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**](https://github.com/SideShowBoBGOT/DataAnalysisFourthSemester)

**!!!**

**Код Stage Моделей:**

stage\_models.py

\_\_init\_\_.py

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

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)

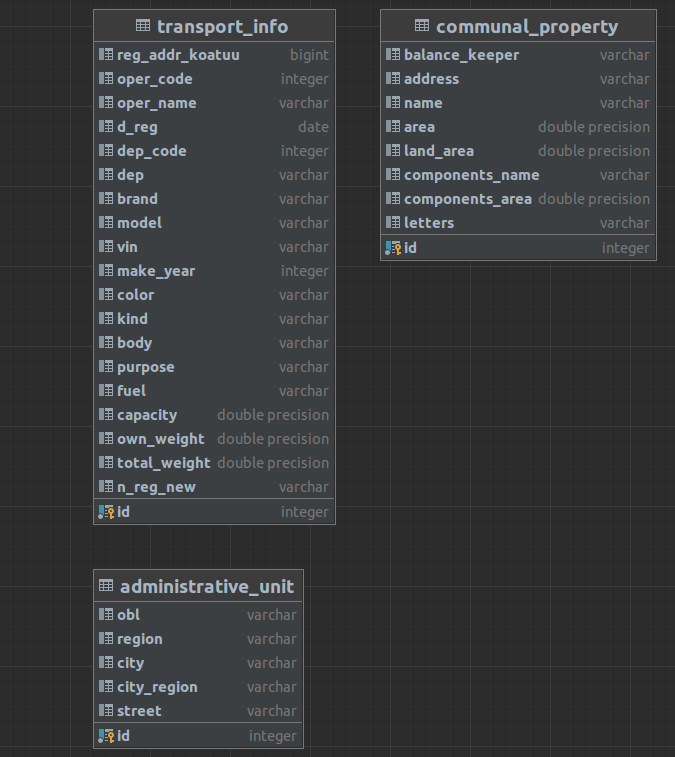
communal\_property.py

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)

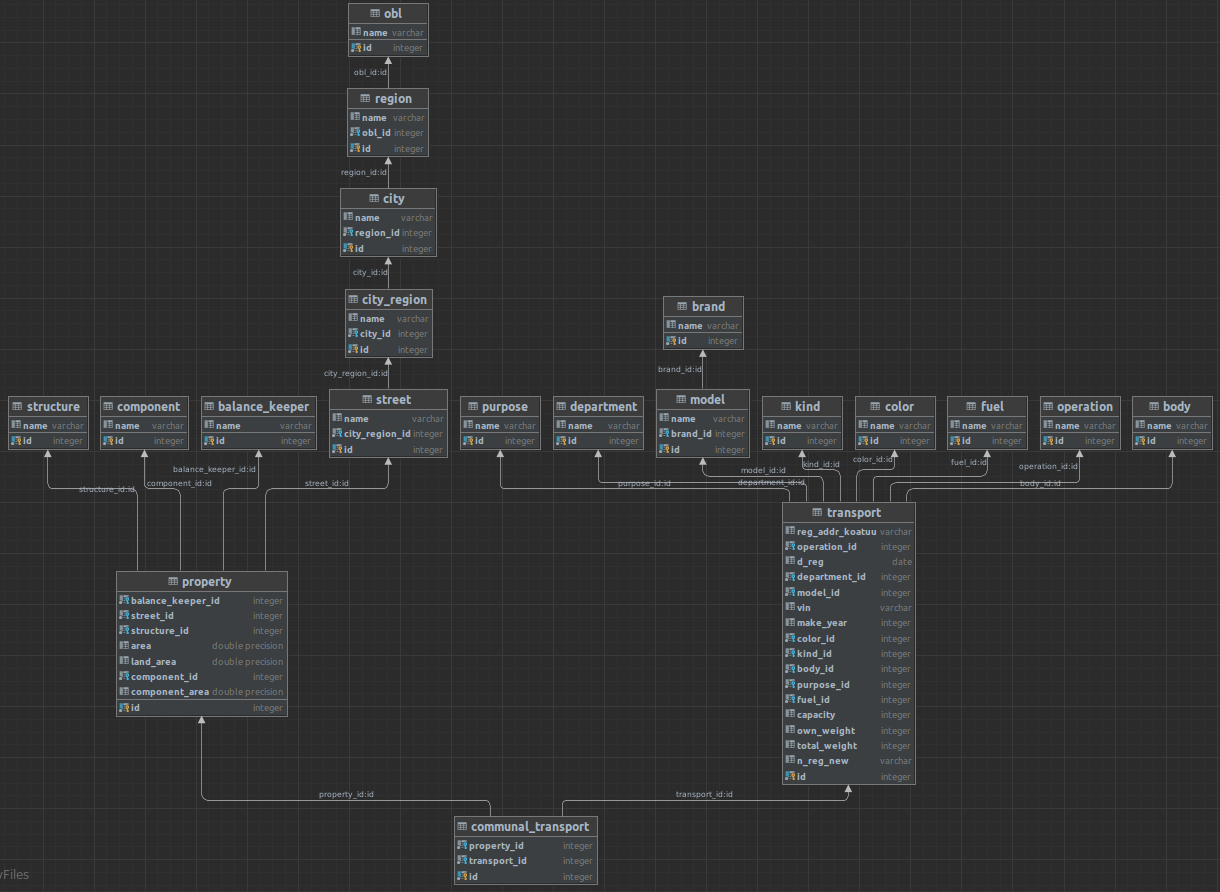
transport\_info.py

from sqlalchemy import Column, String, Integer, BIGINT, Date, Floatfrom ..mixins import ReprAttributesStringfrom . import Baseclass 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  
  
