Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут”
Кафедра АСОІУ

ЗВІТ
про виконання лабораторної роботи №1
з дисципліни
“ Аналіз даних в інформаційно-управляючих системах”
СТВОРЕННЯ СХОВИЩА ДАНИХ

Виконав Студент
3 курсу групи ІП-11
Панченко Сергій

Київ 2023

Мета роботи: ознайомитись з підходами до створення сховищ даних.

1 Відкриті джерела даних:

1) Реєстр адміністративно-територіального устрою(28-ex_xml_atu.xml): https://data.gov.ua/dataset/a2d6c060-e7e6-4471-ac67-42cfa1742a19
2) Відомості про транспортні засоби та їх власників (tz_opendata_z01012022_po01012023.csv): https://data.gov.ua/dataset/06779371-308f-42d7-895e-5a39833375f0/resource/7a58e8f7-9323-47d4-a21d-19486e014eb4
3) Перелік транспортних засобів комунальної власності Кременчуцької міської територіальної громади: https://data.gov.ua/dataset/a4e15058-8356-4a38-a78d-39ae957d2c7a/resource/f44a20c5-8446-422a-b932-5bc9eb998cef/download/perelik-obiektiv-komunalnoyi-vlasnosti-kremenchutskoyi-miskoyi-teritorialnoyi-gromadi.xls

Робота виконується на мові Python за допомогою ORM SQLAlchemy, що широко використовується провідними компаніями для безепечної та універсальної роботи з будь-яким діалектом SQL. Наразі я підключи PostgreSQL до даної ORM.

2 Моделювання Stage зони для ETL процесів:

!!!

Для кращого розуміння, як мій код працює, перейдіть на мій github:

https://github.com/SideShowBoBGOT/DataAnalysisFourthSemester

!!!

Код Stage Моделей:

stage_models.py

__init__.py

```python
from sqlalchemy.ext.declarative import declarative_base
Base = declarative_base()
from .administrative_unit import AdministrativeUnit
from .communal_property import CommunalProperty
from .transport_info import TransportInfo
```

administrative_unit.py

```python
from sqlalchemy import Column, String, Integer
from ..mixins import ReprAttributesString
from . import Base


class AdministrativeUnit(Base, ReprAttributesString):
    __tablename__ = 'administrative_unit'

    def __init__(self, obl: str, region: str, city: str, city_region: str, street: str) -> None:
        self.obl = obl
        self.region = region
        self.city = city
        self.city_region = city_region
        self.street = street

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

```
    obl = Column(String)
    region = Column(String)
    city = Column(String)
    city_region = Column(String)
    street = Column(String)
```

communal_property.py

```
from sqlalchemy import Column, String, Integer, Float
from ..mixins import ReprAttributesString
from . import Base


class CommunalProperty(Base, ReprAttributesString):
    __tablename__ = 'communal_property'

    def __init__(self, balance_keeper: str, address: str, name: str,
                 area: float, land_area: float, components_name: str,
                 components_area: float, letters: str) -> None:
        self.balance_keeper = balance_keeper
        self.address = address
        self.name = name
        self.area = area
        self.land_area = land_area
        self.components_name = components_name
        self.components_area = components_area
        self.letters = letters

    id = Column(Integer, primary_key=True, autoincrement=True)
    balance_keeper = Column(String)
    address = Column(String)
    name = Column(String)
    area = Column(Float)
    land_area = Column(Float)
    components_name = Column(String)
    components_area = Column(Float)
    letters = Column(String)
```

transport_info.py

```
from sqlalchemy import Column, String, Integer, BIGINT, Date, Float
from ..mixins import ReprAttributesString
from . import Base

class TransportInfo(Base, ReprAttributesString):
    __tablename__ = 'transport_info'

    def __init__(self, reg_addr_koatuu: int, oper_code: int,
                 oper_name: str, d_reg: str, dep_code: int, dep: str,
                 brand: str, model: str, vin: str, make_year: int,
                 color: str, kind: str, body: str, purpose: str, fuel: str,
                 capacity: float, own_weight: float, total_weight: float, n_reg_new: str) -> None:
        self.reg_addr_koatuu = reg_addr_koatuu
        self.oper_code = oper_code
        self.oper_name = oper_name
        self.d_reg = d_reg
        self.dep_code = dep_code
        self.dep = dep
        self.brand = brand
```

```
        self.model = model
        self.vin = vin
        self.make_year = make_year
        self.color = color
        self.kind = kind
        self.body = body
        self.purpose = purpose
        self.fuel = fuel
        self.capacity = capacity
        self.own_weight = own_weight
        self.total_weight = total_weight
        self.n_reg_new = n_reg_new

