ГУАП

КАФЕДРА № 41

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

| Старший преподаватель | | Б.К. Акопян |
|---|---|---|
| должность, уч. степень, звание | подпись, дата | инициалы, фамилия |

# ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №6

# СВЯЗЬ СУБД POSTGRESQL И PYTHON

по курсу: БАЗЫ ДАННЫХ

РАБОТУ ВЫПОЛНИЛ

| СТУДЕНТ ГР. № 4217 | | Д.М. Никитин |
|---|---|---|
| | подпись, дата | инициалы, фамилия |

Санкт-Петербург 2025

**Цель работы**

Произвести связь базы данных в PostgreSQL и Python, изучить операции по манипулированию с данными БД, а также созданию простейших пользовательских функций.

**Решение**

Сначала была осуществлена связь PostgreSQL и Python, для большего погружения в лабораторную работу была дополнительно выбрана библиотека SQLAlchemy – одна из лучших ORM на SQL и Python. Далее была создана новая таблица под названием locations. См. рис. 1.

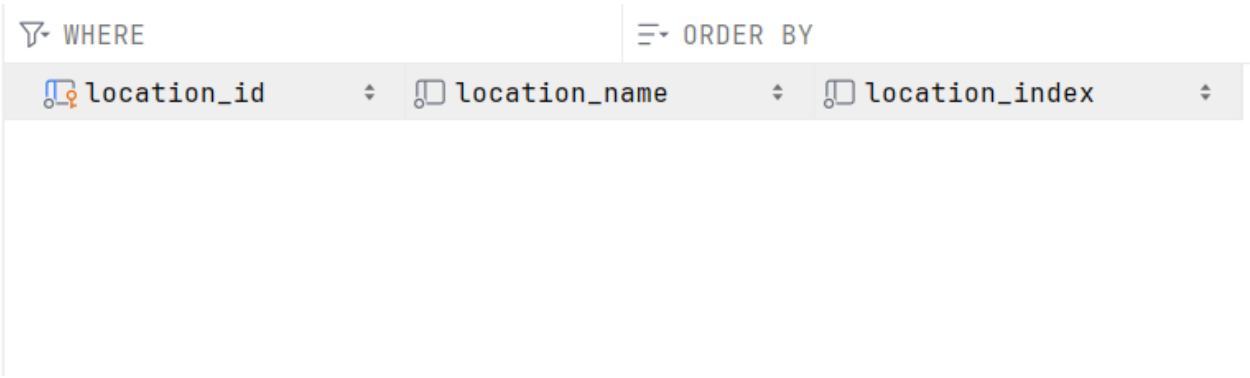


Рисунок 1 – Созданная таблица

Далее таблица была заполнена с помощью следующего набора команд. Код заполняет таблицу только в том случае, если она пуста. Результат на рисунке 2.

```
# Создание сессии для работы с базой данных для добавления
данных в таблицу locations
with Session() as session:
    if len(session.query(Location).all()) == 0:

        # SQL-запросы для вставки данных
        sql_queries = [
            "INSERT INTO locations VALUES (1, 'Roma',
'00989');",
            "INSERT INTO locations VALUES (2, 'Venice',
'10934');",
            "INSERT INTO locations VALUES (3, 'Tokyo',
'1689');",
            "INSERT INTO locations VALUES (4, 'Hiroshima',
'6823');",
            "INSERT INTO locations VALUES (5, 'Southlake',
'26192');",
            "INSERT INTO locations VALUES (6, 'South San
Francisco', '99236');",
```

```
            "INSERT INTO locations VALUES (7, 'South Brunswick',
'50090');",
            "INSERT INTO locations VALUES (8, 'Seattle',
'98199');",
            "INSERT INTO locations VALUES (9, 'Toronto', 'M5V
2L7');",
            "INSERT INTO locations VALUES (10, 'Whitehorse',
'YSW 9T2');"
        ]

        # Выполнение SQL-запросов по одному
        for query in sql_queries:
            session.execute(text(query))
        # Фиксация изменений в базе данных
        session.commit()
    else:
        pass
```

