

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 і засоби управління**

*на тему: “ Засоби оптимізації роботи СУБД PostgreSQL”*

Виконала:

студентка ІII курсу

групи КВ-02

Майстренко Ольга Олексіївна

Перевірив: Павловський Володимир Ілліч

Київ – 2022

*Метою роботи* є здобуття практичних навичок використання засобів оптимізації СУБД PostgreSQL.

*Завдання* роботи полягає у наступному:

1. Перетворити модуль “Модель” з шаблону MVC лабораторної роботи №2 у вигляд об’єктно-реляційної проекції (ORM).

2. Створити та проаналізувати різні типи індексів у PostgreSQL.

3. Розробити тригер бази даних PostgreSQL.

4. Навести приклади та проаналізувати рівні ізоляції транзакцій у PostgreSQL.

**Виконання роботи**

Логічна модель предметної області «Університет»

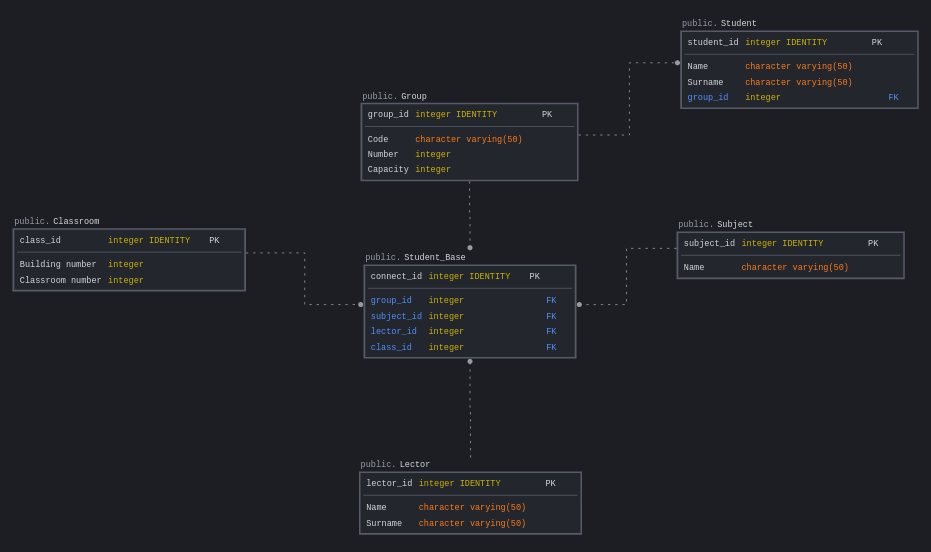


Рисунок 1. Схема бази даних (інструмент: sqldbm.com)

**Середовище та компоненти розробки**

Для розробки використовувалась мова програмування Python, середовище розробки PyCharm та стороння бібліотека SQLAlchemy.

**Структура програми та її опис**

Програма складається з трьох файлів-модулів: Controller, View та Model.

* Controller відповідає за обробку запитів користувача.
* Model за з’єднання та роботу з БД.
* View за оформлення та вивід інформації.

**Модулі моделі ORM**

main.py

**import** baseController **as** Controller

Controller.firstScreen()

baseModel.py

**import** sqlalchemy

**from** sqlalchemy **import** create\_engine

**from** sqlalchemy **import** MetaData, Table, Column, Integer, String, ForeignKey, DateTime, Boolean

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

**from** sqlalchemy.ext.declarative **import** declarative\_base

**from** sqlalchemy.orm **import** relationship, backref

**from** sqlalchemy.orm **import** sessionmaker

metadata = MetaData(schema="public")

Base = declarative\_base()

engine = create\_engine('postgresql://postgres:35171418@localhost/University', echo=True)

**class** Classroom(Base):

\_\_tablename\_\_ = 'Classroom'

class\_id = Column(Integer, primary\_key=True)

number = Column(Integer)

buildingnumber = Column(Integer)

**class** Group(Base):

\_\_tablename\_\_ = 'Group'

group\_id = Column(Integer, primary\_key=True)

code = Column(String(50))

number = Column(Integer)

capacity = Column(Integer)

**class** Lector(Base):

\_\_tablename\_\_ = 'Lector'

lector\_id = Column(Integer, primary\_key=True)

name = Column(String(50))

surname = Column(String(50))

**class** Subject(Base):

\_\_tablename\_\_ = 'Subject'

subject\_id = Column(Integer, primary\_key=True)

name = Column(String(50))

**class** Student(Base):

\_\_tablename\_\_ = 'Student'

student\_id = Column(Integer, primary\_key=True)

name = Column(String(50))

surname = Column(String(50))

group\_id = Column(Integer, ForeignKey('Group.group\_id'))