    id = Column(Integer, primary_key=True, autoincrement=True)
    reg_addr_koatuu = Column(BIGINT)
    oper_code = Column(Integer)
    oper_name = Column(String)
    d_reg = Column(Date)
    dep_code = Column(Integer)
    dep = Column(String)
    brand = Column(String)
    model = Column(String)
    vin = Column(String)
    make_year = Column(Integer)
    color = Column(String)
    kind = Column(String)
    body = Column(String)
    purpose = Column(String)
    fuel = Column(String)
    capacity = Column(Float)
    own_weight = Column(Float)
    total_weight = Column(Float)
    n_reg_new = Column(String)
```

Схема Stage зони:

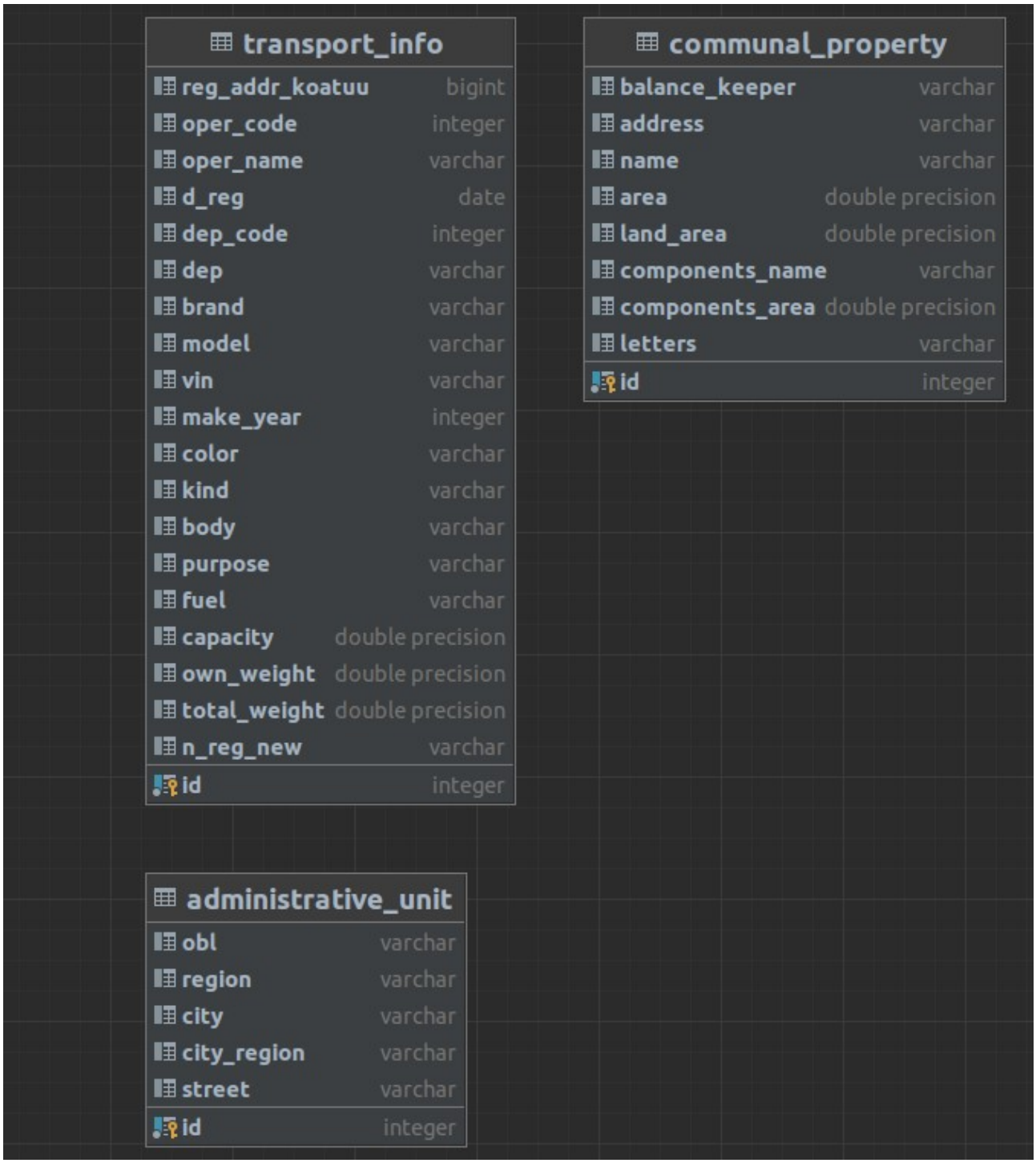
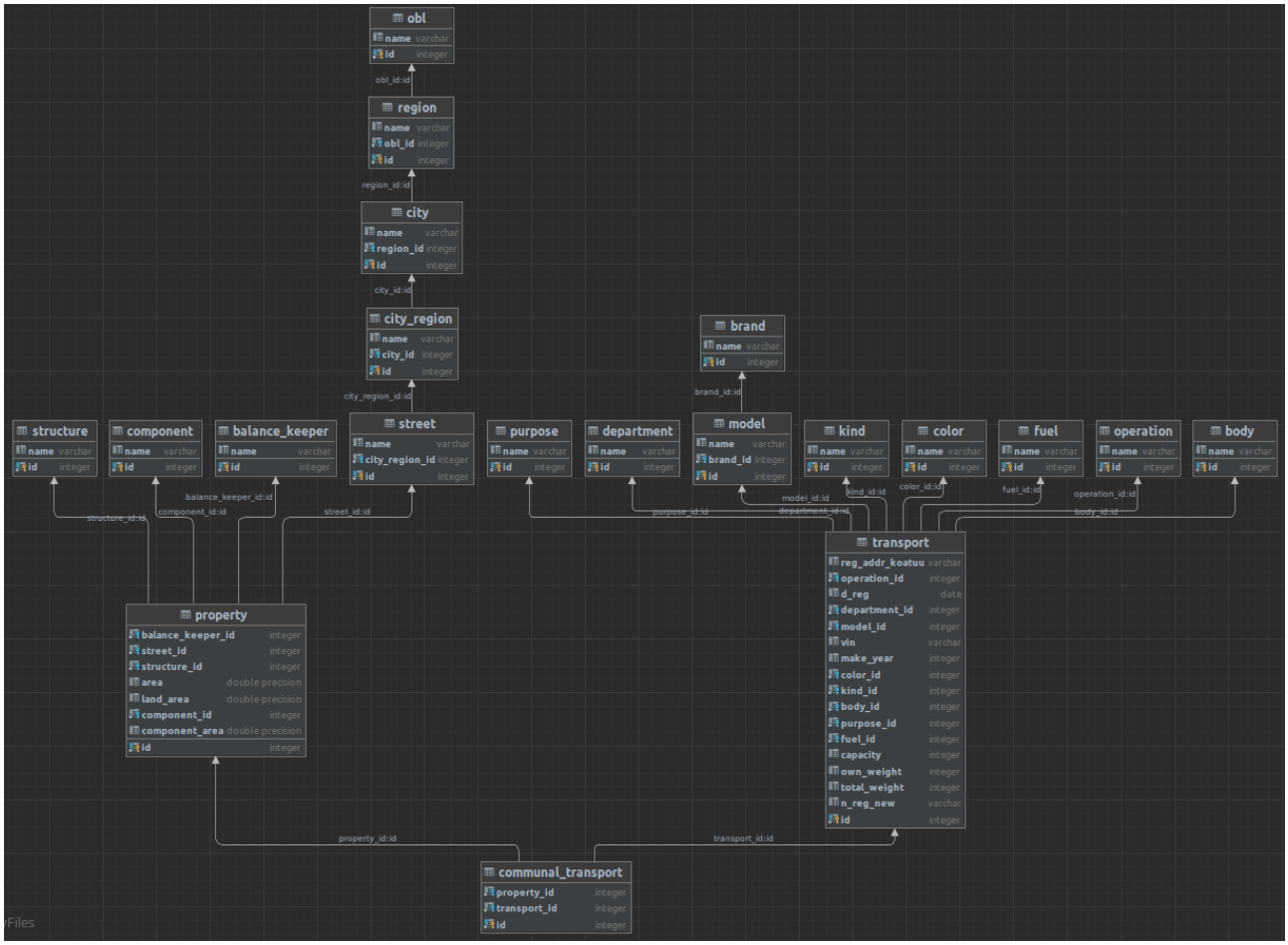


Схема основного сховища за типом “сніжинка” та код моделей:



main_models.py

administrative_unit.py

__init__.py

```
from .. import Base, ReprAttributesString, ID_STR, CASCADE
from .obl import Obl
from .region import Region
from .city import City
from .city_region import CityRegion
from .street import Street
```

city.py

```
from sqlalchemy import Column, String, Integer, ForeignKey

from . import Base, ReprAttributesString, ID_STR, CASCADE
from .region import Region


class City(Base, ReprAttributesString):
    __tablename__ = 'city'

    def __init__(self, name: str, region_id: int) -> None:
        self.name = name
        self.region_id = region_id
```

```
    id = Column(Integer, primary_key=True, autoincrement=True)
    name = Column(String)
    region_id = Column(Integer, ForeignKey(
        Region.__tablename__ + ID_STR, ondelete=CASCADE), nullable=False)
```

city_region.py

```
from sqlalchemy import Column, String, Integer, ForeignKey

from . import Base, ReprAttributesString, ID_STR, CASCADE
from .city import City


class CityRegion(Base, ReprAttributesString):
    __tablename__ = 'city_region'

    def __init__(self, name: str, city_id: int) -> None:
        self.name = name
        self.city_id = city_id

    id = Column(Integer, primary_key=True, autoincrement=True)
    name = Column(String)
    city_id = Column(Integer, ForeignKey(
        City.__tablename__ + ID_STR, ondelete=CASCADE), nullable=False)
```

obl.py

```
from sqlalchemy import Column, String, Integer

from . import Base, ReprAttributesString


class Obl(Base, ReprAttributesString):
    __tablename__ = 'obl'

    def __init__(self, name: str) -> None:
        self.name = name

    id = Column(Integer, primary_key=True, autoincrement=True)
    name = Column(String)
```

region.py

