

# НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського» ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

**Кафедра системного програмування та спеціалізованих** комп'ютерних систем

#### Лабораторна робота № 3

## з дисципліни «Бази даних та засоби управління»

Тема: «Засоби оптимізації роботи СУБД PostgreSQL»

Виконав: студент III курсу

ФПМ групи КВ-84

Бендецький Є.О.

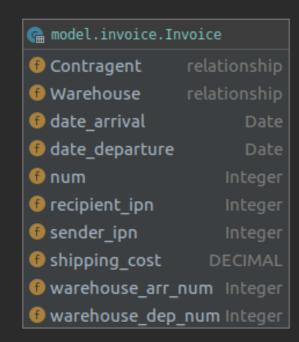
Перевірив: Петрашенко А.В.

Київ

#### Завдання

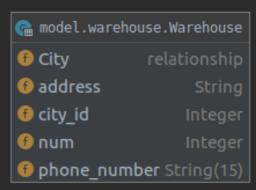
### Варіант 4

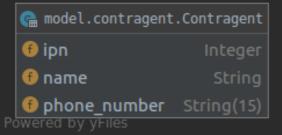
Відношення	Атрибут	Тип даних
Відношення " <b>Invoices</b> " Вміщує інформацію про транспортні накладні	num — унікальний номер накладної date_departure — дата оформлення та відправлення посилки date_arrival — дата прибуття посилки shipping_cost — вартість доставки sender_ipn — ІПН відправника recipient_ipn — ІПН отримувача warehouse_dep_num — номер складу, з якого відправлено посилку warehouse_arr_num — номер складу до якого прямує посилка	Числовий Дата Дата Грошовий Числовий Числовий Числовий Числовий
Відношення "Goods" Вміщує інформацію про товари та вантажі, що перевозяться	id — унікальний ідентифікатор товару height — висота посилки width — ширина посилки depth — глибина посилки weight — вага посилки description — опис вантажу invoice_num — номер накладної, до якої належить ця посилка	Числовий Числовий Числовий Числовий Числовий Числовий Текстовий Числовий
Відношення "Contragents" Вміщує інформацію про осіб, які є відправниками або отримувачами	<i>ipn</i> — ідентифікаційний податковий номер особи (ІПН) <i>name</i> — ПІБ особи <i>phone_number</i> — мобільний номер телефона особи	Числовий Текстовий Текстовий
Відношення "Warehouses" Вміщує інформацію про склади, між якими транспортуються вантажі	num — унікальний номер складу address — адреса, за якою знаходиться склад phone_number — номер телефону складу	Числовий Текстовий Текстовий
Відношення "Cities" Вміщує інформацію про міста, в яких знаходяться відділення	id — унікальний номер міста name — наза міста	Числовий Текстовий
Відношення "Reweightings" Вміщує інформацію про неспівпадіння реальної ваги з наданою після перезважування	<ul> <li>id — унікальний номер події</li> <li>date_inspection — дата</li> <li>перезважування</li> <li>weight_before — заявлена вага</li> <li>weight_after — реальна вага</li> <li>parcel_id — унікальний номер</li> <li>посилки, яку перезважували</li> </ul>	Числовий Дата Числовий Числовий Числовий



<pre>← model.goods.Goods</pre>		
f Invoice related	tionship	
<b>⊕</b> depth	Integer	
description	Text	
f height	Integer	
<b>⊕</b> id	Integer	
f invoice_num	Integer	
⊕ weight	Integer	
f width	Integer	

