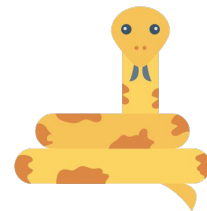


А МОЖНО ЕЩЕ БЫСТРЕЕ?

РАЗРАБОТКА ПРОЕКТОВ
НА PYTHON



КТО Я



Iuliia Volkova

xnuinside

Developer

<https://twitter.com/xnuinside> |

<https://www.linkedin.com/in/xnuinside>

Lead Developer @ 

<https://github.com/xnuinside>





О чем:

- что, как и чем можно ускорить?
- примеры с кейсами с контекстом
- о переиспользовании

PYTHON - ОЧЕНЬ РАЗНООБРАЗНЫЙ

Data Science

Backend

Applications

Data
Engineering

Scripts

etc.

ML

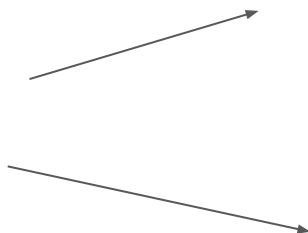


Рутина

Запуск
проектов с 0

ЧТО И КАК УСКОРЯЕМ

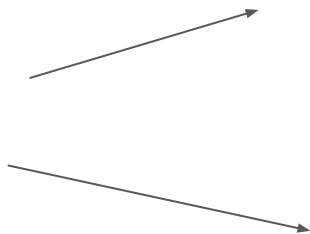
Рутину



Больше автоматизации

ЧТО И КАК УСКОРЯЕМ

Рутину

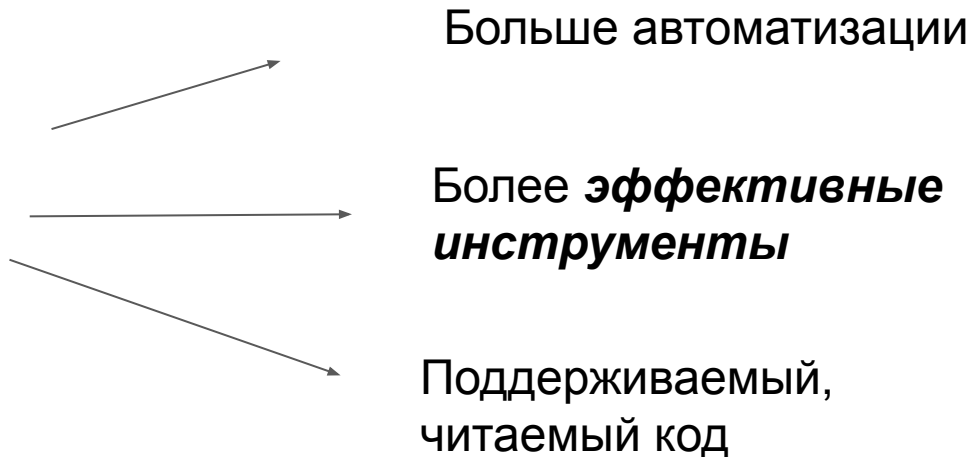


Больше автоматизации

Более **эффективные
инструменты**

ЧТО И КАК УСКОРЯЕМ

Рутину



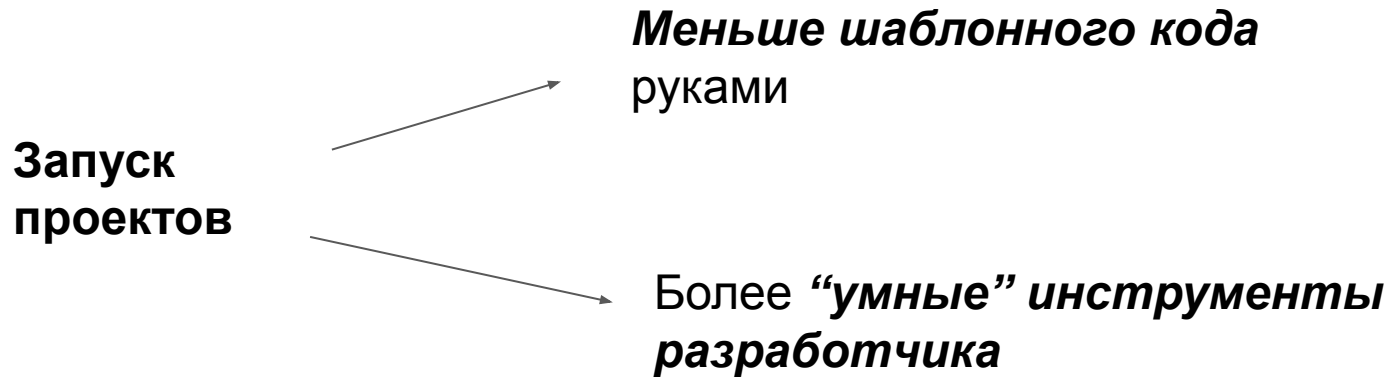
ЗАДАЧИ

Рутина

Запуск
проектов с 0

ЧТО И КАК УСКОРЯЕМ

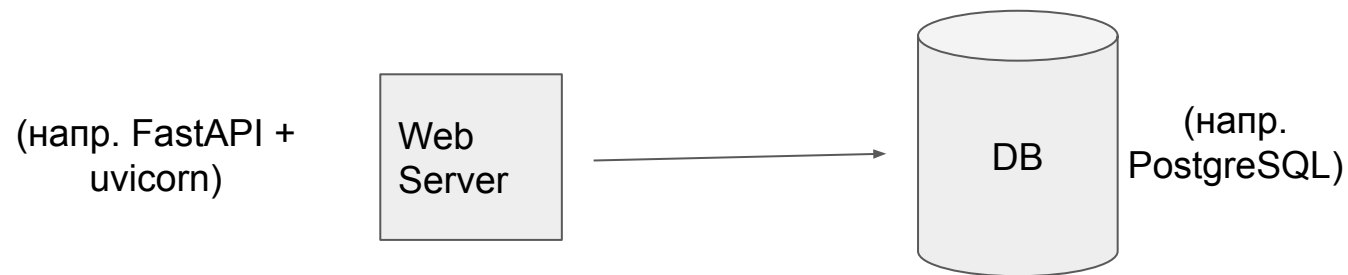






НА ЛИЧНОМ ПРИМЕРЕ

КЛАССИЧЕСКИЙ БЭК-ЕНД С SQL БАЗОЙ



КОГДА ТЫ НАЧИНАЕШЬ КАК PYTHON DEV

```
class Material(Base):  
  
    __tablename__ = 'materials'  
  
    id = sa.Column(sa.Integer(), autoincrement=True,  
primary_key=True)  
  
    title = sa.Column(sa.String(), nullable=False)  
  
    description = sa.Column(sa.Text())  
  
    link = sa.Column(sa.String(), nullable=False)  
  
    type = sa.Column(sa.Enum(MaterialType))
```

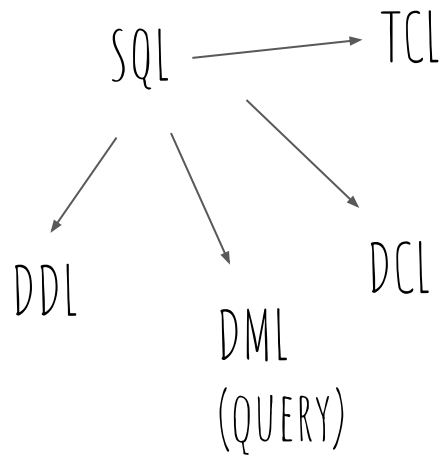
ORM МОДЕЛИ

DDL - ЯЗЫК ОПИСАНИЯ ТАБЛИЦ (НА САМОМ ДЕЛЕ НЕТ)