```
from sqlalchemy import Column, String, Integer, ForeignKey

from . import Base, ReprAttributesString, ID_STR, CASCADE
from .obl import Obl


class Region(Base, ReprAttributesString):
    __tablename__ = 'region'

    def __init__(self, name: str, obl_id: int) -> None:
        self.name = name
        self.obl_id = obl_id

    id = Column(Integer, primary_key=True, autoincrement=True)
    name = Column(String)
    obl_id = Column(Integer, ForeignKey(
        Obl.__tablename__ + ID_STR, ondelete=CASCADE), nullable=False)
```

street.py

```
from sqlalchemy import Column, String, Integer, ForeignKey

from . import Base, ReprAttributesString, ID_STR, CASCADE
from .city_region import CityRegion
```

```
class Street(Base, ReprAttributesString):
    __tablename__ = 'street'

    def __init__(self, name: str, city_region_id: int) -> None:
        self.name = name
        self.city_region_id = city_region_id

    id = Column(Integer, primary_key=True, autoincrement=True)
    name = Column(String)
    city_region_id = Column(Integer, ForeignKey(
        CityRegion.__tablename__ + ID_STR, ondelete=CASCADE), nullable=False)
```

communal_property.py

__init__.py

```
from .. import Base, ReprAttributesString, ID_STR, CASCADE
from ..administrative_unit import Street
from .balance_keeper import BalanceKeeper
from .structure import Structure
from .component import Component
from .property import Property
```

balance_keeper.py

```
from sqlalchemy import Column, String, Integer

from . import Base, ReprAttributesString


class BalanceKeeper(Base, ReprAttributesString):
    __tablename__ = 'balance_keeper'

    def __init__(self, name: str) -> None:
        self.name = name

    id = Column(Integer, primary_key=True, autoincrement=True)
    name = Column(String)
```

component.py

```
from sqlalchemy import Column, String, Integer

from . import Base, ReprAttributesString


class Component(Base, ReprAttributesString):
    __tablename__ = 'component'

    def __init__(self, name: str) -> None:
        self.name = name

    id = Column(Integer, primary_key=True, autoincrement=True)
    name = Column(String)
```

property.py

```
from sqlalchemy import Column, String, Integer, ForeignKey, Float

from . import Base, ReprAttributesString, ID_STR, CASCADE, Street
from .balance_keeper import BalanceKeeper
from .structure import Structure
from .component import Component
```

```
class Property(Base, ReprAttributesString):
    __tablename__ = 'property'

    def __init__(self, balance_keeper_id: int, street_id: int,
                 structure_id: int, area: Float, land_area: float,
                 component_id: int, component_area: float) -> None:
        self.balance_keeper_id = balance_keeper_id
        self.street_id = street_id
        self.structure_id = structure_id
        self.area = area
        self.land_area = land_area
        self.component_id = component_id
        self.component_area = component_area

    id = Column(Integer, primary_key=True, autoincrement=True)
    balance_keeper_id = Column(Integer, ForeignKey(
        BalanceKeeper.__tablename__ + ID_STR, ondelete=CASCADE), nullable=False)
    street_id = Column(Integer, ForeignKey(
        Street.__tablename__ + ID_STR, ondelete=CASCADE), nullable=False)
    structure_id = Column(Integer, ForeignKey(
        Structure.__tablename__ + ID_STR, ondelete=CASCADE), nullable=False)
    area = Column(Float)
    land_area = Column(Float)
    component_id = Column(Integer, ForeignKey(
        Component.__tablename__ + ID_STR, ondelete=CASCADE), nullable=False)
    component_area = Column(Float)
```

structure.py

```
from sqlalchemy import Column, String, Integer

from . import Base, ReprAttributesString


class Structure(Base, ReprAttributesString):
    __tablename__ = 'structure'

    def __init__(self, name: str) -> None:
        self.name = name

    id = Column(Integer, primary_key=True, autoincrement=True)
    name = Column(String)
```

transport_info.py

__init__.py

```
from .. import Base, ReprAttributesString, ID_STR, CASCADE
from .operation import Operation
from .department import Department
from .brand import Brand
from .model import Model
from .color import Color
from .kind import Kind
from .body import Body
from .purpose import Purpose
from .fuel import Fuel
from .transport import Transport
```

body.py

```
from sqlalchemy import Column, String, Integer

from . import Base, ReprAttributesString
```

```
class Body(Base, ReprAttributesString):
    __tablename__ = 'body'

    def __init__(self, name: str) -> None:
        self.name = name

    id = Column(Integer, primary_key=True, autoincrement=True)
    name = Column(String)
```

brand.py

```
from sqlalchemy import Column, String, Integer

from . import Base, ReprAttributesString


class Brand(Base, ReprAttributesString):
    __tablename__ = 'brand'

    def __init__(self, name: str) -> None:
        self.name = name

    id = Column(Integer, primary_key=True, autoincrement=True)
    name = Column(String)
```

color.py

```
from sqlalchemy import Column, String, Integer

from . import Base, ReprAttributesString


class Color(Base, ReprAttributesString):
    __tablename__ = 'color'

    def __init__(self, name: str) -> None:
        self.name = name

    id = Column(Integer, primary_key=True, autoincrement=True)
    name = Column(String)
```

department.py

```
from sqlalchemy import Column, String, Integer

from . import Base, ReprAttributesString


class Department(Base, ReprAttributesString):
    __tablename__ = 'department'

    def __init__(self, id: int, name: str) -> None:
        self.id = id
        self.name = name

    id = Column(Integer, primary_key=True)
    name = Column(String)
```

fuel.py

```
from sqlalchemy import Column, String, Integer

from . import Base, ReprAttributesString


class Fuel(Base, ReprAttributesString):
    __tablename__ = 'fuel'
```

```
    def __init__(self, name: str) -> None:
        self.name = name

    id = Column(Integer, primary_key=True, autoincrement=True)
    name = Column(String)
```

kind.py

```
from sqlalchemy import Column, String, Integer

from . import Base, ReprAttributesString


class Kind(Base, ReprAttributesString):
    __tablename__ = 'kind'

    def __init__(self, name: str) -> None:
        self.name = name

    id = Column(Integer, primary_key=True, autoincrement=True)
    name = Column(String)
```

model.py

```
from sqlalchemy import Column, String, Integer, ForeignKey

from . import Base, ReprAttributesString, ID_STR, CASCADE
from .brand import Brand


class Model(Base, ReprAttributesString):
    __tablename__ = 'model'

    def __init__(self, name: str, brand_id: int) -> None:
        self.name = name
        self.brand_id = brand_id

    id = Column(Integer, primary_key=True, autoincrement=True)
    name = Column(String)
    brand_id = Column(Integer, ForeignKey(
        Brand.__tablename__ + ID_STR, ondelete=CASCADE), nullable=False)
```

operation.py

```
from sqlalchemy import Column, String, Integer

from . import Base, ReprAttributesString


class Operation(Base, ReprAttributesString):
    __tablename__ = 'operation'

    def __init__(self, id: int, name: str) -> None:
        self.id = id
        self.name = name

    id = Column(Integer, primary_key=True)
    name = Column(String)
```

purpose.py

```
from sqlalchemy import Column, String, Integer

from . import Base, ReprAttributesString


```

```
class Purpose(Base, ReprAttributesString):
    __tablename__ = 'purpose'

    def __init__(self, name: str) -> None:
        self.name = name

    id = Column(Integer, primary_key=True, autoincrement=True)
    name = Column(String)
```

transport.py

