



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 KB-84

Бендецький Є.О.

Перевірів: Петрашенко А.В.

Київ

2020

Завдання

Варіант 4

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

model.invoice.Invoice	
f Contragent	relationship
f Warehouse	relationship
f date_arrival	Date
f date_departure	Date
f num	Integer
f recipient_ipn	Integer
f sender_ipn	Integer
f shipping_cost	DECIMAL
f warehouse_arr_num	Integer
f warehouse_dep_num	Integer

model.goods.Goods	
f Invoice	relationship
f depth	Integer
f description	Text
f height	Integer
f id	Integer
f invoice_num	Integer
f weight	Integer
f width	Integer

model.reweightings.Reweightings	
f Goods	relationship
f date_inspection	DateTime
f id	Integer
f parcel_id	Integer
f weight_after	Integer
f weight_before	Integer

model.warehouse.Warehouse	
f City	relationship
f address	String
f city_id	Integer
f num	Integer
f phone_number	String(15)

model.contragent.Contragent	
f ipn	Integer
f name	String
f phone_number	String(15)

model.city.City	
f id	Integer
f name	String

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(l
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.py

```
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)

    def __str__(self):
        return f"Contragent [ipn={self.ipn}, name={self.name},\n\
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},\n\
width={self.width}, depth={self.depth}, " \
            f"weight={self.weight}, description={self.description},\n\
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")

__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 min(shipping_cost) FROM invoices;
SELECT DISTINCT sender_ipn FROM (SELECT sender_ipn FROM goods INNER JOIN
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 $$  
    declare  
        description_with_time text;  
        current_word prohibited_items_dict.word%TYPE;  
    begin  
        if old is not null and old.weight != new.weight then  
            INSERT INTO reweightings (weight_before, weight_after,  
date_inspection, parcel_id)  
                VALUES (old.weight, new.weight, current_timestamp, new.id);  
        elseif old is null then  
            description_with_time = coalesce(new.description, '') || ' added            UPDATE goods SET description = description_with_time WHERE id =  
new.id;  
        end if;  
        for current_word in SELECT word FROM prohibited_items_dict loop  
            if new.description ilike '%' || current_word || '%' then  
                raise 'This item is prohibited';  
            end if;  
        end loop;  
        return new;  
    exception  
        when no_data_found then  
        when too_many_rows then  
            raise 'Could not check description, try again later';  
    end;  
$$ 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();
```