  
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

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  
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'  
}

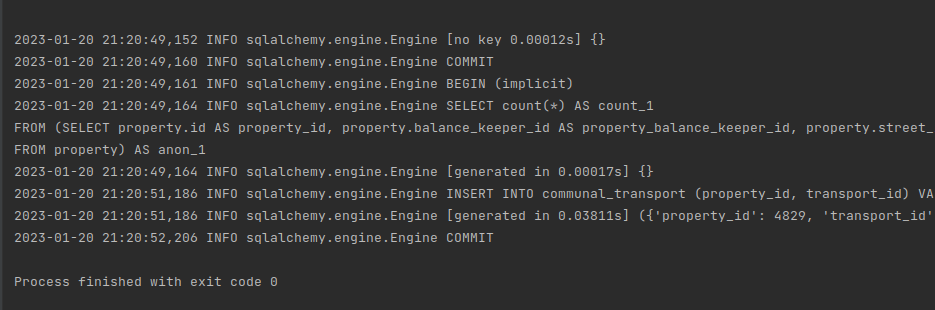
connector.py

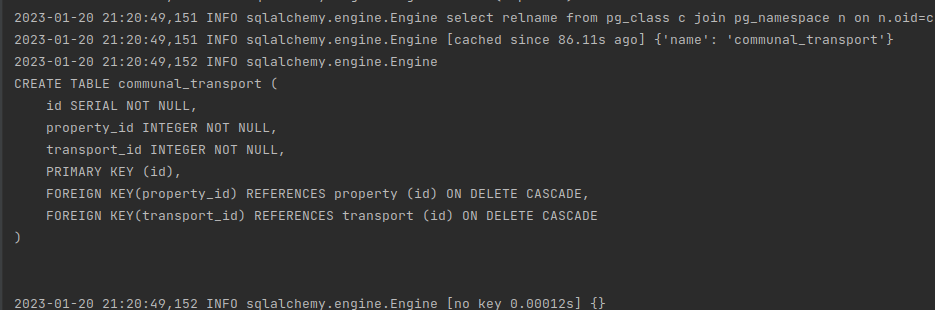
from sqlalchemy import create\_engine  
from sqlalchemy.engine import Engine  
from sqlalchemy.orm import sessionmaker, Session  
from sqlalchemy\_utils import database\_exists, create\_database, drop\_database  
import my\_config  
  
  
def get\_engine(user: str, passwd: str, host: str, port: int, 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)()