**class** Student\_Base(Base):

\_\_tablename\_\_ = 'Student\_Base'

conect\_id = Column(Integer, primary\_key=True)

class\_id = Column(Integer, ForeignKey('Classroom.class\_id'))

subject\_id = Column(Integer, ForeignKey('Subject.subject\_id'))

lector\_id = Column(Integer, ForeignKey('Lector.lector\_id'))

group\_id = Column(Integer, ForeignKey('Group.group\_id'))

makerSession = sessionmaker(bind=engine)

dbController.py

**import** baseModel **as** Model

**import** baseView **as** View

**def** performInsert(userChoice):

**print**("Enter values for " + userChoice + " and separate them with commas")

insval = View.getSplitInput()

conn = Model.makerSession()

**if** userChoice == "Classroom":

res = Model.Classroom(number=insval[0], buildingnumber=insval[1])

conn.add(res)

conn.commit()

**elif** userChoice == "Group":

res = Model.Group(code=insval[0], number=insval[1], capacity=insval[2])

conn.add(res)

conn.commit()

**elif** userChoice == "Lector":

res = Model.Lector(name=insval[0], surname=insval[1])

conn.add(res)

conn.commit()

**elif** userChoice == "Subject":

res = Model.Subject(name=insval[0])

conn.add(res)

conn.commit()

**elif** userChoice == "Student":

res = Model.Student(name=insval[0], surname=insval[1], group\_id=insval[2])

conn.add(res)

conn.commit()

**elif** userChoice == "Student\_Base":

res = Model.Student(class\_id=insval[0], subject\_id=insval[1], lector\_id=insval[2], group\_id=insval[3])

conn.add(res)

conn.commit()

conn.close()

**def** performDelete(userChoice):

**print**("Enter ID in " + userChoice + " table")

insval = View.getInput(None)

conn = Model.makerSession()

**if** userChoice == "Classroom":

res = conn.query(Model.Classroom).filter(Model.Classroom.class\_id == insval).first()

conn.delete(res)

conn.commit()

**elif** userChoice == "Group":

res = conn.query(Model.Group).filter(Model.Group.group\_id == insval).first()

conn.delete(res)

conn.commit()

**elif** userChoice == "Lector":

res = conn.query(Model.Lector).filter(Model.Lector.lector\_id == insval).first()

conn.delete(res)

conn.commit()

**elif** userChoice == "Subject":

res = conn.query(Model.Subject).filter(Model.Subject.subject\_id == insval).first()

conn.delete(res)

conn.commit()

**elif** userChoice == "Student":

res = conn.query(Model.Student).filter(Model.Student.student\_id == insval).first()

conn.delete(res)

conn.commit()

**elif** userChoice == "Student\_Base":

res = conn.query(Model.Student\_Base).filter(Model.Student\_Base.conect\_id == insval).first()

conn.delete(res)

conn.commit()

conn.close()

**def** performEdit(userChoice):

**print**("Enter ID in " + userChoice + " table")

obID = View.getInput(None)

**print**("Enter values for " + userChoice + " and separate them with commas")

insval = View.getSplitInput()

conn = Model.makerSession()

**if** userChoice == "Classroom":

res = conn.query(Model.Classroom).filter(Model.Classroom.class\_id == obID).first()

res.number = insval[0]

res.buildingnumber = insval[0]

conn.commit()

**elif** userChoice == "Group":

res = conn.query(Model.Group).filter(Model.Group.groupt\_id == obID).first()

res.code = insval[0]

res.number = insval[1]

res.capacity = insval[2]

conn.commit()

**elif** userChoice == "Lector":

res = conn.query(Model.Lector).filter(Model.Lector.lector\_id == obID).first()

res.name = insval[0]

res.surname = insval[1]

conn.commit()

**elif** userChoice == "Subject":

res = conn.query(Model.Subject).filter(Model.Subject.subject\_id == obID).first()

res.name = insval[0]

conn.commit()

**elif** userChoice == "Student":

res = conn.query(Model.Student).filter(Model.Student.student\_id == obID).first()

res.name = insval[0]

res.surname = insval[1]

res.group\_id = insval[2]

conn.commit()

**elif** userChoice == "Student\_Base":

res = conn.query(Model.Student\_Base).filter(Model.Student\_Base.conect\_id == obID).first()

res.class\_id = insval[0]

res.subject\_id = insval[1]

res.lector\_id = insval[2]

res.group\_id = insval[3]

conn.commit()

conn.close()

**def** firstScreen():

View.showMainMenu()

setMenuChoice = View.getInput(None)

setCHoiceTable = View.getInput(" Choose table to perform operation: ")

**if** setMenuChoice == '1':

performInsert(setCHoiceTable)

**elif** setMenuChoice == '2':

performDelete(setCHoiceTable)