Pull Requests

ORDER BY

| | location_id | location_name | location_index |
|---|---|---|---|
| 1 | 1 | Roma | 00989 |
| 2 | 2 | Venice | 10934 |
| 3 | 3 | Tokyo | 1689 |
| 4 | 4 | Hiroshima | 6823 |
| 5 | 5 | Southlake | 26192 |
| 6 | 6 | South San Francisco | 99236 |
| 7 | 7 | South Brunswick | 50090 |
| 8 | 8 | Seattle | 98199 |
| 9 | 9 | Toronto | M5V 2L7 |
| 10 | 10 | Whitehorse | YSW 9T2 |

Рисунок 2 – Заполненная таблица

Далее была создана колонка с внешним ключом на таблицу locations. С помощью кода создаётся колонка, если она ещё не существует. См. рис. 3, 4.

```
try:
    with Session() as session:
        inspector = inspect(engine)
        columns = inspector.get_columns('employees',
schema='hr')
        columns = [column['name'] for column in columns]
        new_column_name = 'location_id'
        if new_column_name not in columns:
            print(f"Создание колонки {new_column_name}")
            session.execute(text(
                f"""
                ALTER TABLE hr.employees ADD COLUMN
```

```
{new_column_name} INTEGER REFERENCES
hr.locations({new_column_name});
                    """
                ))
                session.commit()
                print(f"Колонка {new_column_name} добавлена")
            else:
                print(f"Колонка уже {new_column_name} существует")
except Exception as e:
    print("ERROR", e)
```

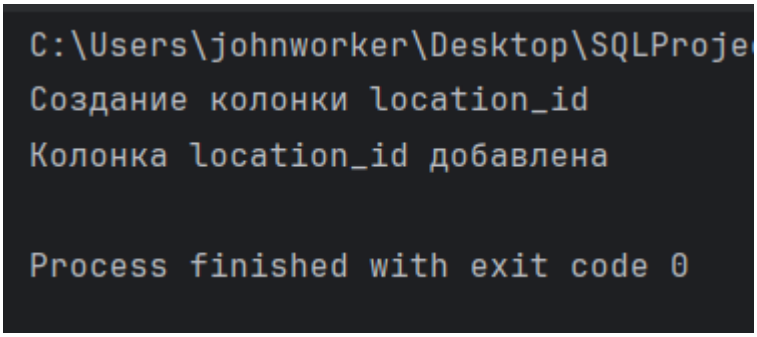


Рисунок 3 – Создание колонки

| | employee_id | first_name | last_name | job_id | salary | manager_id | department_id | location_id |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | Steven | King | AD_PRES | 24000 | <null> | 90 | <null> |
| 2 | 2 | Neena | Kochhar | AD_VP | 17000 | 1 | 90 | <null> |
| 3 | 3 | Lex | De Haan | AD_VP | 17000 | 1 | 90 | <null> |
| 4 | 4 | Alexander | Hunold | IT_PROG | 9000 | 3 | 60 | <null> |
| 5 | 5 | Bruce | Ernst | IT_PROG | 6000 | 4 | 60 | <null> |
| 6 | 6 | David | Austin | IT_PROG | 4800 | 4 | 60 | <null> |
| 7 | 7 | Valli | Pataballa | IT_PROG | 4800 | 4 | 60 | <null> |
| 8 | 8 | Diana | Lorentz | IT_PROG | 4200 | 4 | 60 | <null> |
| 9 | 9 | Nancy | Greenberg | FI_MGR | 12008 | 2 | 100 | <null> |
| 10 | 10 | Daniel | Faviet | FI_ACCOUNT | 9000 | 9 | 100 | <null> |
| 11 | 11 | John | Chen | FI_ACCOUNT | 8200 | 9 | 100 | <null> |
| 12 | 12 | Ismael | Sciarra | FI_ACCOU | 7700 | 9 | 100 | <null> |
| 13 | 13 | Jose Manuel | Urman | FI_ACCOUNT | 7800 | 9 | 100 | <null> |