a model.reweightings.Reweightings		
<b>6</b> Goods	relationship	
date_inspection	DateTime	
<b></b> id	Integer	
parcel_id	Integer	
weight_after	Integer	
weight_before	Integer	







```
model/__init__.py
class Model:
    def __init__(self, session: Session, cls: Type):
        self.\__session = session
        self._cls = cls
    def create(self, item: object):
        self.\__session.add(item)
        self.__commit_or_rollback_on_failure()
        return item
    def create many(self, items: [object]):
        self.__session.add all(items)
        self.__commit_or_rollback_on_failure()
        return items
    def read(self, pk: int):
        item = self. session.query(self. cls).get(pk)
        if item is None:
            raise Exception(f"No item with such primary key {pk} was found")
        return item
    def read all(self, offset: int = 0, limit: int = None):
        pk name = self.get primary key name()
        items =
self. session.query(self. cls).order by(asc(pk name)).offset(offset).limit(1
imit).all()
        return items
    def count all(self):
        return self. session.query(self. cls).count()
    def update(self, item: dict):
        pk name = self.get primary key name()
        pk = item[pk name]
        try:
            self. session.query(self. cls).filter by(**{pk name:
pk}).update(item)
            self. session.commit()
        except Exception:
           self. session.rollback()
            raise
        return self.read(pk)
    def delete(self, pk: int):
        pk name = self.get primary key name()
        try:
            self. session.query(self. cls).filter by(**{pk name:
pk}).delete()
            self. session.commit()
        except Exception:
            self.__session.rollback()
            raise
model/city.py
class City(Base):
    tablename = 'cities'
    id = Column(Integer, primary_key=True)
    name = Column(String, nullable=False)
    def str (self):
        return f"City [id={self.id}, name={self.name}]"
```

```
model/contragent.pv
class Contragent(Base):
    tablename = 'contragents'
    ipn = Column(Integer, primary_key=True, autoincrement=False)
    name = Column(String, nullable=False)
    phone number = Column(String(15), nullable=False)
        __str__(self):
        return f"Contragent [ipn={self.ipn}, name={self.name},
phone number={self.phone number}]"
    @staticmethod
    def get distinct names(session: Session):
        return [name for (name,) in session.query(Contragent.name).all()]
model/goods.py
class Goods (Base):
    tablename = 'goods'
    id = Column(Integer, primary key=True)
   height = Column(Integer, nullable=False)
    width = Column(Integer, nullable=False)
    depth = Column(Integer, nullable=False)
    weight = Column(Integer, nullable=False)
    description = Column(Text)
    invoice num = Column(Integer, ForeignKey('invoices.num',
onupdate='restrict', ondelete='restrict'), nullable=False)
    invoice = relationship("Invoice", backref="goods")
    table args = (
        Index(
            'goods descriptions index',
            func.to tsvector('english', description),
            postgresql using='gin'
        ),
        Index(
            'volume index',
            height, width, depth,
            postgresql using='brin'
        ),
        Index(
            'invoice num index',
            invoice num,
            postgresql using='brin'
        ),
    )
    def str (self):
        return f"Goods [id={self.id}, height={self.height},
width={self.width}, depth={self.depth}, " \
               f"weight={self.weight}, description={self.description},
invoice num={self.invoice num}]"
model/invoice.py
class Invoice (Base):
    tablename = 'invoices'
    num = Column(Integer, primary key=True)
    date departure = Column(Date, nullable=False)
```