```
from sqlalchemy import Column, String, Integer, Date, ForeignKey

from . import Base, ReprAttributesString, ID_STR, CASCADE
from .operation import Operation
from .department import Department
from .model import Model
from .color import Color
from .kind import Kind
from .body import Body
from .purpose import Purpose
from .fuel import Fuel


class Transport(Base, ReprAttributesString):
    __tablename__ = 'transport'

    def __init__(self, reg_addr_koatuu: str, operation_id: int,
                 d_reg: str, department_id: int, model_id: int, vin: str,
                 make_year: int, color_id: int, kind_id: int, body_id: int,
                 purpose_id: int, fuel_id: int, capacity: int,
                 own_weight: int, total_weight: int, n_reg_new: str) -> None:
        self.reg_addr_koatuu = reg_addr_koatuu
        self.operation_id = operation_id
        self.d_reg = d_reg
        self.department_id = department_id
        self.model_id = model_id
        self.vin = vin
        self.make_year = make_year
        self.color_id = color_id
        self.kind_id = kind_id
        self.body_id = body_id
        self.purpose_id = purpose_id
        self.fuel_id = fuel_id
        self.capacity = capacity
        self.own_weight = own_weight
        self.total_weight = total_weight
        self.n_reg_new = n_reg_new

    id = Column(Integer, primary_key=True, autoincrement=True)
    reg_addr_koatuu = Column(String)
    operation_id = Column(Integer, ForeignKey(
        Operation.__tablename__ + ID_STR, ondelete=CASCADE), nullable=False)
    d_reg = Column(Date)
    department_id = Column(Integer, ForeignKey(
        Department.__tablename__ + ID_STR, ondelete=CASCADE), nullable=False)
    model_id = Column(Integer, ForeignKey(
        Model.__tablename__ + ID_STR, ondelete=CASCADE), nullable=False)
    vin = Column(String)
    make_year = Column(Integer)
    color_id = Column(Integer, ForeignKey(Color.__tablename__ + ID_STR, ondelete=CASCADE),
nullable=False)
    kind_id = Column(Integer, ForeignKey(
```

```
        Kind.__tablename__ + ID_STR, ondelete=CASCADE), nullable=False)
    body_id = Column(Integer, ForeignKey(
        Body.__tablename__ + ID_STR, ondelete=CASCADE), nullable=False)
    purpose_id = Column(Integer, ForeignKey(
        Purpose.__tablename__ + ID_STR, ondelete=CASCADE), nullable=False)
    fuel_id = Column(Integer, ForeignKey(
        Fuel.__tablename__ + ID_STR, ondelete=CASCADE), nullable=False)
    capacity = Column(Integer)
    own_weight = Column(Integer)
    total_weight = Column(Integer)
    n_reg_new = Column(String)
```

communal_transport.py

```
from sqlalchemy import Column, Integer, ForeignKey

from . import Base, ReprAttributesString, ID_STR, CASCADE
from my_models.main_models.transport_info import Transport
from my_models.main_models.communal_property import Property


class CommunalTransport(Base, ReprAttributesString):
    __tablename__ = 'communal_transport'

    def __init__(self, property_id: int, transport_id: int) -> None:
        self.property_id = property_id
        self.transport_id = transport_id

    id = Column(Integer, primary_key=True, autoincrement=True)
    property_id = Column(Integer, ForeignKey(
        Property.__tablename__ + ID_STR, ondelete=CASCADE), nullable=False)
    transport_id = Column(Integer, ForeignKey(
        Transport.__tablename__ + ID_STR, ondelete=CASCADE), nullable=False)
```

staging.py

```
from my_etl_controller import get_main_engine, get_stage_engine, get_session, Extractor
from my_models import stage_models

def staging():
    stage_engine = get_stage_engine(True)
    stage_models.Base.metadata.create_all(stage_engine)
    stage_session = get_session(stage_engine)
    extractor = Extractor(stage_session)
    extractor.extract_data()

staging()
```

ETL-Засоби:

extractor.py

```
import pandas as pd
import numpy as np
from sqlalchemy.orm import Session
from my_models.stage_models import AdministrativeUnit, CommunalProperty, TransportInfo
```

```python
class Extractor:
    transport_info_file: str = 'data_sets/small_transport_info.csv'
    transport_info_delim: str = ';'
    communal_property_file: str = 'data_sets/communal_property.xls'
    administrative_unit_file: str = 'data_sets/administrative_unit.xml'

    def __init__(self, stage_session: Session):
        self.stage_session: Session = stage_session

    def extract_data(self):
        self.extract_communal_property_data()
        self.extract_transport_info_data()
        self.extract_administrative_unit_data()

    def extract_transport_info_data(self) -> None:
        data = pd.read_csv(self.transport_info_file,
                           delimiter=self.transport_info_delim)
        data = data.replace({np.nan: None})
        for el in data.values:
            self.stage_session.add(TransportInfo(*el[1:]))
        self.stage_session.commit()

    def extract_communal_property_data(self) -> None:
        data = pd.read_excel(self.communal_property_file)
        data['components_area'] = data['components_area'].str.replace(',', '.').astype(float)
        data['land_area'] = data['land_area'].str.replace(',', '.').astype(float)
        data['object_area'] = data['object_area'].str.replace(',', '.').astype(float)
        data = data.replace({np.nan: None})
        for el in data.values:
            self.stage_session.add(CommunalProperty(*el))
        self.stage_session.commit()

    def extract_administrative_unit_data(self) -> None:
        data = pd.read_xml(self.administrative_unit_file)
        data = data.replace({np.nan: None})
        for el in data.values:
            self.stage_session.add(AdministrativeUnit(*el))
        self.stage_session.commit()
```

transloader.py
__init__.py

```python
from abc import ABC, abstractmethod
from typing import Any

from sqlalchemy import MetaData
from sqlalchemy.engine import Engine
from my_models.main_models import Base

from my_etl_controller.connector import get_session


class TransLoader(ABC):

    def __init__(self, stage_engine: Engine, main_engine: Engine) -> None:
        self.stage_engine = stage_engine
        self.main_engine = main_engine
        self.stage_session = get_session(stage_engine)
        self.main_session = get_session(main_engine)
        self.meta = MetaData()
```

```
    def update_row_and_latest_id(self, value: str, d: dict[str, int], latest_id_ref: [int]) -> int:
        id = d.get(value)
        if id is None:
            id = latest_id_ref[0]
            d.update([(value, id)])
            latest_id_ref[0] = latest_id_ref[0] + 1
        return id

    def load_from_dict(self, d: dict[Any, Any], model: Base, autoincrement=True) -> None:
        if autoincrement:
            for name, _ in d.items():
                self.main_session.add(model(name))
        else:
            for name, id in d.items():
                self.main_session.add(model(id, name))
        self.main_session.commit()

    def load_from_list(self, l: list[tuple], model: Base) -> None:
        for row in l:
            self.main_session.add(model(*row))
        self.main_session.commit()

    @abstractmethod
    def models(self) -> list[Base]:
        pass

    def create_models(self):
        tables = list(map(lambda x: Base.metadata.tables[x.__tablename__], self.models()))
        Base.metadata.create_all(bind=self.main_engine, tables=tables)