53 rows

Рисунок 4 – Результат изменения

Далее колонка была заполнена случайными локациями с равномерным распределением. С помощью следующего кода. Результат выполнения см. рис. 5, 6.

```
# Заполнение location_id только тех строк, где значение null
with Session() as session:
    location_ids = [x[0] for x in
session.query(Location.location_id).all()]
    employee_ids = [x[0] for x in
session.query(Employee.employee_id).where(Employee.location_id.i
s_(None))]
    if len(location_ids) == 0:
        print("Локации отсутствуют")
    elif len(employee_ids) == 0:
        print("Работники отсутствуют")
    else:
        for employee_id in employee_ids:
```

```
            session.query(Employee).where(Employee.employee_id
== employee_id).update({Employee.location_id:
choice(location_ids)}, synchronize_session='evaluate')
        session.commit()
        print("Все ID вставлены")
```

| job_id | salary | manager_id | department_id | location_id |
|---|---|---|---|---|
| AD_VP | 17000 | 1 | 90 | 8 |
| IT_PROG | 4800 | 4 | 60 | 1 |
| IT_PROG | 4200 | 4 | 60 | 9 |
| FI_MGR | 12008 | 2 | 100 | 2 |
| FI_ACCOUNT | 9000 | 9 | 100 | 10 |
| FI_ACCOUNT | 8200 | 9 | 100 | 10 |
| FI_ACCOUNT | 7700 | 9 | 100 | 10 |
| FI_ACCOUNT | 7800 | 9 | 100 | 9 |
| FI_ACCOUNT | 6900 | 9 | 100 | 2 |
| PU_MAN | 11000 | 1 | 30 | 10 |
| PU_CLERK | 3100 | 15 | 30 | 1 |
| PU_CLERK | 2900 | | 30 | 7 |