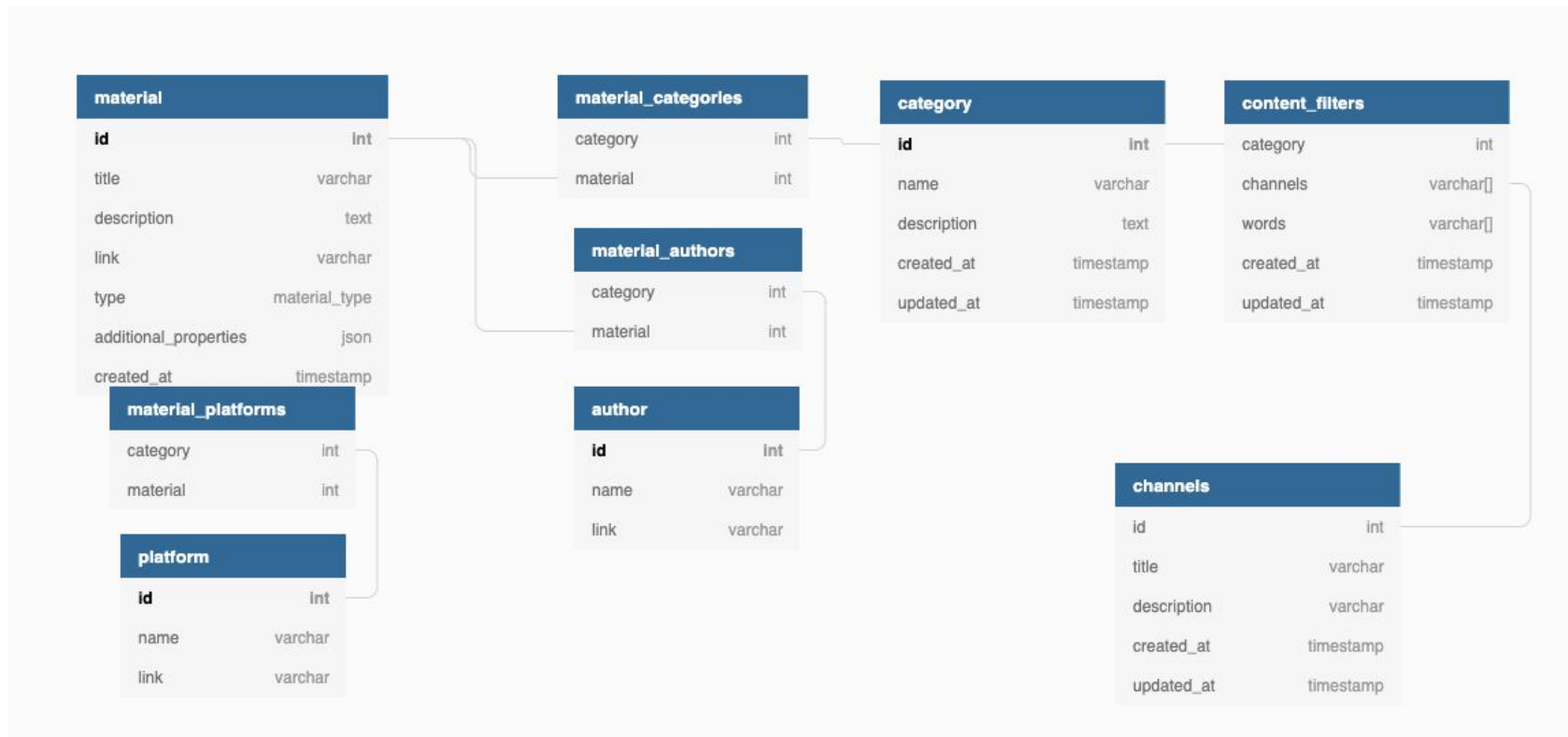
```
CREATE TABLE "material" (  
  "id" SERIAL PRIMARY KEY,  
  "title" varchar NOT NULL,  
  "description" text,  
  "link" varchar NOT NULL,  
  "type" material_type,  
  "additional_properties" json,  
  "created_at" timestamp DEFAULT (now()),  
  "updated_at" timestamp  
);
```

DDL - ЯЗЫК ОПИСАНИЯ ТАБЛИЦ (НА САМОМ ДЕЛЕ НЕТ)

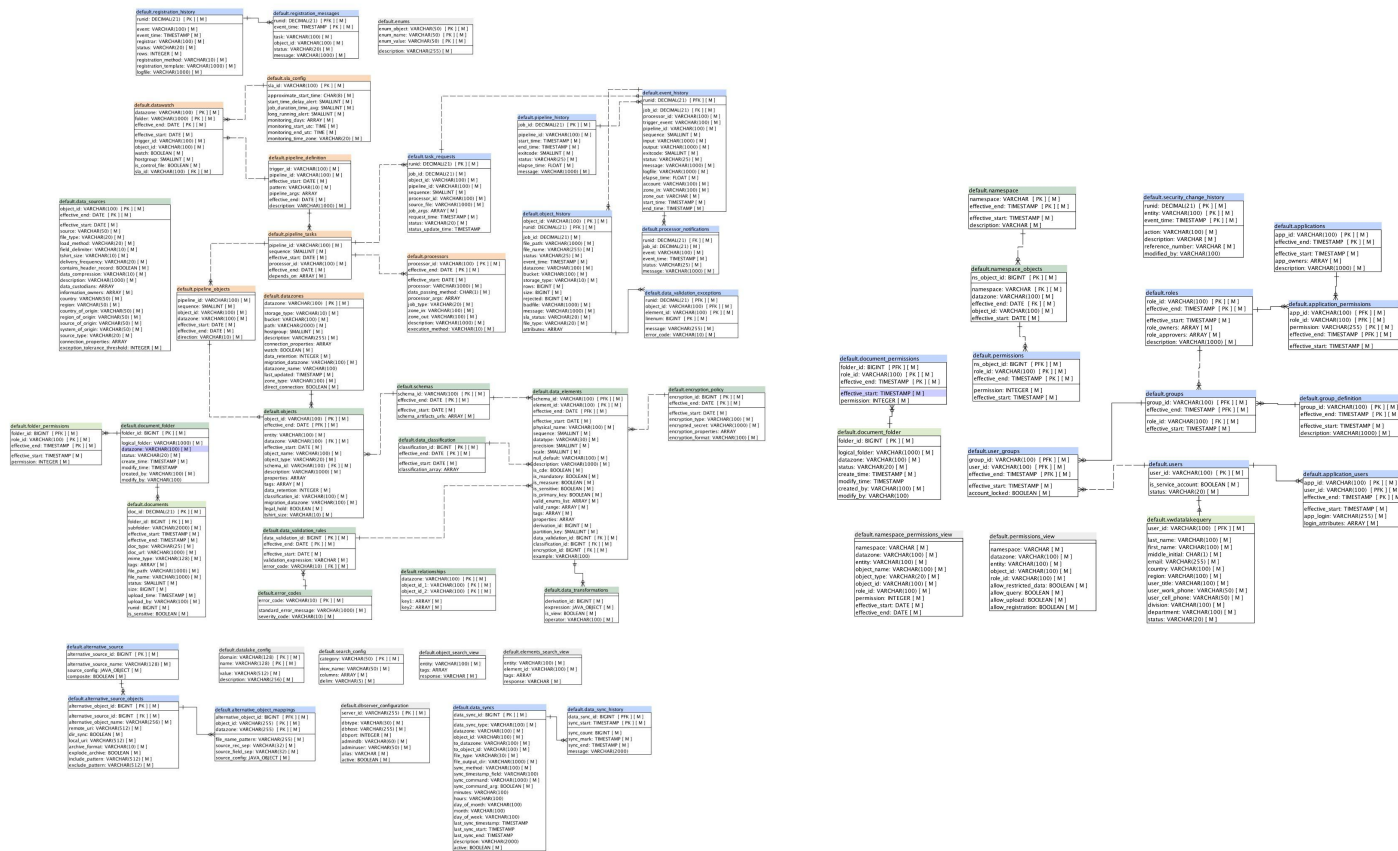
```
CREATE TABLE "material" (  
  "id" SERIAL PRIMARY KEY,  
  "title" varchar NOT NULL,  
  "description" text,  
  "link" varchar NOT NULL,  
  "type" material_type,  
  "additional_properties" json,  
  "created_at" timestamp DEFAULT (now()),  
  "updated_at" timestamp  
);
```



КАК ВЫГЛЯДИТ СХЕМА БАЗЫ СРЕДНЕГО ПРОЕКТА



А ПОТОМ ТЫ ПРИХОДИШЬ В ЭТО



НАДО:

Понять, что изменилось

```
CREATE TABLE "material" (  
  "id" SERIAL PRIMARY KEY,  
  "title" varchar NOT NULL,  
  "description" text,  
  "link" varchar NOT NULL,  
  "type" material_type,  
  "additional_properties" json,  
  "created_at" timestamp DEFAULT (now()),  
  "updated_at" timestamp  
);
```

```
CREATE TABLE "author" (  
  "id" SERIAL PRIMARY KEY,  
  "name" varchar,  
  "link" varchar  
);
```

НАДО:

Сделать новые
модели

ИЛИ

изменить
существующие

```
class MaterialType(Enum):
```

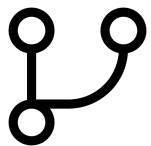
```
    article = 'article'  
    video = 'video'
```

```
class Material(Base):
```

```
    __tablename__ = 'material'
```

```
    id = sa.Column(sa.Integer(), autoincrement=True, primary_key=True)  
    title = sa.Column(sa.String(), nullable=False)  
    description = sa.Column(sa.Text())  
    link = sa.Column(sa.String(), nullable=False)  
    type = sa.Column(sa.Enum(MaterialType))  
    additional_properties = sa.Column(JSON(), server_default='{"key":  
"value"}')  
    created_at = sa.Column(sa.TIMESTAMP(), server_default=func.now())  
    updated_at = sa.Column(sa.TIMESTAMP())
```