    @abstractmethod
    def transform(self) -> None:
        pass

    @abstractmethod
    def load(self) -> None:
        pass

    def transload(self) -> None:
        self.create_models()
        self.transform()
        self.load()


from .au_transloader import AUTransLoader
from .cp_transloader import CPTransLoader
from .ti_transloader import TITransLoader
from .ct_transloader import CTTransLoader


def transform_data(stage_engine: Engine, main_engine: Engine) -> None:
    AUTransLoader(stage_engine, main_engine).transload()
    CPTransLoader(stage_engine, main_engine).transload()
    TITransLoader(stage_engine, main_engine).transload()
    CTTransLoader(stage_engine, main_engine).transload()
```

au_transloader.py

```
import pandas as pd
from sqlalchemy.engine import Engine
from my_models.stage_models import AdministrativeUnit
```

```python
from my_models.main_models import Base
from my_models.main_models.administrative_unit import Obl, Region, City, CityRegion, Street
from . import TransLoader


class AUTransLoader(TransLoader):

    def __init__(self, stage_engine: Engine, main_engine: Engine) -> None:
        TransLoader.__init__(self, stage_engine, main_engine)
        self.df_administrative_unit = pd.read_sql_table(
            AdministrativeUnit.__tablename__, self.stage_engine)
        self.table = {}

    def models(self) -> list[Base]:
        return [Obl, Region, City, CityRegion, Street]

    def transform(self) -> None:
        for i, row in self.df_administrative_unit.iterrows():
            _, obl_name, region_name, city_name, city_region_name, street_name = row
            regions = self.table.get(obl_name)
            if regions is None:
                regions = {}
                self.table.update([(obl_name, regions)])
            cities = regions.get(region_name)
            if cities is None:
                cities = {}
                regions.update([(region_name, cities)])
            city_regions = cities.get(city_name)
            if city_regions is None:
                city_regions = {}
                cities.update([(city_name, city_regions)])
            streets = city_regions.get(city_region_name)
            if streets is None:
                streets = []
                city_regions.update([(city_region_name, streets)])
            streets.append(street_name)

    def load_obl(self):
        for obl_name, regions in self.table.items():
            self.main_session.add(Obl(obl_name))
        self.main_session.commit()

    def load_region(self):
        obl_id = 1
        for obl_name, regions in self.table.items():
            for region_name, cities in regions.items():
                self.main_session.add(Region(region_name, obl_id))
            obl_id = obl_id + 1
        self.main_session.commit()

    def load_city(self):
        region_id = 1
        for obl_name, regions in self.table.items():
            for region_name, cities in regions.items():
                for city_name, city_regions in cities.items():
                    self.main_session.add(City(city_name, region_id))
                region_id = region_id + 1
        self.main_session.commit()

    def load_city_region(self):
        city_id = 1
```

```
        for obl_name, regions in self.table.items():
            for region_name, cities in regions.items():
                for city_name, city_regions in cities.items():
                    for city_region_name, streets in city_regions.items():
                        self.main_session.add(CityRegion(city_region_name, city_id))
                    city_id = city_id + 1
        self.main_session.commit()

    def load_street(self):
        city_region_id = 1
        for obl_name, regions in self.table.items():
            for region_name, cities in regions.items():
                for city_name, city_regions in cities.items():
                    for city_region_name, streets in city_regions.items():
                        for street_name in streets:
                            self.main_session.add(Street(street_name, city_region_id))
                        city_region_id = city_region_id + 1
        self.main_session.commit()

    def load(self):
        self.load_obl()
        self.load_region()
        self.load_city()
        self.load_city_region()
        self.load_street()
```

cp_transloader.py

```
import pandas as pd
import numpy as np
from sqlalchemy.engine import Engine
from my_models.main_models import Base
from my_models.stage_models import CommunalProperty
from my_models.main_models.communal_property import BalanceKeeper, Component,
Property, Structure
from my_models.main_models.administrative_unit import Street
from . import TransLoader


class CPTransLoader(TransLoader):

    def __init__(self, stage_engine: Engine, main_engine: Engine) -> None:
        TransLoader.__init__(self, stage_engine, main_engine)
        self.df_communal_property = pd.read_sql_table(
            CommunalProperty.__tablename__, self.stage_engine)
        self.property: list[tuple[int, int, int, float, float, int, float]] = []
        self.balance_keeper: dict[str, int] = {}
        self.component: dict[str, int] = {}
        self.structure: dict[str, int] = {}

    def models(self) -> list[Base]:
        return [BalanceKeeper, Component, Property, Structure]

    def transform(self) -> None:
        ids = {'bk_id': [1], 'struct_id': [1], 'comp_id': [1]}
        street_count = self.main_session.query(Street).count()
        for _, row in self.df_communal_property.iterrows():
            _, balance_keeper_name, _, structure_name, area, land_area, components_name,
component_area, _ = row
            bk_id = self.update_row_and_latest_id(
                balance_keeper_name, self.balance_keeper, ids['bk_id'])
```

```
            struct_id = self.update_row_and_latest_id(
                structure_name, self.structure, ids['struct_id'])
            comp_id = self.update_row_and_latest_id(
                components_name, self.component, ids['comp_id'])
            street_id = np.random.randint(1, street_count)
            self.property.append(
                (bk_id, street_id, struct_id, area, land_area, comp_id, component_area))

    def load(self) -> None:
        self.load_from_dict(self.balance_keeper, BalanceKeeper)
        self.load_from_dict(self.component, Component)
        self.load_from_dict(self.structure, Structure)
        self.load_from_list(self.property, Property)
```

ct_transloader.py

```
import pandas as pd
import numpy as np
from sqlalchemy.engine import Engine
from my_models.main_models import Base
from my_models.main_models.transport_info import Transport
from my_models.main_models.communal_property import Property
from my_models.main_models.communal_transport import CommunalTransport
from . import TransLoader


class CTTransLoader(TransLoader):

    def __init__(self, stage_engine: Engine, main_engine: Engine) -> None:
        TransLoader.__init__(self, stage_engine, main_engine)
        self.df_transport = pd.read_sql_table(
            Transport.__tablename__, self.main_engine)
        self.df_property = pd.read_sql_table(
            Property.__tablename__, self.main_engine)
        self.table: [(int, int)] = []

    def models(self) -> list[Base]:
        return [CommunalTransport]

    def transform(self) -> None:
        property_count = self.main_session.query(Property).count()
        for _, row in self.df_transport.iterrows():
            self.table.append((np.random.randint(1, property_count), row[0]))

    def load(self):
        self.load_from_list(self.table, CommunalTransport)
```

ti_transloader.py

```
import pandas as pd
import numpy as np
from sqlalchemy.engine import Engine
from my_models.stage_models import TransportInfo
from my_models.main_models.transport_info import Body, Brand, Color, \
    Department, Fuel, Kind, Model, Operation, Purpose, Transport
from my_models.main_models import Base
from . import TransLoader
```