53 rows

Рисунок 5 – Таблица employees с заполненным столбцом location_id

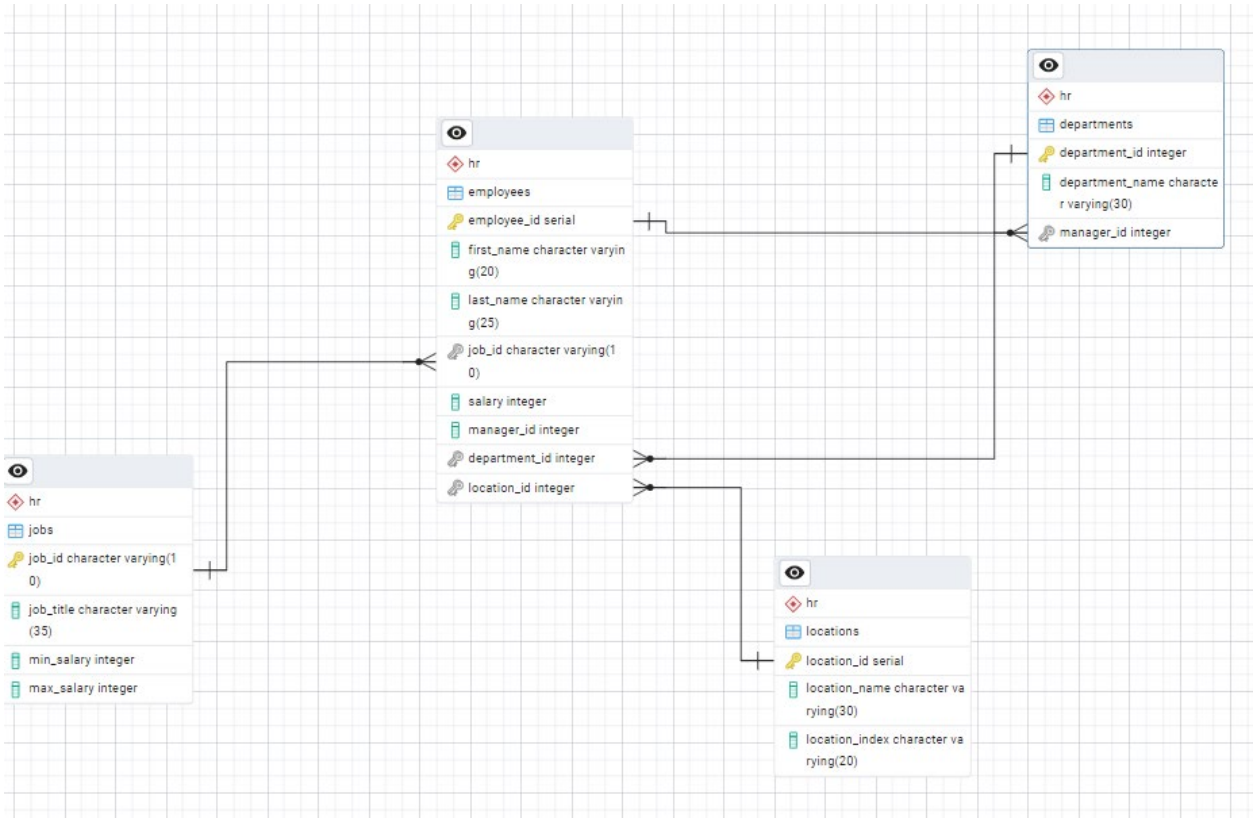


Рисунок 6 – Полученная схема данных

Далее с помощью следующего кода была создана функция. См. рис. 7.

```
CREATE OR REPLACE FUNCTION select_data(id_dept INTEGER) RETURNS
SETOF departments AS $$
```

```
    SELECT * FROM departments WHERE departments.department_id >
id_dept ORDER BY departments.department_id;

$$ LANGUAGE sql;
```

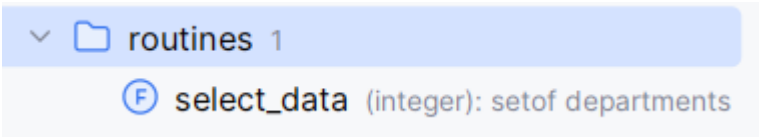


Рисунок 7 – Созданная хранимая функция

Далее функция была вызвана с помощью кода на Python. См. рис. 8.

| ID | Одел | Manager ID |
|---|---|---|
| 40 | Human Resources | 53 |
| 50 | Shipping | 22 |
| 60 | IT | 4 |
| 80 | Sales | 46 |
| 90 | Executive | 1 |
| 100 | Finance | 9 |

Рисунок 8 – Результат вызова функции

Далее с помощью следующего кода была написана функция и положена в файл, при этом не создана в базе данных.

```
-- ФИО и должность для сотрудника, который получает зарплату
ниже среднего
CREATE OR REPLACE FUNCTION get_low_price_employees()
RETURNS TABLE(first_name VARCHAR, last_name VARCHAR, job_title
VARCHAR) AS $$
    SELECT e.first_name, e.last_name, j.job_title
    FROM hr.employees e
    JOIN hr.jobs j ON e.job_id = j.job_id
    WHERE e.salary < (SELECT AVG(salary) FROM hr.employees)
$$ LANGUAGE sql;
```

Далее данная функция была создана с помощью Python. Вывод см. рис. 9.

```
-- ФИО и должность для сотрудника, который получает зарплату
ниже среднего
CREATE OR REPLACE FUNCTION hr.get_low_price_employees()
RETURNS TABLE(first_name VARCHAR, last_name VARCHAR, job_title
VARCHAR) AS $$
    SELECT e.first_name, e.last_name, j.job_title
    FROM hr.employees e
    JOIN hr.jobs j ON e.job_id = j.job_id
```

```
    WHERE e.salary < (SELECT AVG(salary) FROM hr.employees)
$$ LANGUAGE sql;
```

```
Фукнция успешно создана
('Valli', 'Pataballa', 'Programmer')
('Diana', 'Lorentz', 'Programmer')
('Alexander', 'Khoo', 'Purchasing Clerk')
('Shelli', 'Baida', 'Purchasing Clerk')
('Sigal', 'Tobias', 'Purchasing Clerk')
('Guy', 'Himuro', 'Purchasing Clerk')
('Karen', 'Colmenares', 'Purchasing Clerk')
('Kevin', 'Mourgos', 'Stock Manager')
('Julia', 'Nayer', 'Stock Clerk')
('Irene', 'Mikkilineni', 'Stock Clerk')
('James', 'Landry', 'Stock Clerk')
('Steven', 'Markle', 'Stock Clerk')
('Laura', 'Bissot', 'Stock Clerk')
('Mozhe', 'Atkinson', 'Stock Clerk')
('James', 'Marlow', 'Stock Clerk')
('TJ', 'Olson', 'Stock Clerk')
('Jason', 'Mallin', 'Stock Clerk')
('Michael', 'Rogers', 'Stock Clerk')
('Ki', 'Gee', 'Stock Clerk')
('Hazel', 'Philtanker', 'Stock Clerk')
('Renske', 'Ladwig', 'Stock Clerk')
('Stephen', 'Stiles', 'Stock Clerk')
('John', 'Seo', 'Stock Clerk')
('Joshua', 'Patel', 'Stock Clerk')
('Trenna', 'Rajs', 'Stock Clerk')
```