CI PIPELINE

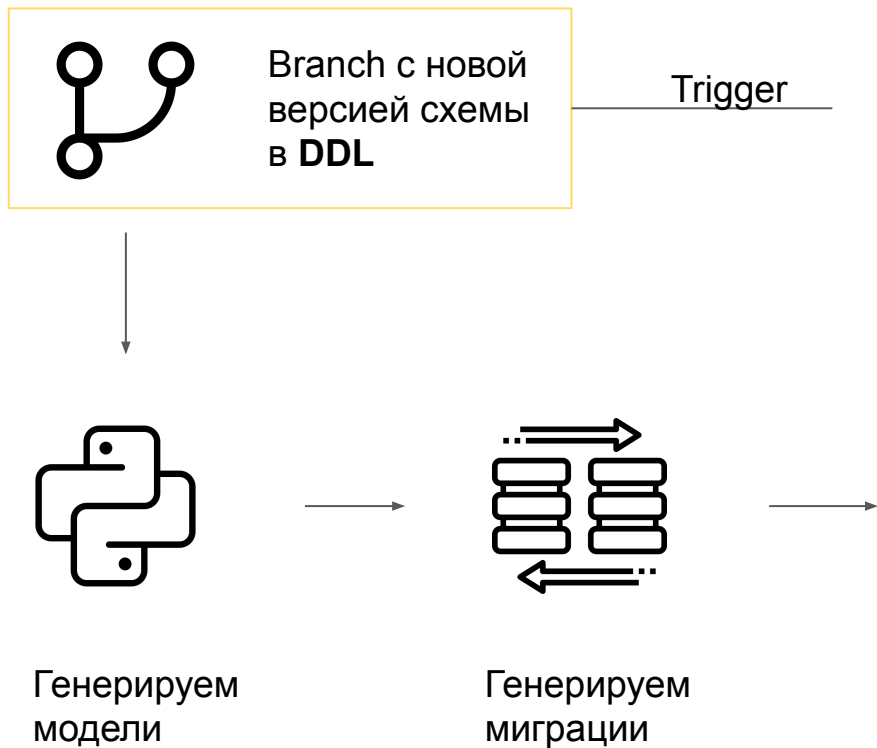


Branch с новой
версией схемы
в **DDL**

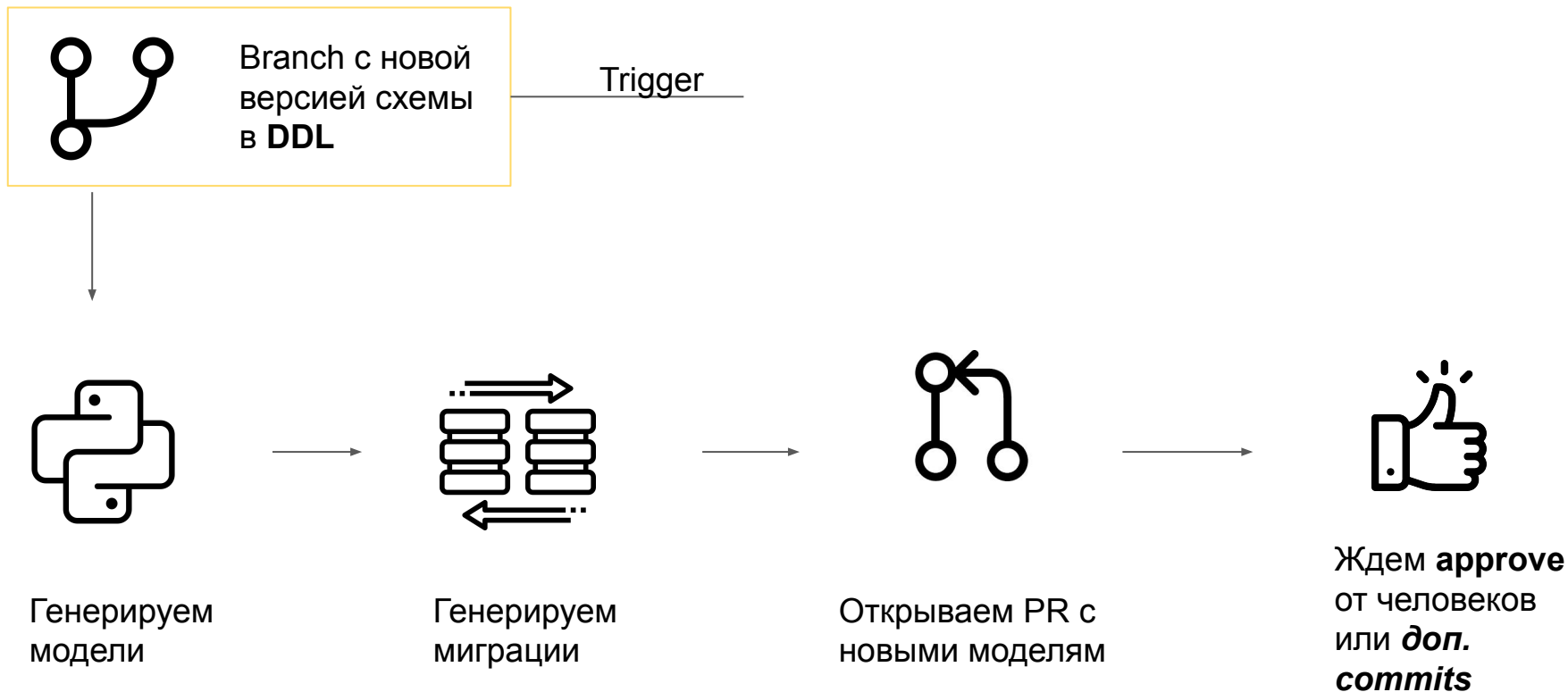
Trigger



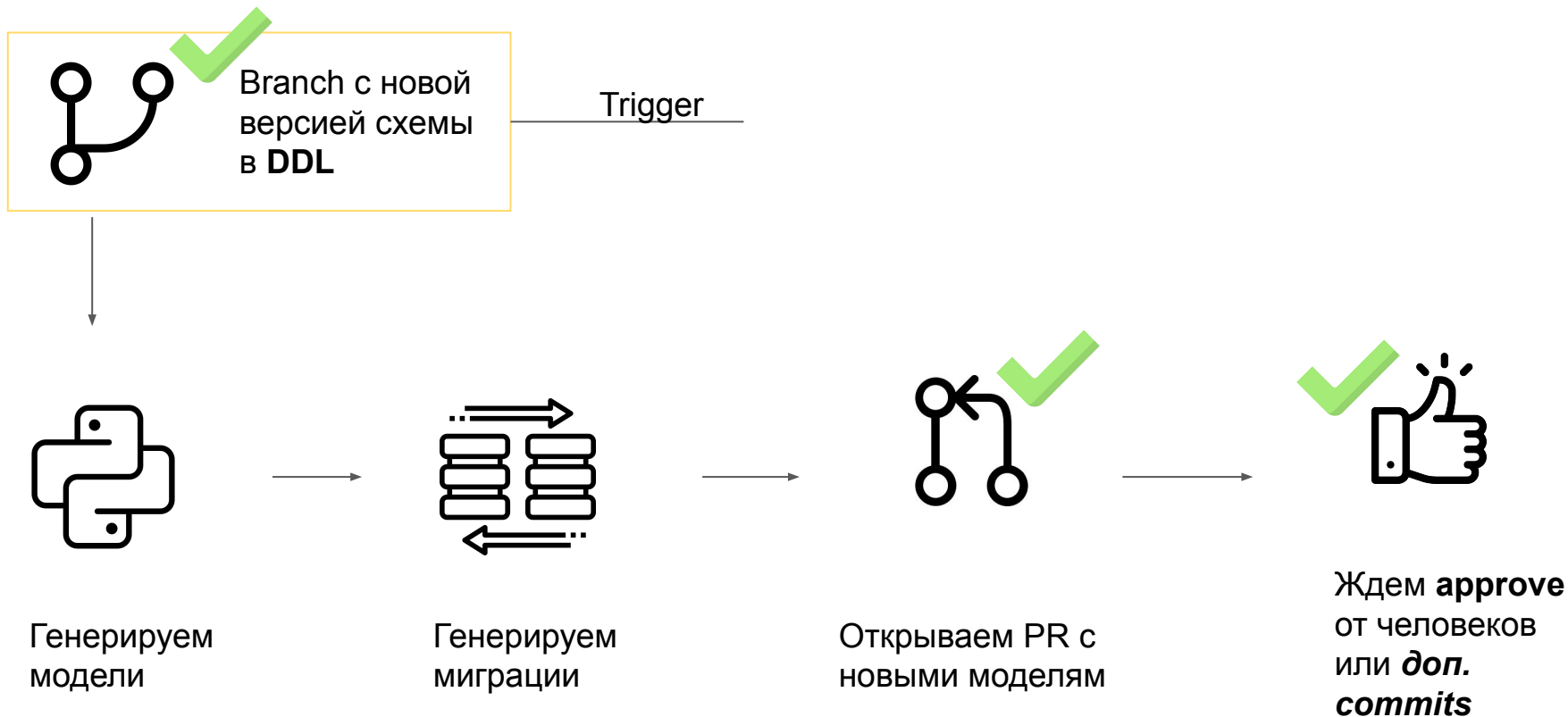
CI PIPELINE



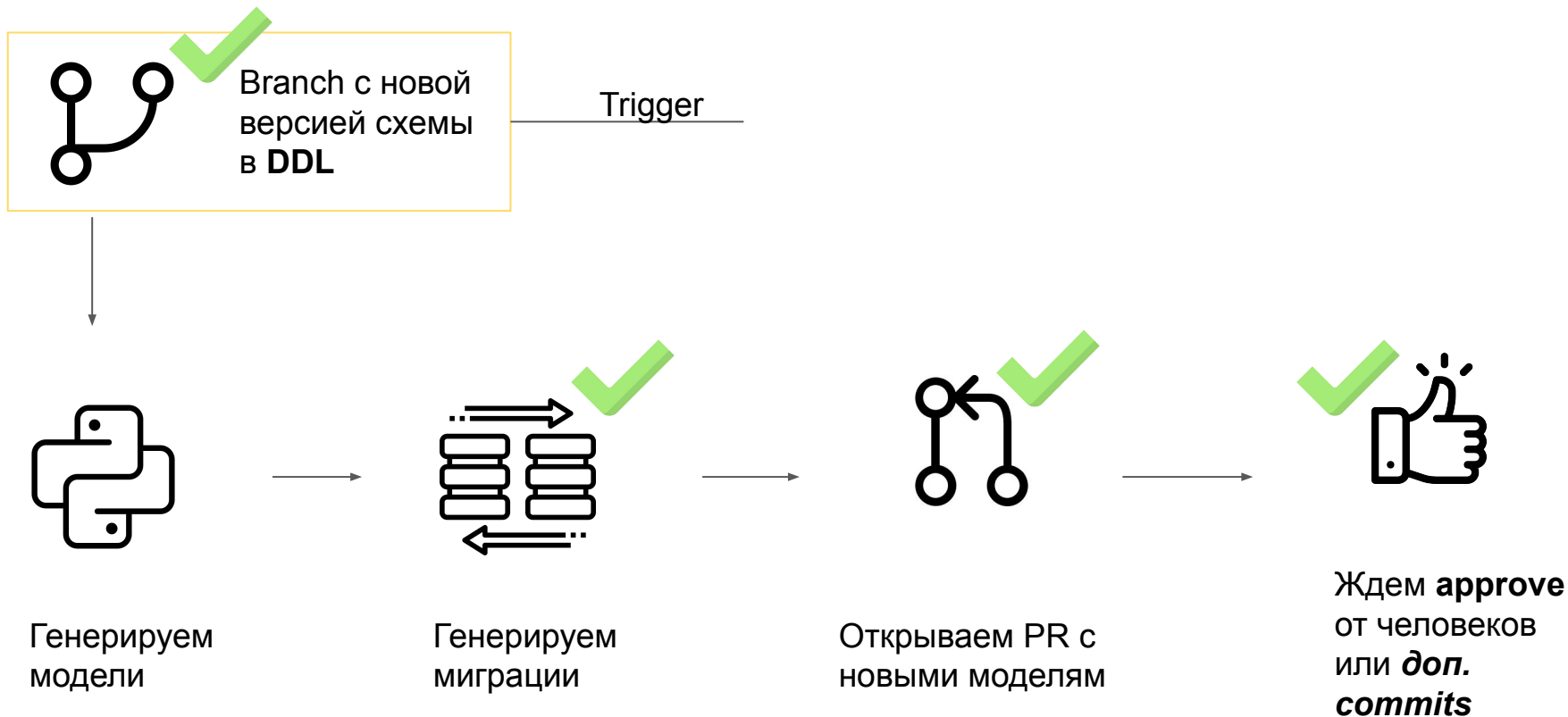
CI PIPELINE



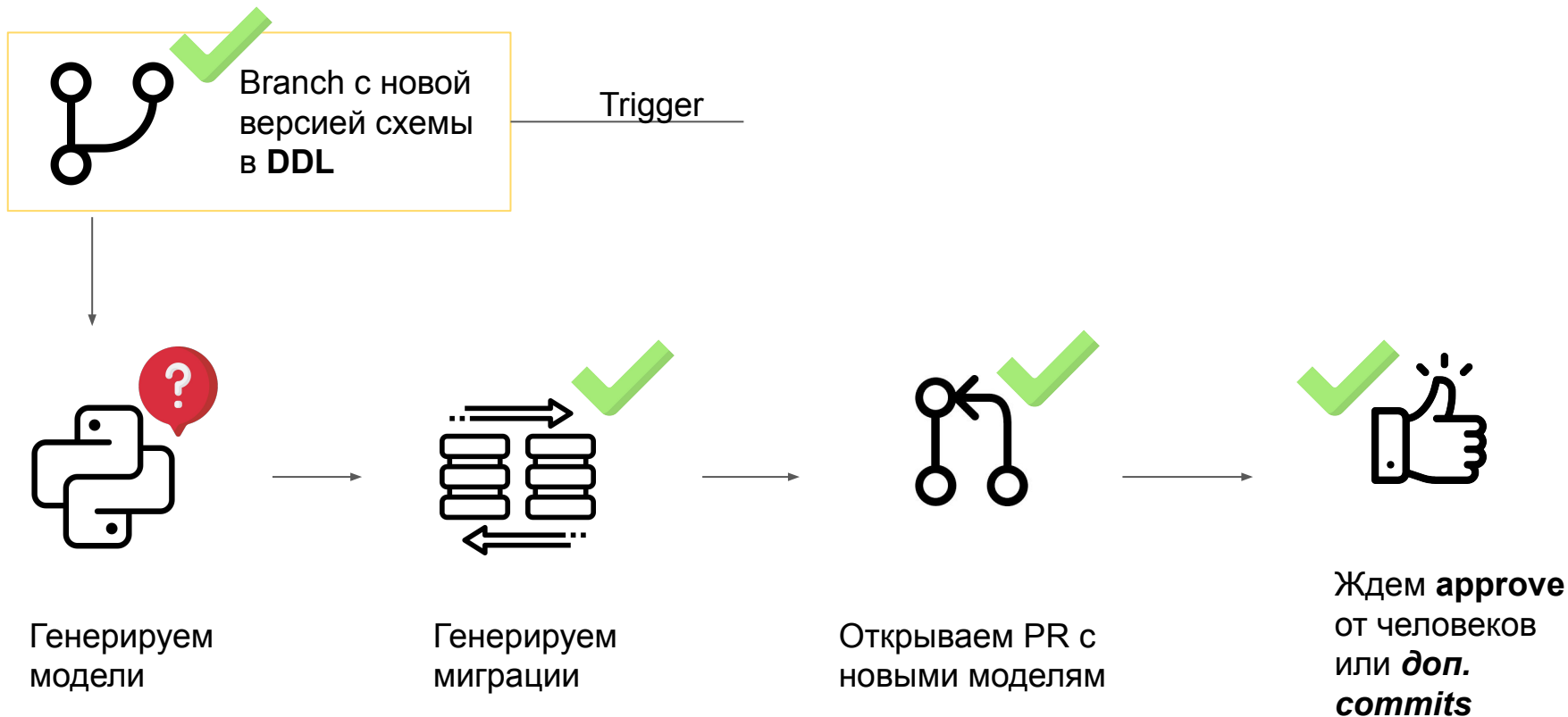
CI PIPELINE



CI PIPELINE



CI PIPELINE



ГЕНЕРАЦИЯ МОДЕЛЕЙ: ВАРИАНТЫ



Генерируем
модели

Если бы мы были на Django...

```
python manage.py inspectdb > models.py
```

ГЕНЕРАЦИЯ МОДЕЛЕЙ: ВАРИАНТЫ



Генерируем
модели

Если бы мы были на SQLAlchemy ...

```
sqlacodegen postgresql:///some_local_db
```

1) Но у нас GINOORM

2) И НЕ ХОЧЕТСЯ дополнительный шаг с базой

ЗАХОТЕЛОСЬ НАПИСАТЬ ЛИБУ
КОНЕЧНО ЖЕ В СВОЕ СВОБОДНОЕ ВРЕМЯ

ПРОСТАЯ ИДЕЯ

```
CREATE TABLE "material" (  
    "id" SERIAL PRIMARY KEY,  
    "title" varchar NOT NULL,  
    "description" text,  
    "link" varchar NOT NULL,  
    "type" material_type,  
    "additional_properties" json,  
    "created_at" timestamp DEFAULT (now()),  
    "updated_at" timestamp  
);
```

```
class Material(Base):
```

```
    __tablename__ = 'material'
```

```
    id = sa.Column(sa.Integer(),  
        autoincrement=True, primary_key=True)
```

```
    title = sa.Column(sa.String(), nullable=False)
```

```
    description = sa.Column(sa.Text())
```

```
    link = sa.Column(sa.String(), nullable=False)
```

```
    type = sa.Column(sa.Enum(MaterialType))
```