```
class TITransLoader(TransLoader):

    def __init__(self, stage_engine: Engine, main_engine: Engine) -> None:
        TransLoader.__init__(self, stage_engine, main_engine)
        self.df_transport_info = pd.read_sql_table(
            TransportInfo.__tablename__, self.stage_engine)
        self.transport: list[tuple] = []
        self.color: dict[str, int] = {}
        self.kind: dict[str, int] = {}
        self.fuel: dict[str, int] = {}
        self.body: dict[str, int] = {}
        self.purpose: dict[str, int] = {}
        self.reg_addr_koatuu: dict[int, int] = {}
        self.purpose: dict[str, int] = {}
        self.operation: dict[str, int] = {}
        self.department: dict[str, int] = {}
        self.purpose: dict[str, int] = {}
        # brand contains models
        self.brand: dict[str, dict[str, int]] = {}

    def models(self) -> list[Base]:
        return [Body, Brand, Color, Department, Fuel,
                Kind, Model, Operation, Purpose, Transport]

    def transform_brand_model(self) -> None:
        model_id = 1
        for _, row in self.df_transport_info.iterrows():
            brand_name, model_name = (row[7], row[8])
            models = self.brand.get(brand_name)
            if models is None:
                models = {}
                self.brand.update([(brand_name, models)])
            if models.get(model_name) is None:
                models.update([(model_name, model_id)])
                model_id = model_id + 1

    def load_model(self) -> None:
        brand_id = 1
        for _, models in self.brand.items():
            for model_name, _ in models.items():
                self.main_session.add(Model(model_name, brand_id))
            brand_id = brand_id + 1
        self.main_session.commit()

    def nan_to_zero(self, number: int) -> int:
        result = number
        if np.isnan(result):
            result = 0
        return result

    def transform(self) -> None:
        self.transform_brand_model()
        ids = {'color_id': [1], 'kind_id': [1], 'fuel_id': [1],
               'body_id': [1], 'purpose_id': [1]}
        for _, row in self.df_transport_info.iterrows():
            _, reg_addr_koatuu, oper_code, oper_name, d_reg, \
                dep_code, dep_name, brand_name, model_name, vin, make_year, color_name, \
                kind_name, body_name, purpose_name, fuel_name, \
                capacity, own_weight, total_weight, n_reg_new = row
            color_id = self.update_row_and_latest_id(color_name, self.color, ids['color_id'])
```

```
            kind_id = self.update_row_and_latest_id(kind_name, self.kind, ids['kind_id'])
            fuel_id = self.update_row_and_latest_id(fuel_name, self.fuel, ids['fuel_id'])
            body_id = self.update_row_and_latest_id(body_name, self.body, ids['body_id'])
            purpose_id = self.update_row_and_latest_id(
                purpose_name, self.purpose, ids['purpose_id'])
            self.department.update([(dep_name, dep_code)])
            self.operation.update([(oper_name, oper_code)])
            model_id = self.brand[brand_name][model_name]
            capacity = self.nan_to_zero(capacity)
            own_weight = self.nan_to_zero(own_weight)
            total_weight = self.nan_to_zero(total_weight)
            self.transport.append(
                (reg_addr_koatuu, oper_code, d_reg, dep_code, model_id,
                 vin, make_year, color_id, kind_id, body_id, purpose_id,
                 fuel_id, capacity, own_weight, total_weight, n_reg_new))

    def load(self) -> None:
        self.load_from_dict(self.brand, Brand)
        self.load_model()
        self.load_from_dict(self.color, Color)
        self.load_from_dict(self.kind, Kind)
        self.load_from_dict(self.fuel, Fuel)
        self.load_from_dict(self.body, Body)
        self.load_from_dict(self.purpose, Purpose)
        self.load_from_dict(self.operation, Operation, autoincrement=False)
        self.load_from_dict(self.department, Department, autoincrement=False)
        self.load_from_list(self.transport, Transport)
```

main.py

```
from my_etl_controller import get_main_engine, get_stage_engine, transform_data

def main():
    stage_engine = get_stage_engine(False)
    main_engine = get_main_engine(False)
    transform_data(stage_engine, main_engine)

if __name__ == '__main__':
    main()
```

Інші файли:

my_config.py

__init__.py

```
keys = ['user', 'passwd', 'host', 'port', 'db']
stage_connector = {
    keys[0]: 'postgres',
    keys[1]: 'qwerty',
    keys[2]: 'localhost',
    keys[3]: 5432,
    keys[4]: 'da_lab_one_stage'
}
main_connector = {
    keys[0]: 'postgres',
    keys[1]: 'qwerty',
    keys[2]: 'localhost',
    keys[3]: 5432,
    keys[4]: 'da_lab_one_main'
}
```

connector.py

```
from sqlalchemy import create_engine
from sqlalchemy.engine import Engine
from sqlalchemy.orm import sessionmaker, Session
from sqlalchemy_utils import database_exists, create_database, drop_database
import my_config


def get_engine(user: str, passwd: str, host: str, port: int, db: str, is_recreate_db: bool) ->
Engine:
    url = f"postgresql://{user}:{passwd}@{host}:{port}/{db}"
    if is_recreate_db:
        if database_exists(url):
            drop_database(url)
        create_database(url)
    engine = create_engine(url, pool_size=50, echo=True)
    return engine


def check_connect_info(info: dict) -> None:
    if not all(key in my_config.keys for key in info.keys()):
        raise Exception('Bad info connect')


def get_stage_engine(is_recreate_db: bool) -> Engine:
    check_connect_info(my_config.stage_connector)
    info = my_config.stage_connector
    return get_engine(
        user=info['user'],
        passwd=info['passwd'],
        host=info['host'],
        port=info['port'],
        db=info['db'],
        is_recreate_db=is_recreate_db
    )


def get_main_engine(is_recreate_db: bool) -> Engine:
    check_connect_info(my_config.main_connector)
    info = my_config.main_connector
    return get_engine(
        user=info['user'],
        passwd=info['passwd'],
        host=info['host'],
        port=info['port'],
        db=info['db'],
        is_recreate_db=is_recreate_db
    )


def get_session(engine: Engine) -> Session:
    return sessionmaker(bind=engine)()
```

constants.py