Рисунок 9 – Вывод созданной хранимой функции

Также были выполнены три запроса. Два из них их пособия, один придуман самостоятельно. Код запросов ниже, результат выполнения на рисунках 10-12.

```
# Найти средние зарплаты по каждому из отделов(можно ли как - то округлить полученные значения?).
with Session() as session:
    temp = (
```

```
        session
        .query(
            Employee.department_id,
            func.round(func.avg(Employee.salary), 2)
        )
        .group_by(Employee.department_id)
        .order_by(func.round(Employee.department_id))
    )

    print(f"{'Отдел':<10} | {'Средняя зарплата':>15}")
    print("-" * 28)
    for dept, avg_salary in temp:
        print(f"{str(dept):<10} | {avg_salary:>15.2f}")

# Вывести названия должностей и количество сотрудников,
соответствующих им. Отсортировать данные по убыванию числа
сотрудников.
with Session() as session:
    temp = (
        session.query(Job.job_title,
func.count(Employee.employee_id))
        .join(Employee, Employee.job_id == Job.job_id)
        .group_by(Job.job_title)
        .order_by(func.count(Employee.employee_id).desc())
        .all()
    )
    print(f"{'Должность':<35} | {'Сотрудников':>11}")
    print("-" * 50)
    for job_title, count in temp:
        print(f"{job_title:<35} | {count:>11}")

# Количество сотрудников по городам
with Session() as session:
    result = (
        session.query(Location.location_name,
func.count(Employee.employee_id).label("emp_count"))
        .join(Employee, Employee.location_id ==
Location.location_id)
        .group_by(Location.location_name)
        .order_by(func.count(Employee.employee_id).desc())
        .all()
    )

    print(f"{'Местоположение':<30} | {'Сотрудников':>11}")
    print("-" * 45)
    for location_name, count in result:
        print(f"{location_name:<30} | {count:>11}")
```

| Отдел | Средняя зарплата |
|---|---|
| 10 | 4400.00 |
| 20 | 13000.00 |
| 30 | 4150.00 |
| 40 | 6500.00 |
| 50 | 3684.00 |
| 60 | 5760.00 |
| 80 | 12200.00 |
| 90 | 19333.33 |
| 100 | 8601.33 |

Рисунок 10 – Результат первого запроса

| Должность | Сотрудников |
|---|---|
| Stock Clerk | 20 |
| Purchasing Clerk | 5 |
| Sales Manager | 5 |
| Stock Manager | 5 |
| Accountant | 5 |
| Programmer | 5 |
| Administration Vice President | 2 |
| Finance Manager | 1 |
| President | 1 |
| Human Resources Representative | 1 |
| Administration Assistant | 1 |
| Marketing Manager | 1 |
| Purchasing Manager | 1 |

Рисунок 11 – Результат второго запроса

| Местоположение | Сотрудников |
|---|---|
| Venice | 9 |
| Whitehorse | 9 |
| Roma | 7 |
| Hiroshima | 6 |
| Toronto | 5 |
| Southlake | 4 |
| South Brunswick | 4 |
| Tokyo | 3 |
| South San Francisco | 3 |
| Seattle | 3 |

Рисунок 12 – Результат третьего запроса

**Листинг кода**

См. приложения.

**Вывод**

В ходе работы была успешно реализована связь между PostgreSQL и Python с использованием библиотеки SQLAlchemy, что позволило не только управлять таблицами и записями в базе данных, но и создавать новые элементы схемы — такие как таблицы, колонки и внешние ключи. Также были созданы пользовательские SQL-функции: одна — напрямую через PostgreSQL, другая — через выполнение SQL-скрипта из Python. Для закрепления навыков были выполнены аналитические запросы с группировками и сортировками, позволяющие получить агрегированные данные по зарплатам, должностям и распределению сотрудников по городам. Работа продемонстрировала как базовые возможности ORM, так и гибкость SQL в связке с Python.