ПРОСТАЯ ИДЕЯ

```
CREATE TABLE "material" (  
    "id" SERIAL PRIMARY KEY,  
    "title" varchar NOT NULL,  
    "description" text,  
    "link" varchar NOT NULL,  
    "type" material_type,  
    "additional_properties" json,  
    "created_at" timestamp DEFAULT (now()),  
    "updated_at" timestamp  
);
```

```
class Material(Base):
```

```
    __tablename__ = 'material'
```

```
    id = sa.Column(sa.Integer(),  
        autoincrement=True, primary_key=True)
```

```
    title = sa.Column(sa.String(), nullable=False)
```

```
    description = sa.Column(sa.Text())
```

```
    link = sa.Column(sa.String(), nullable=False)
```

```
    type = sa.Column(sa.Enum(MaterialType))
```


ДА, ЭТО КОДОГЕНЕРАЦИЯ

- Трансляция инструкций в программный код

Инструкция →

Хочу такую кнопку, на которую пользователь нажимает и видит отчет за прошлый год

- Трансляция инструкций в программный код

Инструкция →

```
CREATE TABLE "material" (  
  "id" SERIAL PRIMARY KEY,  
  "title" varchar NOT NULL,  
  "description" text,  
  "link" varchar NOT NULL,  
  "type" material_type,  
  "additional_properties" json,  
  "created_at" timestamp DEFAULT (now()),  
  "updated_at" timestamp  
);
```

Трансляция инструкций в программный код

Сущности



Отношения
(Как связаны
сущность А с
сущность Б)

Из примера:

- Кнопка
- Пользователь
- Отчет

Трансляция инструкций в программный код

Сущности



Отношения

(Как
связаны
сущность А
с сущность
Б)



Действия

(что можно
делать с
сущностями?)

Из примера:

- *Кнопку можно нажать*
- *Отчет выгружается за какую-то дату*

Трансляция входящих инструкций в программный код

Сущности



Отношения
(Как
связаны
сущность А
с сущность
Б)



Действия
(что можно
делать с
сущностями?)

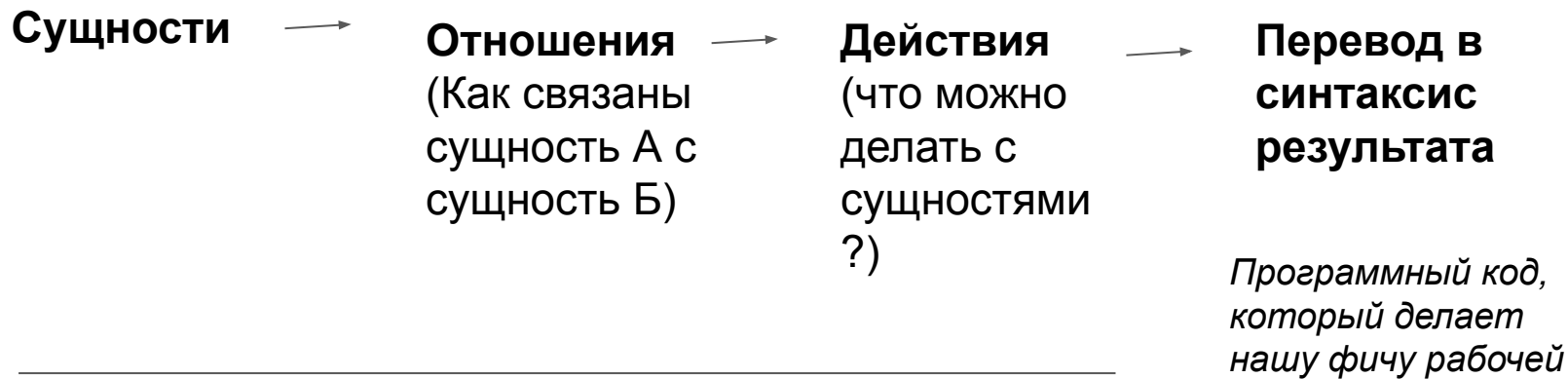
*На самом деле всё
сложнее:*

*Actors в действиях
Условия и
ограничения и тд*

Из примера:

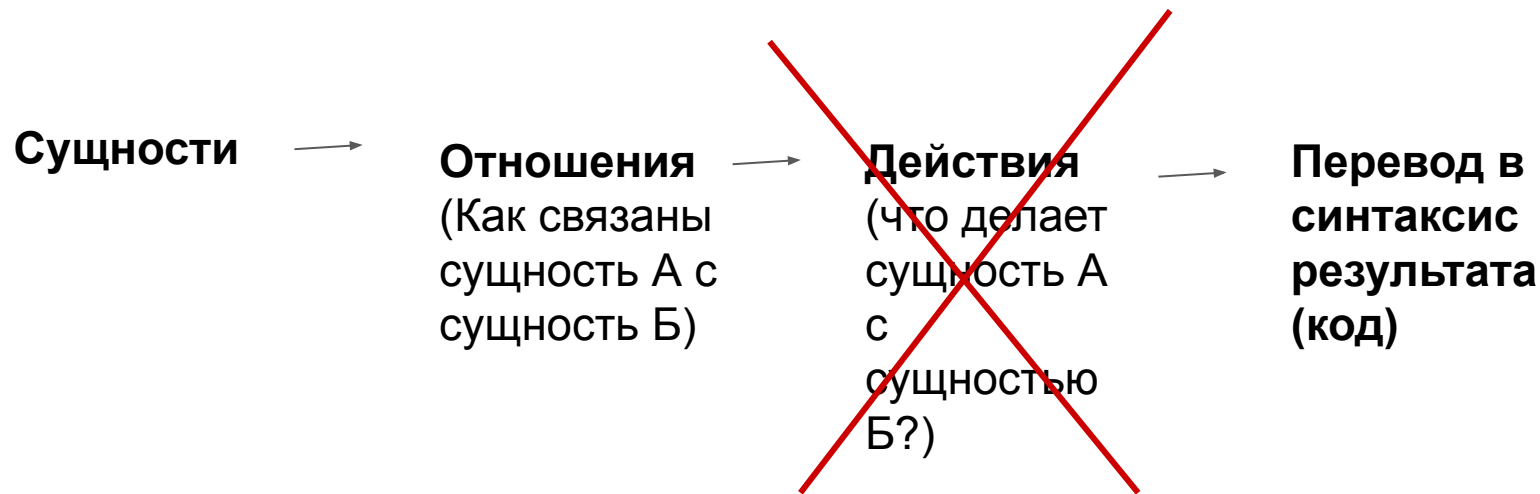
- *Кнопку можно нажать*
- *Отчет выгружается
за какую-то дату*

Трансляция входящих инструкций в программный код



Что нужно сделать?

Трансляция DDL инструкций



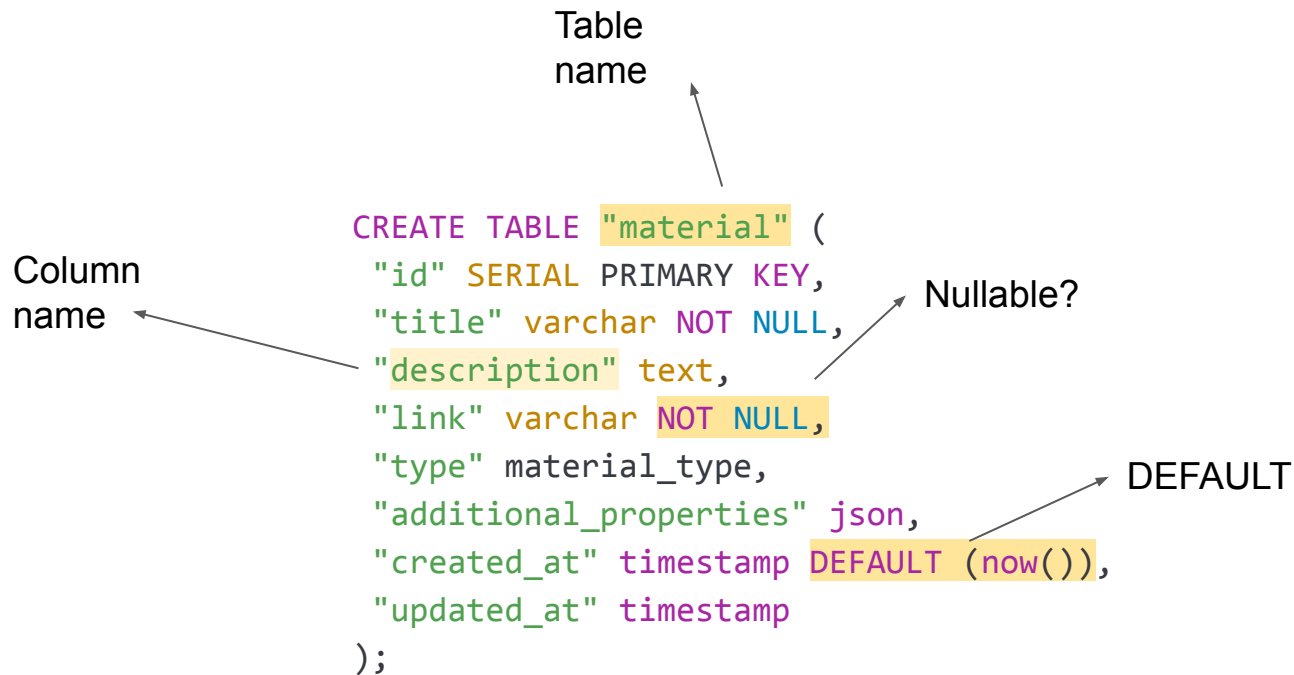
- Чем меньше количество исходных транслируемых инструкций и чем более они однозначны - тем проще написать алгоритм кодогенерации