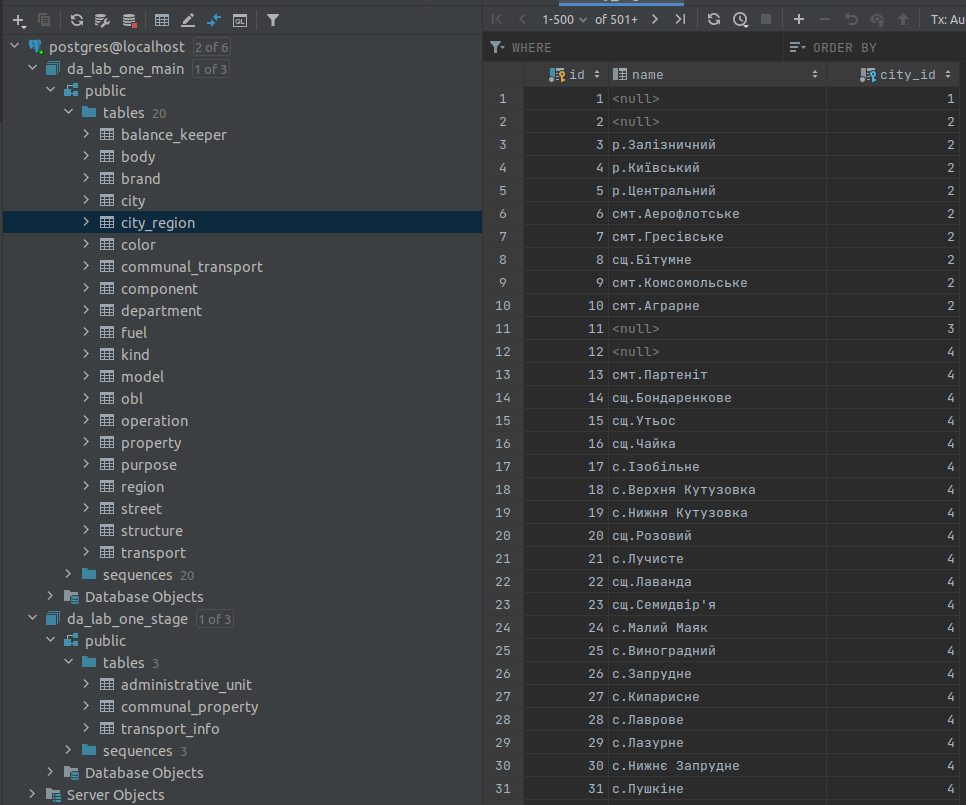
constants.py

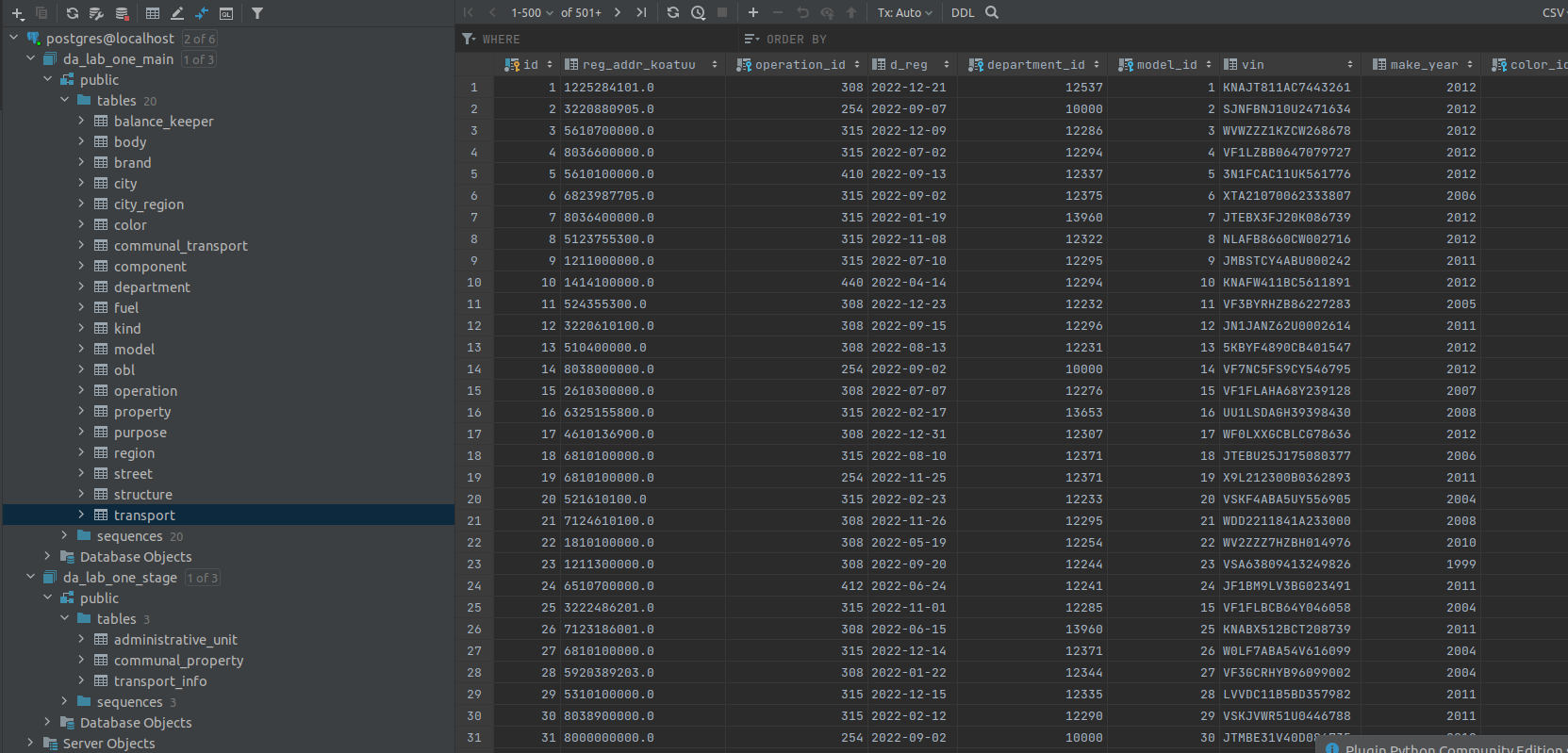
ID\_STR = '.id'  
CASCADE = 'CASCADE'

**Приклад завантажених даних:**









**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:

1. *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.
2. *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.
3. *Завантаження* даних включає перенесення їх у плоский файл або сховище даних.

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

1. “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.
2. “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**](https://www.youtube.com/watch?v=XVms0SA-6Xs)

**ETL:** [**https://uk.wikipedia.org/wiki/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**](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**](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>