# Приложение А

# «Код программы на Python»

```
import sqlalchemy
from sqlalchemy import create_engine, text, inspect
from sqlalchemy import String, Integer, Column, ForeignKey, func
from sqlalchemy.orm import sessionmaker, declarative_base
from dotenv import load_dotenv
from os import getenv
from random import choice

Base = declarative_base()


class Employee(Base):
    __tablename__ = 'employees'
    __table_args__ = {'schema': 'hr'}
    employee_id = Column(Integer, primary_key=True,
nullable=False)
    first_name = Column(String(20), nullable=False)
    last_name = Column(String(25), nullable=False)
    job_id = Column(String(10), ForeignKey("jobs.job_id"),
nullable=False)
    salary = Column(Integer, nullable=True)
    manager_id = Column(Integer)
    department_id = Column(Integer,
ForeignKey("departments.department_id"), nullable=False)
    location_id = Column(Integer,
ForeignKey("locations.location_id"), nullable=False)

    def __repr__(self):
        return f"employee_id: {self.employee_id}, first_name:
{self.first_name}, last_name: {self.last_name}, job_id:
{self.job_id}, salary: {self.salary}, manager_id:
{self.manager_id}, department_id: {self.department_id}"


class Department(Base):
    __tablename__ = 'departments'
    __table_args__ = {'schema': 'hr'}
    department_id = Column(Integer, primary_key=True,
nullable=False)
    department_name = Column(String(30), nullable=False)
    manager_id = Column(Integer,
ForeignKey("employees.employee_id"))


class Job(Base):
    __tablename__ = 'jobs'
    __table_args__ = {'schema': 'hr'}
    job_id = Column(String(10), primary_key=True,
nullable=False)
```

```
    job_title = Column(String(35), nullable=False)
    min_salary = Column(Integer)
    max_salary = Column(Integer)


class Location(Base):
    __tablename__ = 'locations'
    __table_args__ = {'schema': 'hr'}

    location_id = Column(Integer, primary_key=True,
nullable=False)
    location_name = Column(String(30), nullable=False)
    location_index = Column(String(20), nullable=False)


if __name__ == '__main__':
    load_dotenv()  # Загрузка секретных переменных
    # Секретное формирование ссылки для подключения к БД
    postgres_link =
f"{getenv("DBTYPE")}+{getenv("PSQLDRIVER")}://{getenv("DBUSERNAM
E")}:{getenv("PASSWORD")}@{getenv("HOST")}:{getenv("PORT")}/{get
env("DATABASE")}"
    engine = create_engine(postgres_link)  # Создание движка для
работы с БД
    Base.metadata.create_all(engine)  # Создание всех таблиц
    Session = sessionmaker(bind=engine)  # Фабрика сессий

    # Создание сессии для работы с базой данных для добавления
данных в таблицу locations
    with Session() as session:
        if
session.query(func.count(Location.location_id)).scalar() == 0:

            # SQL-запросы для вставки данных
            sql_queries = [
                "INSERT INTO locations VALUES (1, 'Roma',
'00989');",
                "INSERT INTO locations VALUES (2, 'Venice',
'10934');",
                "INSERT INTO locations VALUES (3, 'Tokyo',
'1689');",
                "INSERT INTO locations VALUES (4, 'Hiroshima',
'6823');",
                "INSERT INTO locations VALUES (5, 'Southlake',
'26192');",
                "INSERT INTO locations VALUES (6, 'South San
Francisco', '99236');",
                "INSERT INTO locations VALUES (7, 'South
Brunswick', '50090');",
                "INSERT INTO locations VALUES (8, 'Seattle',
'98199');",
                "INSERT INTO locations VALUES (9, 'Toronto',
'M5V 2L7');",
```

```
                "INSERT INTO locations VALUES (10, 'Whitehorse',
'YSW 9T2');"
            ]

            # Выполнение SQL-запросов по одному
            for query in sql_queries:
                session.execute(text(query))
            # Фиксация изменений в базе данных
            session.commit()
        else:
            pass

    # Создание колонки в том случае, если её ещё нет
    try:
        with Session() as session:
            inspector = inspect(engine)
            columns = inspector.get_columns('employees',
schema='hr')
            columns = [column['name'] for column in columns]