- Конвертер DDL в модели: легко

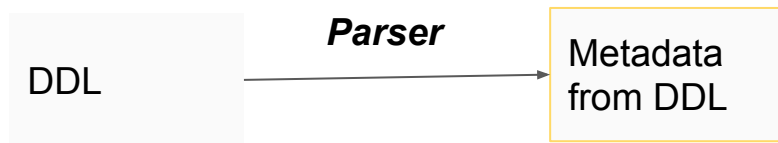
- Конвертер DDL в модели: легко
- Генерация unit тестов для Python кода: сложно

- Конвертер DDL в модели: легко
- Генерация unit тестов для Python кода: сложно
- Автоматическая интерпретация “эээ ну мне чтобы вот так же, но чуть по другому” - ...

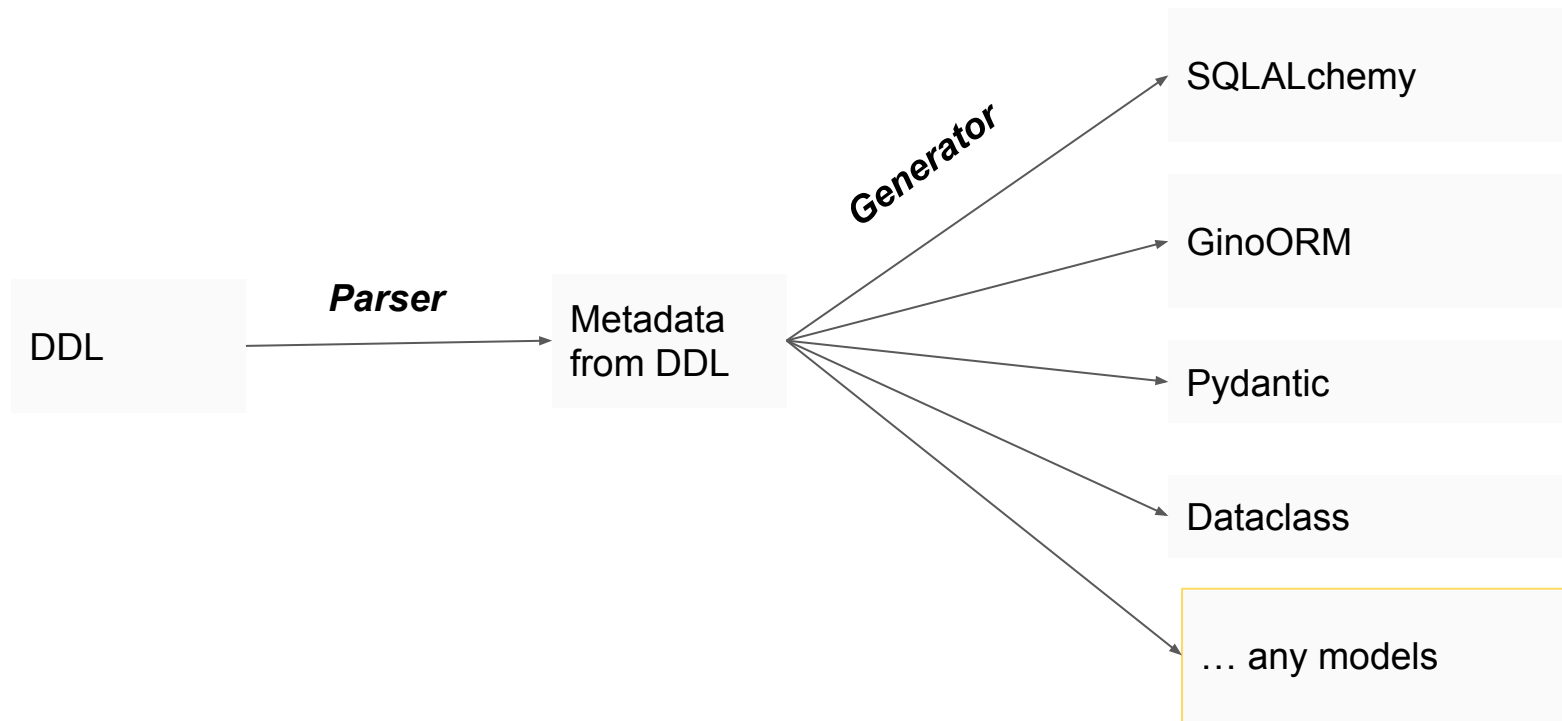
СУЩНОСТИ И ИХ СВОЙСТВА



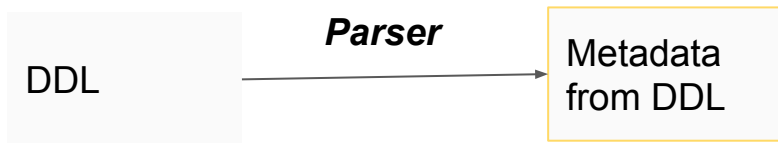
КОДОГЕНЕРАЦИЯ



КОДОГЕНЕРАЦИЯ



ПЕРЕИСПОЛЬЗОВАНИЕ



<https://github.com/andialbrecht/sqlparse>

andialbrecht/sqlparse on Feb 18

- **Several issues with DDL parsing - most annoying is table name identified as a function**
#610
`'create table A(...'` will identify A as a function
`'CREATE table A(...'` will not - ...

ПРИШЛОСЬ НАПИСАТЬ СВОЙ DDL PARSER

SIMPLE-DDL-PARSER (PLY - LEX&YACC)

```
'tables': [{ 'alter': {},
             'checks': [],
             'columns': [...
                        { 'check': None,
                          'default': 'now()',
                          'name': '"created_at"',
                          'nullable': True,
                          'references': None,
                          'size': None,
                          'type': 'timestamp',
                          'unique': False},...],
             'index': [],
             'partitioned_by': [],
             'primary_key': ['"id"'],
             'schema': None,
             'table_name': '"material"',
             'tablespace': None},
```

<https://github.com/xnuinside/simple-ddl-parser>

С поддержкой:

- HQL,
- T-SQL,
- Snowflake
- etc.

А ПОТОМ ЗАКРАЛАСЬ МЫСЛЬ

ЕДИНАЯ МЕТАДАТА

Tortoise ORM

```
class Event(Model):
    id = fields.IntField(pk=True)
    name = fields.TextField()
    datetime = fields.DatetimeField(null=True)

    class Meta:
        table = "event"
```

Pony ORM

```
class Event(db.Entity):
    id = PrimaryKey(int)
    name = Required(str)
    datetime = Optional(datetime)
```

ОДИНАКОВАЯ МЕТАДАТА
(СУЩНОСТИ И ТД)

PY-MODELS-PARSER (PARSIMONIOUS - PEG)

```
[{
    "attrs": [
        {
            "default": None,
            "name": "artist",
            "properties": {
                "foreign_key": "Musician",
                "on_delete": "models.CASCADE",
            },
            "type": "serial",
        }, ...
    ],
    "name": "Album",
    "parents": ["models.Model"],
    "properties": {},
},
]
```

[https://github.com/xnuinside/
py-models-parser](https://github.com/xnuinside/py-models-parser)

Поддержка 10+ разных
типов классов

(Pydal, Pydantic,
DjangoORM, SQLAlchemy,
etc)

ГЕНЕРАЦИЯ МОДЕЛЕЙ: O!MYMODELS

<https://github.com/xnuinside/omymodels>

```
$ omm /path/to/your.ddl --models_type pydantic
```