date arrival = Column(Date)

```
shipping cost = Column(DECIMAL, nullable=False)
    sender ipn = \
        Column(Integer, ForeignKey('contragents.ipn', onupdate='restrict',
ondelete='restrict'), nullable=False)
    recipient ipn = \
        Column(Integer, ForeignKey('contragents.ipn', onupdate='restrict',
ondelete='restrict'), nullable=False)
    warehouse dep num = \
        Column (Integer, ForeignKey ('warehouses.num', onupdate='restrict',
ondelete='restrict'), nullable=False)
    warehouse arr num = \
        Column(Integer, ForeignKey('warehouses.num', onupdate='restrict',
ondelete='restrict'), nullable=False)
    sender = relationship("Contragent", backref="invoices outbox",
foreign keys=[sender ipn])
    recipient = relationship("Contragent", backref="invoices inbox",
foreign keys=[recipient ipn])
    warehouse arrival = relationship("Warehouse",
backref="invoices arriving", foreign keys=[warehouse arr num])
    warehouse departure = relationship("Warehouse",
backref="invoices departing", foreign keys=[warehouse dep num])
    table args = (
        Index(
            'sender ipn index',
            sender ipn,
            postgresql using='gin'
        ),
        Index(
            'shipping cost index',
            shipping cost,
            postgresql using='brin'
        ),
    def str (self):
        return f"Invoice [num={self.num},
date departure={self.date departure}, date arrival={self.date arrival}, " \
               f"shipping cost={self.shipping cost},
sender_ipn={self.sender_ipn}, " \
               f"recipient ipn={self.recipient ipn},
warehouse_dep_num={self.warehouse_dep num}, " \
               f"warehouse arr num={self.warehouse arr num}]"
    @staticmethod
    def get extremum shipping cost(session: Session):
        min query = func.min(Invoice.shipping cost)
        max query = func.max(Invoice.shipping cost)
        return session.query(min query).scalar(),
session.query(max query).scalar()
model/reweightings.py
class Reweightings(Base):
    tablename = 'reweightings'
    id = Column(Integer, primary key=True)
    weight before = Column(Integer, nullable=False)
    weight after = Column(Integer, nullable=False)
    date inspection = Column(DateTime, nullable=False)
    parcel id = Column(Integer, ForeignKey('goods.id', onupdate='restrict',
ondelete='restrict'), nullable=False)
```

```
parcel = relationship("Goods", backref="reweightings")
    _{\rm table\_args\_} = (
        Index(
            'date inspection index',
            date inspection,
            postgresql using='brin'
        ),
    )
    def str (self):
        return f"Reweightings [id={self.id},
weight_before={self.weight_before}, weight_after={self.weight after}, " \
               f"date inspection={self.date inspection},
parcel id={self.parcel id}]"
model/warehouse.py
class Warehouse (Base):
    __tablename__ = 'warehouses'
   num = Column(Integer, primary_key=True)
    address = Column(String, nullable=False)
   phone number = Column(String(15), nullable=False)
    city id = Column(Integer, ForeignKey('cities.id', ondelete='restrict',
onupdate='restrict'), nullable=False)
    city = relationship("City", backref="warehouses")
    def str (self):
        return f"Warehouse [num={self.num}, address={self.address}, " \
               f"phone_number={self.phone_number}, city_id={self.city id}]"
Створення індексів
CREATE INDEX IF NOT EXISTS volume index ON goods USING brin (height, width,
depth);
CREATE INDEX IF NOT EXISTS invoice num index ON goods USING gin
(invoice num);
CREATE INDEX IF NOT EXISTS sender ipn index ON invoices USING gin
(sender ipn);
CREATE INDEX IF NOT EXISTS shipping cost index ON invoices USING brin
(shipping cost);
CREATE INDEX IF NOT EXISTS date inspection index ON reweightings USING brin
(date inspection);
SQL-запити
SELECT * FROM goods WHERE invoice num = 170;
SELECT id from reweightings WHERE date inspection BETWEEN '2020-12-20
13:04:07' AND '2020-12-20 13:04:20'
```

SELECT DISTINCT sender ipn FROM (SELECT sender ipn FROM goods INNER JOIN

SELECT min(shipping cost) FROM invoices;

invoices on invoice num = num) t;

#### Тригер

```
CREATE TABLE IF NOT EXISTS prohibited items dict (
    id serial PRIMARY KEY,
    word varchar(255) NOT NULL
);
TRUNCATE prohibited items dict RESTART IDENTITY;
INSERT INTO prohibited items dict (word) VALUES ('aerosol'), ('gun'),
('batteries');
CREATE OR REPLACE function goods trigger() returns trigger as $$
    declare
        description with time text;
        current word prohibited items dict.word%TYPE;
    begin
        if old is not null and old.weight != new.weight then
            INSERT INTO reweightings (weight before, weight after,
date inspection, parcel id)
                VALUES (old.weight, new.weight, current timestamp, new.id);
        elseif old is null then
            description_with_time = coalesce(new.description, '') || ' added
to db at: ' || current timestamp::char(19);
            UPDATE goods SET description = description with time WHERE id =
new.id;
        end if;
        for current word in SELECT word FROM prohibited items dict loop
            if new.description ilike '%' || current word || '%' then
                raise 'This item is prohibited';
        end loop;
        return new;
        exception
            when no data found then
            when too many rows then
                raise 'Could not check description, try again later';
    end;
$$ language plpgsql;
DROP trigger IF EXISTS goods helper on goods;
CREATE trigger goods helper AFTER DELETE OR INSERT on goods for each row
EXECUTE procedure goods trigger();
```