            new_column_name = 'location_id'
            if new_column_name not in columns:
                print(f"Создание колонки {new_column_name}")
                session.execute(text(
                    f"""
                    ALTER TABLE hr.employees ADD COLUMN
{new_column_name} INTEGER REFERENCES
hr.locations({new_column_name});
                    """
                ))
                session.commit()
                print(f"Колонка {new_column_name} добавлена")
            else:
                print(f"Колонка уже {new_column_name}
существует")
    except Exception as e:
        print("ERROR", e)

    # Заполнение location_id только тех строк, где значение null
    with Session() as session:
        location_ids = [x[0] for x in
session.query(Location.location_id).all()]
        employee_ids = [x[0] for x in
session.query(Employee.employee_id).where(Employee.location_id.i
s_(None))]
        if len(location_ids) == 0:
            print("Локации отсутствуют")
        elif len(employee_ids) == 0:
            print("Работники отсутствуют")
        else:
            for employee_id in employee_ids:

session.query(Employee).where(Employee.employee_id ==
employee_id).update({Employee.location_id:
```

```
choice(location_ids)}, synchronize_session='evaluate')
            session.commit()
            print("Все ID вставлены")

    # Найти средние зарплаты по каждому из отделов(можно ли как
- то округлить полученные значения?).
    with Session() as session:
        temp = (
            session
            .query(
                Employee.department_id,
                func.round(func.avg(Employee.salary), 2)
            )
            .group_by(Employee.department_id)
            .order_by(func.round(Employee.department_id))
        )

        print(f"{'Отдел':<10} | {'Средняя зарплата':>15}")
        print("-" * 28)
        for dept, avg_salary in temp:
            print(f"{str(dept):<10} | {avg_salary:>15.2f}")

    # Вывести названия должностей и количество сотрудников,
соответствующих им. Отсортировать данные по убыванию числа
сотрудников.
    with Session() as session:
        temp = (
            session.query(Job.job_title,
func.count(Employee.employee_id))
            .join(Employee, Employee.job_id == Job.job_id)
            .group_by(Job.job_title)
            .order_by(func.count(Employee.employee_id).desc())
            .all()
        )
        print(f"{'Должность':<35} | {'Сотрудников':>11}")
        print("-" * 50)
        for job_title, count in temp:
            print(f"{job_title:<35} | {count:>11}")

    # Количество сотрудников по городам
    with Session() as session:
        result = (
            session.query(Location.location_name,
func.count(Employee.employee_id).label("emp_count"))
            .join(Employee, Employee.location_id ==
Location.location_id)
            .group_by(Location.location_name)
            .order_by(func.count(Employee.employee_id).desc())
            .all()
        )

        print(f"{'Местоположение':<30} | {'Сотрудников':>11}")
        print("-" * 45)
```

```
        for location_name, count in result:
            print(f"{location_name:<30} | {count:>11}")

    # Вызов хранимой функции
    with Session() as session:
        result = session.execute(text("SELECT * FROM
select_data(:id_dept)"), {"id_dept": 30}).fetchall()
        print(f"{'ID':<3} | {'Одел':^20} | {'Manager ID':<3}")
        for row in result:
            print(f"{row[0]:<3} | {row[1]:^20} | {row[2]:<3}")

    # Создание функции с для получения ФИО и должности для
сотрудника, который получает зарплату ниже среднего
    try:
        with open('func_create_form_book.sql', 'r',
encoding='utf-8') as file:
            sql_script = text(file.read())
        with Session() as session:
            session.execute(sql_script)
            session.commit()
        print("Фукнция успешно создана")
    except Exception as error:
        print(error)

    with Session() as session:
        result = session.execute(text("SELECT * FROM
hr.get_low_price_employees();"))
        for row in result:
            print(row)
```