```
ID_STR = '.id'
CASCADE = 'CASCADE'
```

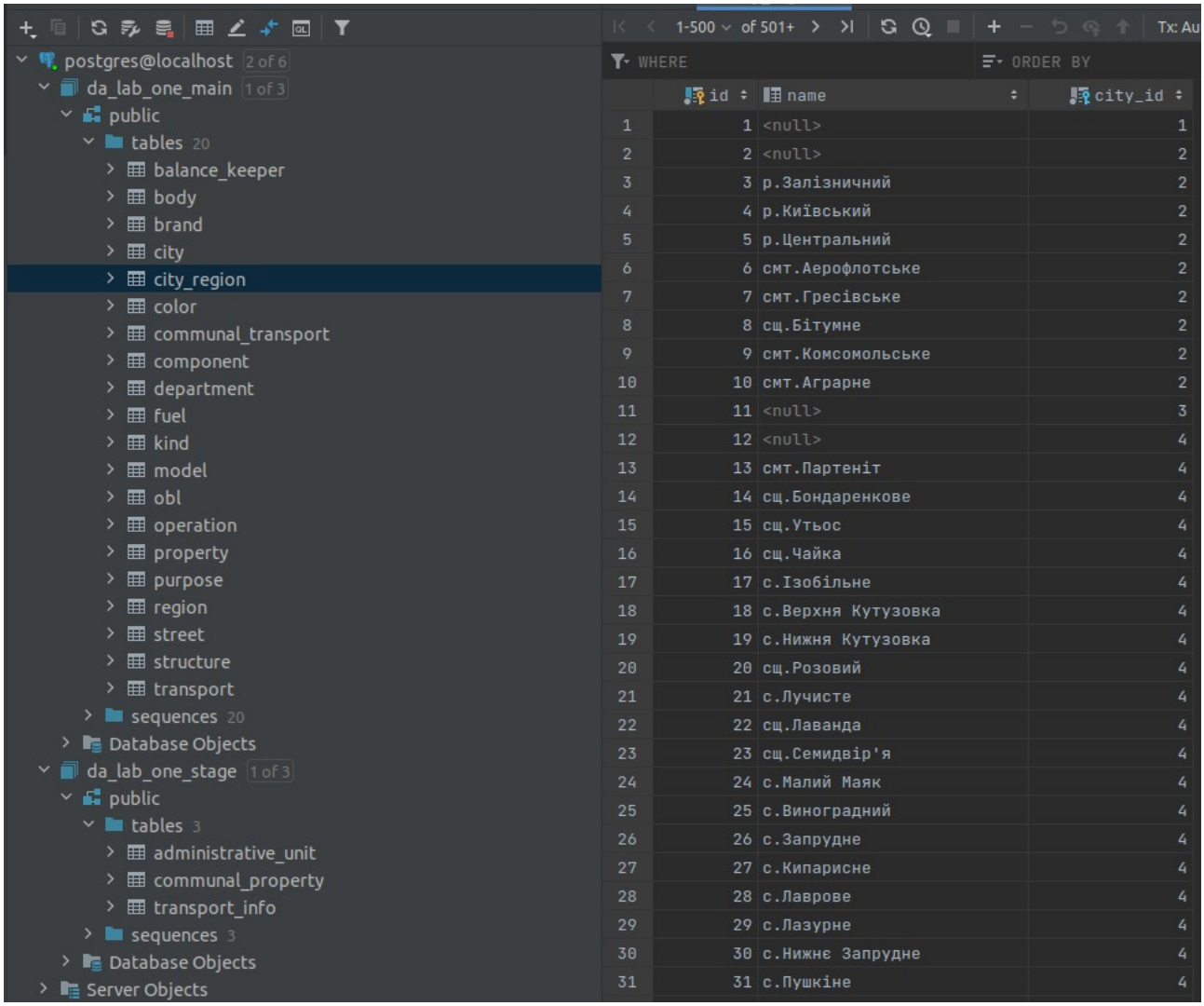
Приклад завантажених даних:

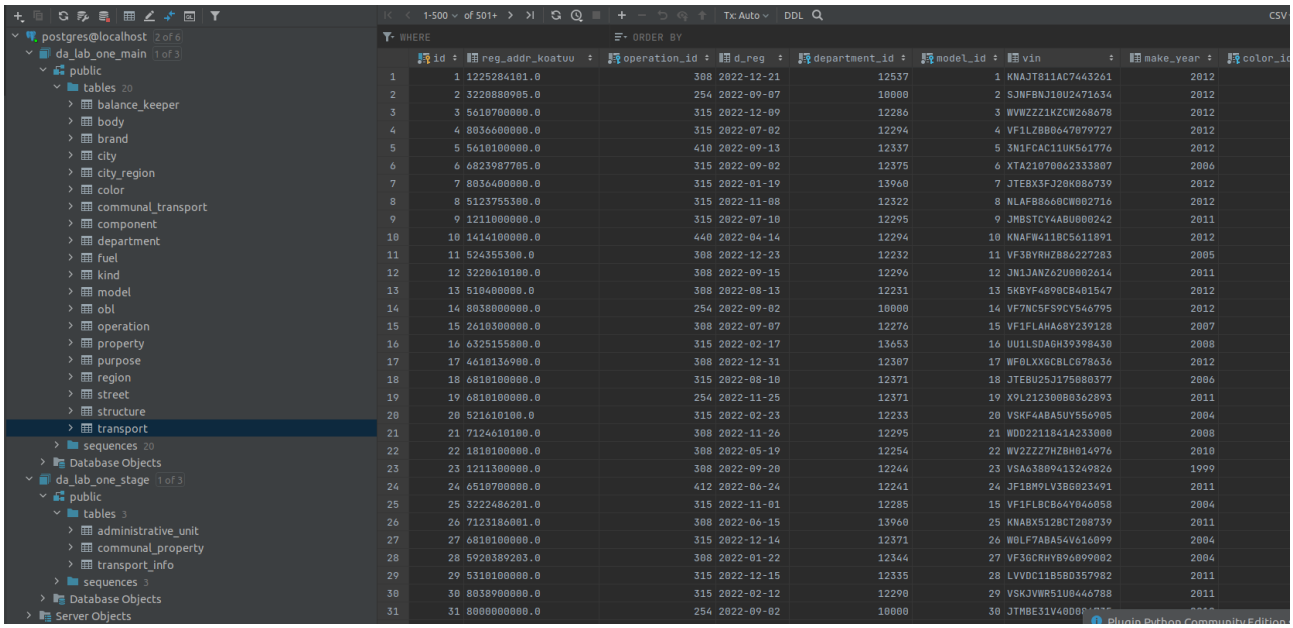
```
2023-01-20 21:20:49,152 INFO sqlalchemy.engine.Engine [no key 0.00012s] {}
2023-01-20 21:20:49,160 INFO sqlalchemy.engine.Engine COMMIT
2023-01-20 21:20:49,161 INFO sqlalchemy.engine.Engine BEGIN (implicit)
2023-01-20 21:20:49,164 INFO sqlalchemy.engine.Engine SELECT count(*) AS count_1
FROM (SELECT property.id AS property_id, property.balance_keeper_id AS property_balance_keeper_id, property.street_
FROM property) AS anon_1
2023-01-20 21:20:49,164 INFO sqlalchemy.engine.Engine [generated in 0.00017s] {}
2023-01-20 21:20:51,186 INFO sqlalchemy.engine.Engine INSERT INTO communal_transport (property_id, transport_id) VA
2023-01-20 21:20:51,186 INFO sqlalchemy.engine.Engine [generated in 0.03811s] ({'property_id': 4829, 'transport_id'
2023-01-20 21:20:52,206 INFO sqlalchemy.engine.Engine COMMIT

Process finished with exit code 0
```

```
2023-01-20 21:20:49,151 INFO sqlalchemy.engine.Engine select relname from pg_class c join pg_namespace n on n.oid=c
2023-01-20 21:20:49,151 INFO sqlalchemy.engine.Engine [cached since 86.11s ago] {'name': 'communal_transport'}
2023-01-20 21:20:49,152 INFO sqlalchemy.engine.Engine
CREATE TABLE communal_transport (
    id SERIAL NOT NULL,
    property_id INTEGER NOT NULL,
    transport_id INTEGER NOT NULL,
    PRIMARY KEY (id),
    FOREIGN KEY(property_id) REFERENCES property (id) ON DELETE CASCADE,
    FOREIGN KEY(transport_id) REFERENCES transport (id) ON DELETE CASCADE
)