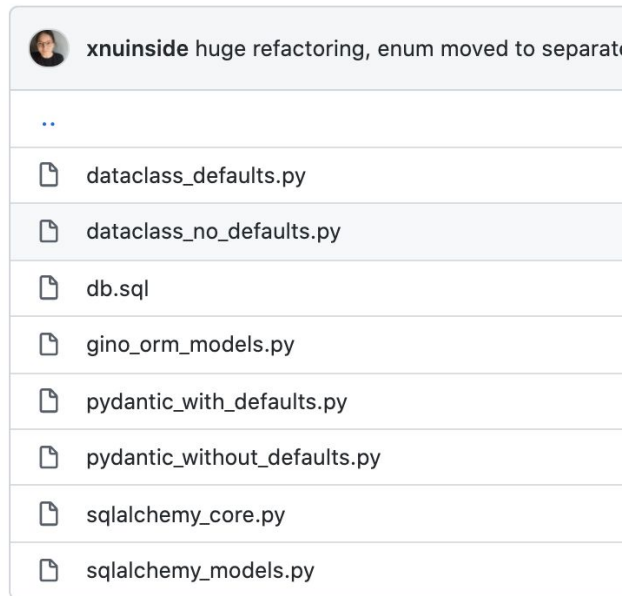
ГЕНЕРАЦИЯ И КОНВЕРТАЦИЯ МОДЕЛЕЙ: O!MYMODELS

Поддерживаемые модели:

- 1) SQLAlchemy,
- 2) GinoORM,
- 3) Pydantic
- 4) Dataclasses

В ближайших планах:

- 1) Pydal
- 2) TortoiseORM



ГЕНЕРАЦИЯ И КОНВЕРТАЦИЯ МОДЕЛЕЙ: O!MYMODELS

<https://github.com/xnuinside/omymodels>

DDL -> Model

Model -> Model

ЧТО ЕСЛИ НЕ DDL

OPEN API

```
openapi: 3.0.3
info:
  title: Generated API
  version: "1.0"
paths:
  /fruits:
    get:
      responses:
        200:
          description: OK
          content:
            application/json: {}
    post:
      requestBody:
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Fruit'
      responses:
        ...
```

OH JE SWAGGER

Schemes

HTTP

Authorize

pet Everything about your Pets

POST

/pet

Add a new pet to the store

PUT

/pet

Update an existing pet

GET

/pet/findByStatus

Finds Pets by status

GET

/pet/findByTags

Finds Pets by tags

GET

/pet/{petId}

Find pet by ID

POST

/pet/{petId}

Updates a pet in the store with form data

DELETE

/pet/{petId}

Deletes a pet

POST

/pet/{petId}/uploadImage

uploads an image

store Access to Petstore orders

OPEN API

```
openapi: 3.0.3
info:
  title: Generated API
  version: "1.0"
paths:
  /fruits:
    get:
      responses:
        200:
          description: OK
          content:
            application/json: {}
    post:
      requestBody:
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Fruit'
      responses:
```

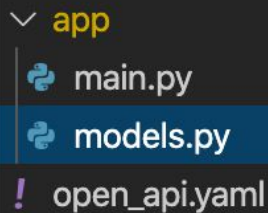
Schemas:

```
components:
  schemas:
    Fruit:
      properties:
        description:
          type: string
        name:
          type: string
```

FAST API BACKEND: FASTAPI-CODE-GENERATOR

<https://github.com/koxudaxi/fastapi-code-generator>

```
$ fastapi-codegen -i open_api.yaml -o app/
```



```
▼ app  
  main.py  
  models.py  
  ! open_api.yaml
```

BACKEND: FASTAPI-CODE-GENERATOR

```
# generated by fastapi-codegen:
#   filename: open_api.yaml
#   timestamp: 2021-07-24T22:17:02+00:00
```

```
from __future__ import annotations
```

```
from typing import Optional
```

```
from pydantic import BaseModel
```

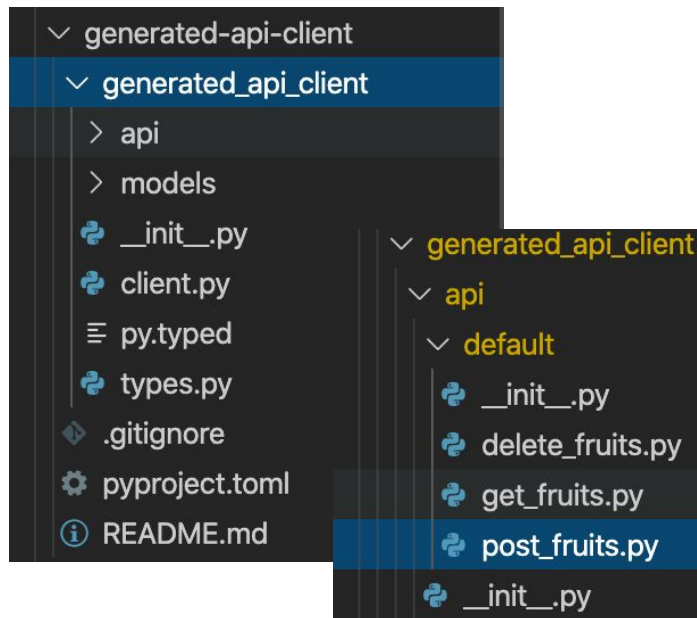
```
class Fruit(BaseModel):
    description: Optional[str] = None
    name: Optional[str] = None
```

```
5
6
7     @app.get('/fruits', response_model=Fruit)
8     def get_fruits() -> Fruit:
9         pass
10
11
12     @app.post('/fruits', response_model=Fruit)
13     def post_fruits(body: Fruit = None) -> Fruit:
14         pass
15
16
17     @app.delete('/fruits', response_model=Fruit)
18     def delete_fruits(body: Fruit = None) -> Fruit:
19         pass
20
```

PYTHON API CLIENT: OPENAPI-PYTHON-CLIENT

<https://github.com/openapi-generators/openapi-python-client>

```
$ openapi-python-client generate --path open_api.yaml
```



PYTHON API CLIENT: OPENAPI-PYTHON-CLIENT

```
async def asyncio_detailed(  
    *,  
    client: Client,  
    json_body: Any,  
) -> Response[Any]:  
    kwargs = _get_kwargs(  
        client=client,  
        json_body=json_body,  
    )  
  
    async with httpx.AsyncClient() as _client:  
        response = await _client.post(**kwargs)  
  
    return _build_response(response=response)
```

```
def sync_detailed(  
    *,  
    client: Client,  
    json_body: Any,  
) -> Response[Any]:  
    kwargs = _get_kwargs(  
        client=client,  
        json_body=json_body,  
    )  
  
    response = httpx.post(  
        **kwargs,  
    )  
  
    return _build_response(response=response)
```

PYTHON API CLIENT: OPENAPI-PYTHON-CLIENT

```
@attr.s(auto_attribs=True)
class AuthenticatedClient(Client):
    """ A Client which has been authenticated for use on secured endpoints """

    token: str

    def get_headers(self) -> Dict[str, str]:
        """ Get headers to be used in authenticated endpoints """
        return {"Authorization": f"Bearer {self.token}", **self.headers}
```

ЧТО ЕЩЁ

PYDANTIC MODELS

<https://github.com/koxudaxi/datamodel-code-generator>

```
[  
  --input-file-type  
    {auto, openapi, jsonschema, json, yaml, dict, csv}  
]
```

PYDANTIC MODELS

```
$ datamodel-codegen --input open_api.yaml  
--output pydantic_models.py
```

```
# generated by datamodel-codegen:  
#   filename: open_api.yaml  
#   timestamp: 2021-07-24T22:32:35+00:00  
  
from __future__ import annotations  
  
from typing import Optional  
  
from pydantic import BaseModel  
  
class Fruit(BaseModel):  
    description: Optional[str] = None  
    name: Optional[str] = None
```

<https://github.com/se2p/pynguin>



Attention

... We recommend running Pynguin in an isolated environment; ...