## Приложение Б
## «Код SQL для создания хранимой функции по идее из пособия»

```
CREATE OR REPLACE FUNCTION select_data(id_dept INTEGER) RETURNS
SETOF departments AS $$

    SELECT * FROM departments WHERE departments.department_id >
id_dept ORDER BY departments.department_id;

$$ LANGUAGE sql;
```

## Приложение В
## «Код SQL для создания хранимой функции по собственной идее»

```sql
-- ФИО и должность для сотрудника, который получает зарплату ниже среднего
CREATE OR REPLACE FUNCTION hr.get_low_price_employees()
RETURNS TABLE(first_name VARCHAR, last_name VARCHAR, job_title VARCHAR) AS $$
    SELECT e.first_name, e.last_name, j.job_title
    FROM hr.employees e
    JOIN hr.jobs j ON e.job_id = j.job_id
    WHERE e.salary < (SELECT AVG(salary) FROM hr.employees)
$$ LANGUAGE sql;
```

# Приложение Г
# «Полный вывод программы на Python»

```
C:\Users\johnworker\Desktop\SQLProjects\DataBases\P2Lab6\.venv\S
cripts\python.exe
C:\Users\johnworker\Desktop\SQLProjects\DataBases\P2Lab6\P2Lab6.
py
Колонка уже location_id существует
Работники отсутствуют
Отдел      | Средняя зарплата
----------------------------
10         |         4400.00
20         |        13000.00
30         |         4150.00
40         |         6500.00
50         |         3684.00
60         |         5760.00
80         |        12200.00
90         |        19333.33
100        |         8601.33
Должность                                 | Сотрудников
-------------------------------------------------------
Stock Clerk                               |          20
Purchasing Clerk                          |           5
Sales Manager                             |           5
Stock Manager                             |           5
Accountant                                |           5
Programmer                                |           5
Administration Vice President             |           2
Finance Manager                           |           1
President                                 |           1
Human Resources Representative            |           1
Administration Assistant                  |           1
Marketing Manager                         |           1
Purchasing Manager                        |           1
Местоположение                    | Сотрудников
-----------------------------------------------
Venice                            |           9
Whitehorse                        |           9
Roma                              |           7
Hiroshima                         |           6
Toronto                           |           5
Southlake                         |           4
South Brunswick                   |           4
Tokyo                             |           3
South San Francisco               |           3
Seattle                           |           3
ID  |          Одел          | Manager ID
40  |   Human Resources      | 53
50  |       Shipping         | 22
60  |          IT            | 4
80  |        Sales           | 46
90  |      Executive         | 1
```

```
100 |       Finance        | 9
Фукнция успешно создана
('Valli', 'Pataballa', 'Programmer')
('Diana', 'Lorentz', 'Programmer')
('Alexander', 'Khoo', 'Purchasing Clerk')
('Shelli', 'Baida', 'Purchasing Clerk')
('Sigal', 'Tobias', 'Purchasing Clerk')
('Guy', 'Himuro', 'Purchasing Clerk')
('Karen', 'Colmenares', 'Purchasing Clerk')
('Kevin', 'Mourgos', 'Stock Manager')
('Julia', 'Nayer', 'Stock Clerk')
('Irene', 'Mikkilineni', 'Stock Clerk')
('James', 'Landry', 'Stock Clerk')
('Steven', 'Markle', 'Stock Clerk')
('Laura', 'Bissot', 'Stock Clerk')
('Mozhe', 'Atkinson', 'Stock Clerk')
('James', 'Marlow', 'Stock Clerk')
('TJ', 'Olson', 'Stock Clerk')
('Jason', 'Mallin', 'Stock Clerk')
('Michael', 'Rogers', 'Stock Clerk')
('Ki', 'Gee', 'Stock Clerk')
('Hazel', 'Philtanker', 'Stock Clerk')
('Renske', 'Ladwig', 'Stock Clerk')
('Stephen', 'Stiles', 'Stock Clerk')
('John', 'Seo', 'Stock Clerk')
('Joshua', 'Patel', 'Stock Clerk')
('Trenna', 'Rajs', 'Stock Clerk')
('Curtis', 'Davies', 'Stock Clerk')
('Randall', 'Matos', 'Stock Clerk')
('Peter', 'Vargas', 'Stock Clerk')
('Jennifer', 'Whalen', 'Administration Assistant')
('David', 'Austin', 'Programmer')
('Bruce', 'Ernst', 'Programmer')

Process finished with exit code 0
```