2023-01-20 21:20:49,152 INFO sqlalchemy.engine.Engine [no key 0.00012s] {}
```


postgres@localhost 2 of 6
da_lab_one_main 1 of 3
public
tables 20
balance_keeper
body
brand
city
city_region
color
communal_transport
component
department
fuel
kind
model
obl
operation
property
purpose
region
street
structure
transport
sequences 20
Database Objects
da_lab_one_stage 1 of 3
public
tables 3
administrative_unit
communal_property
transport_info
sequences 3
Database Objects
Server Objects
1-500 of 501+
Tx: Au
WHERE
ORDER BY
id
name
city_id
1 1 <null> 1
2 2 <null> 2
3 3 р.Залізничний 2
4 4 р.Київський 2
5 5 р.Центральний 2
6 6 смт.Аерофлотське 2
7 7 смт.Гресівське 2
8 8 сщ.Бітумне 2
9 9 смт.Комсомольське 2
10 10 смт.Аграрне 2
11 11 <null> 3
12 12 <null> 4
13 13 смт.Партеніт 4
14 14 сщ.Бондаренкове 4
15 15 сщ.Утьос 4
16 16 сщ.Чайка 4
17 17 с.Ізобільне 4
18 18 с.Верхня Кутузовка 4
19 19 с.Нижня Кутузовка 4
20 20 сщ.Розовий 4
21 21 с.Лучисте 4
22 22 сщ.Лаванда 4
23 23 сщ.Семидвір'я 4
24 24 с.Малий Маяк 4
25 25 с.Виноградний 4
26 26 с.Запрудне 4
27 27 с.Кипарисне 4
28 28 с.Лаврове 4
29 29 с.Лазурне 4
30 30 с.Нижнє Запрудне 4
31 31 с.Пушкіне 4



Висновок:
Під час лабораторної роботи ознайомився з підходами до створення сховищ даних. Була створена Stage зона для витягування даних з файлів. Модель основного сховища даних побудована у схемі “сніжинка”. Код ETL-контроллеру наведений та показаний результат роботи у вигляді кінцевої бази даних, приведеної до 3ьої нормальної форми. Схеми для кращого орієнтування також додані.

Запитання для самоперевірки:

1) Дайте визначення ETL:

ETL – це триетапний процес, де дані витягуються, трансформуються, завантажуються у вихідне сховище даних.

2) На чому базується процес ETL?

ETL базується на трьох етапах:

a) *Витягування* включає у себе такі процеси як: вилучення даних з однорідних або різнорідних джерел; перетворення даних у належний формат/структуру зберігання для цілей запитів та аналізу; вставку даних у кінцеву цільову базу даних, таку як сховище операційних даних, вітрина даних, озеро даних або сховище даних.

b) *Перетворення* даних включає серію правил або функцій, що застосовуються до витягнутих даних, щоб підготувати їх для завантаження в кінцеву ціль. Важливою частиною даного етапу є очищення даних, метою якого є передача лише «належних» даних до вихідного сховища. У інших випадках застосовується: закодовування значень, отримання похідного значення, генерування ключів, деагрегацію повторюваних стовпців тощо.

c) *Завантаження* даних включає перенесення їх у плоский файл або сховище даних.

3) Переваги та недоліки схем «зірка» та сніжинка:

a) “Зірка”:

1. Переваги:
   - Продуктивність запитів підвищується, оскільки вони використовують дуже прості об’єднання під час отримання даних
   - Отримати дані для звітності просто в будь-який момент часу за будь-який період.
2. Недоліки:
   - Схему зірок не рекомендується змінювати та використовувати повторно в довгостроковій перспективі, якщо планується багато змін.
   - Оскільки таблиці не розділені ієрархічно, то з’являється надлишковість даних.

b) “Сніжинка”:

1. Переваги:
   - Надмірність даних повністю вирішується створенням нових таблиць розмірів.
   - У порівнянні із зірковою схемою таблиці розмірів Snow Flaking використовують менше місця.
   - Легко оновити та підтримувати таблиці.
2. Недоліки:
   - Через те, що існують нормалізовані таблиці розмірів, система ETL повинна завантажувати кількість таблиць.
   - Для виконання запиту вам можуть знадобитися складні об’єднання через кількість доданих таблиць. Отже, продуктивність запитів буде погіршена.

4) Підходи до створення багатовимірних моделей?

OLAP (англ. *online analytical processing*, аналітична обробка у реальному часі) - це інтерактивна система що дозволяє переглядати різні підсумки по багатовимірних даних.

Пропонуються різноманітні продукти OLAP, які можна згрупувати в три категорії:

1. багатовимірний ROLAP:
   - ROLAP означає Relational Online Analytical Processing. ROLAP зберігає дані в стовпцях і рядках (також відомих як реляційні таблиці) і отримує інформацію на вимогу за допомогою запитів, наданих користувачем. Доступ до бази даних ROLAP можна отримати за допомогою складних запитів SQL для обчислення інформації. ROLAP може обробляти великі обсяги даних, але чим більше даних, тим повільніше час обробки.
   - Оскільки запити виконуються на вимогу, ROLAP не вимагає зберігання та попереднього обчислення інформації. Однак

недоліком реалізації ROLAP є потенційні обмеження продуктивності та обмеження масштабованості, які є результатом великих і неефективних операцій з'єднання між великими таблицями.

2. реляційний ROLAP:
    - MOLAP означає багатовимірну онлайн-аналітичну обробку. MOLAP використовує багатовимірний куб, який отримує доступ до збережених даних за допомогою різних комбінацій. Дані попередньо обчислюються, підсумовуються та зберігаються (відмінність від ROLAP, де запити обслуговуються на вимогу).
    - Багатокубовий підхід виявився успішним у продуктах MOLAP. У цьому підході серія щільних, маленьких, попередньо обчислених кубів утворює гіперкуб. Інструменти, які включають MOLAP, включають Oracle Essbase, IBM Cognos і Apache Kylin.
3. гібридний HOLAP:
    - HOLAP означає Hybrid Online Analytical Processing. Як випливає з назви, режим зберігання HOLAP поєднує атрибути як MOLAP, так і ROLAP. Оскільки HOLAP передбачає зберігання частини ваших даних у сховищі ROLAP, а іншу частину – у сховищі MOLAP, розробники отримують переваги обох.
    - Завдяки такому використанню двох OLAP дані зберігаються як у багатовимірних базах даних, так і в реляційних базах даних. Рішення про доступ до однієї з баз даних залежить від того, яка з них найбільше підходить для запитаної програми або типу обробки. Це налаштування забезпечує набагато більшу гнучкість обробки даних. Для теоретичної обробки дані зберігаються в багатовимірній базі даних. Для важкої обробки дані зберігаються в реляційній базі даних.

Використані джерела:
How to do ETL: https://www.youtube.com/watch?v=XVms0SA-6Xs
ETL: https://uk.wikipedia.org/wiki/ETL
Snowflake schema: https://en.wikipedia.org/wiki/Snowflake_schema
Типи схем у моделюванні сховища даних - схема Star & SnowFlake: https://uk.myservername.com/schema-types-data-warehouse-modeling-star-snowflake-schema
Багатовимірна модель даних(пункт 1.8): https://elearning.sumdu.edu.ua/free_content/lectured:a8104441b8e00905159c1ff04257b014dd456247/20151109195846/162252/index.html
Багатовимірна модель даних: https://stud.com.ua/35697/informatika/bagatovimirna_model_danih
OLAP, ROLAP, MOLAP, HOLAP: https://www.sisense.com/glossary/olap/
OLAP: https://uk.wikipedia.org/wiki/OLAP

XML parsing in Python: https://www.geeksforgeeks.org/xml-parsing-python/

Python Tutorial: CSV Module - How to Read, Parse, and Write CSV Files: https://www.youtube.com/watch?v=q5uM4VKywbA