет версионирования кода генерации сценариев
4. Реализовать интеграцию с git, как с это делают программисты
5. Автоматически генерировать сценарии при помощи ИИ