НЕ КОДОГЕНЕРАЦИЕЙ ОДНОЙ

- Пишем Pure Python

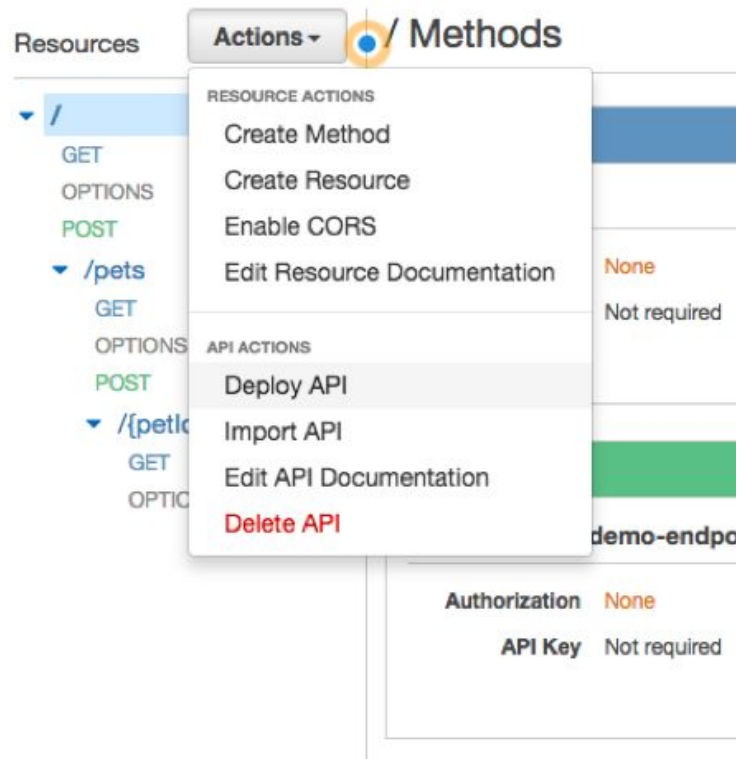
AWS Lambda

```
import os
import json

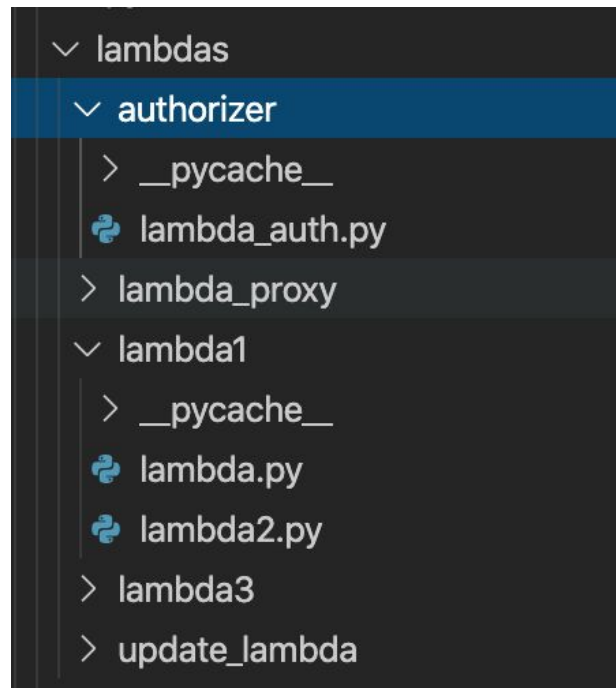
def lambda_handler(event, context):

    # Logic
    # some lambda result from logic
    lambda_result = {}
    return {
        "statusCode": 200,
        "headers": {
            "Content-Type": "application/json"
        },
        "body": json.dumps(lambda_result)
    }
```


- Пишем Pure Python
AWS Lambda
- весь роутинг через **API Gateway**



- Пишем Pure Python
AWS Lambda
- весь роутинг через **API Gateway**
- Source code - **набор python модулей** с лямбдами



- Хочется **локально** дебажить API
- и **запускать API тесты** локально (PostTrafficHook)

ХОЧЕТСЯ ПОТЕСТИРОВАТЬ LAMBDA ЛОКАЛЬНО



USE SAM TO BUILD TEMPLATES THAT DEFINE
YOUR SERVERLESS APPLICATIONS.

AWS SAM

FOR LAMBDA TESTING

Хочется потестировать LAMBDA локально



НЕМНОГО БОЛИ

```
terminal Local (4) Local Local (4) Local (5) T
CoreLayer is a local Layer in the template
CoreLayer is a local Layer in the template
Building image...CoreLayer is a local Layer in the template
Building image...CoreLayer is a local Layer in the template
CoreLayer is a local Layer in the template
Building image...CoreLayer is a local Layer in the template
CoreLayer is a local Layer in the template
CoreLayer is a local Layer in the template
CoreLayer is a local Layer in the template
Building image...CoreLayer is a local Layer in the template
Building image...Building image...Building image...Building image...Building image...Skip pulling im
age-python3.8:rapid-1.24.1.

Mounting C:\development\main\gcs-cms-task-lambda-activities\lambdas\swagger as /var/task:ro,delegated inside runtime
Skip pulling image and use local one: amazon/mws-sam-cli-emulation-image-python3.8:rapid-1.24.1.

Mounting C:\development\main\gcs-cms-task-lambda-activities\lambdas\cors_options as /var/task:ro,delegated inside run
CoreLayer is a local Layer in the template
Building image...CoreLayer is a local Layer in the template
Building image...
```

ЧТО ДЕЛАТЬ? AWS CLOUD FORMATION TEMPLATE

```
EntityCreateFunction:
  Type: AWS::Serverless::Function
  Properties:
    ...
    CodeUri: lambdas/enitites/
    Handler: create_v1.lambda_handler
    ...
  Events:
    MlApi:
      Type: Api
      Properties:
        Path: /v1/entities
        Method: post
        RestApiId: !Ref InternalApiGateway
```

ПОЧЕМУ БЫ И НЕТ?

```
EntityCreateFunction:
```

```
  Type: AWS::Serverless::Function
```

```
  Properties:
```

```
    ...
```

```
  CodeUri: lambdas/enitites/
```

```
  Handler: create_v1.lambda_handler
```

```
    ...
```

```
  Events:
```

```
    MLApi:
```

```
      Type: Api
```

```
      Properties:
```

```
        Path: /v1/entities
```

```
        Method: post
```

```
        RestApiId: !Ref InternalApiGateway
```

Package
Path

Module & Handler
Name

Path

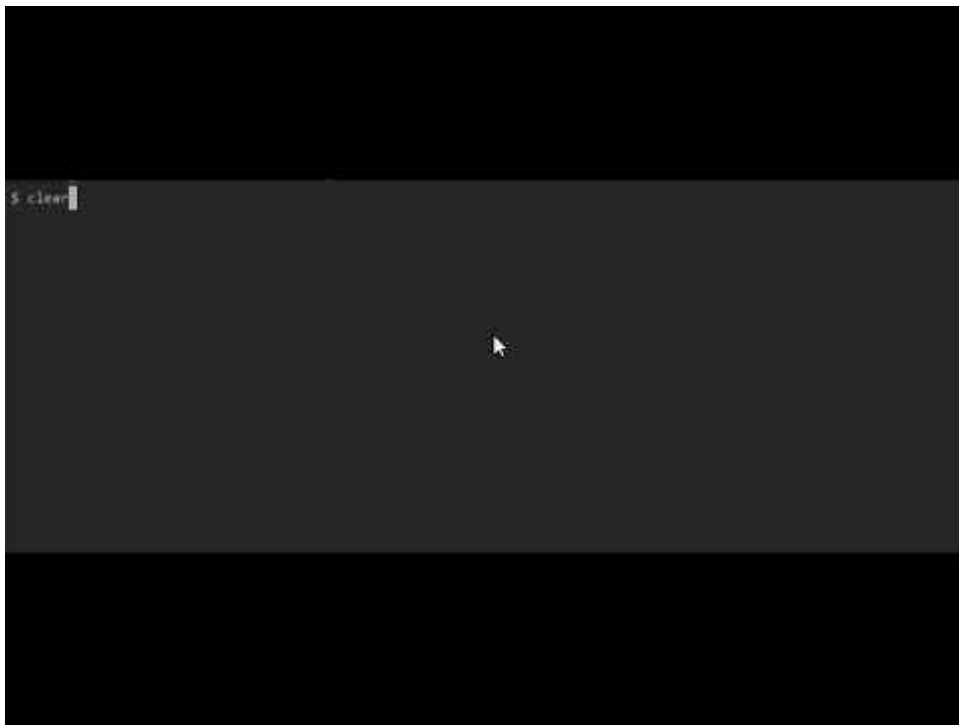
INPUT - ВСЕГДА ОДИН

```
def lambda_handler(event, context):  
    ...
```



```
{  
  "resource": "/my/path",  
  "path": "/my/path",  
  "httpMethod": "GET",  
  "headers": {  
    "header1": "value1",  
    "header2": "value2"  
  },  
  "multiValueHeaders": {  
    "header1": [  
      "value1"  
    ],  
    "header2": [  
      "value1",  
      "value2"  
    ]  
  },  
  "queryStringParameters": {  
    "parameter1": "value1",  
  }, ...}
```

LOF - AWS LAMBDA ON FASTAPI



LoF - LAMBDA S ON FASTAPI

<https://github.com/xnuinside/lof>

При 15+ lambdas:

LoF: около 1 сек запуск

SAM: 300+ сек, если запустится

```
$ lof --env=vars.json
```

ИТОГО

ПАРА СЛОВ ПРО OPEN SOURCE,
МЕНТОРИНГ И КОМЬЮНИТИ

ВСЕМ СПАСИБО!

НА ПРОЕКТЫ



НА СЛАЙДЫ