**elif** setMenuChoice == '3':

performEdit(setCHoiceTable)

dbView.py

**def** showMainMenu():

**print**("**\n\n** KV-02 Maistrenko Olga. Lab Work 3")

**print**(" Maistrenko Olga")

**print**(" Lab Work 3")

**print**("**\n\n**")

**print**(" >>> PROGRAM MENU <<< ")

**print**(" 1. Insert data in table")

**print**(" 2. Remove data in table")

**print**(" 3. Edit data in table ")

**print**(" What do you want to do? ")

**def** getSplitInput():

inp = input().split(', ')

**return** inp

**def** getInput(string):

**if** string **is** None:

inp = input(" > ")

**else**:

inp = input(string)

**return** inp

**Головне меню програми**

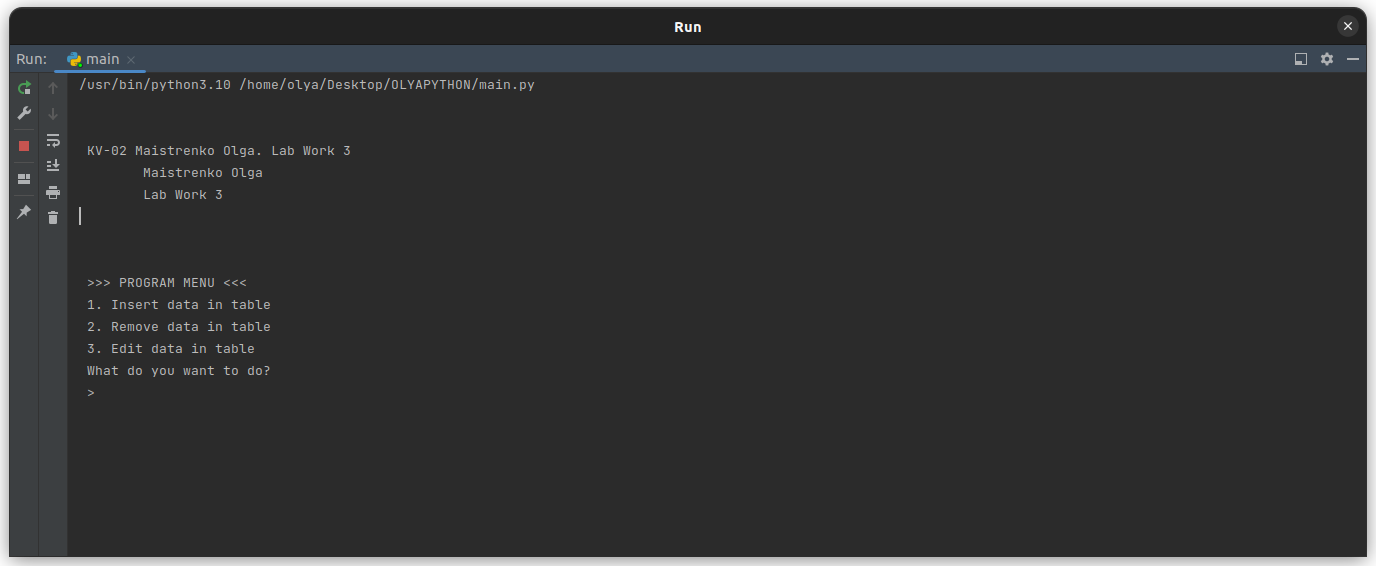


Рисунок 2. Головне меню програми

Меню для користувача складається із трьох пунктів:

* Перший пункт меню пропонує розпочати операцію внесення нових даних до таблиці, яку вибере користувач.
* Другий пункт меню пропонує приступити до операції видалення даних із таблиці, яку вибере користувач.
* Третій пункт пропонує редагувати дані у таблиці, яку вибере користувач.

**Результати роботи програми**

1. Insert

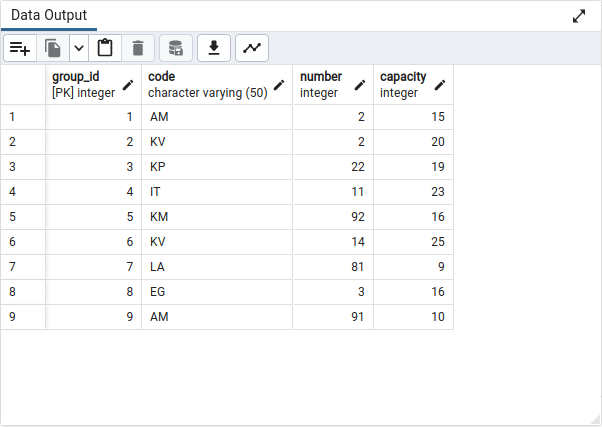


Рисунок 3. Стан таблиці Group

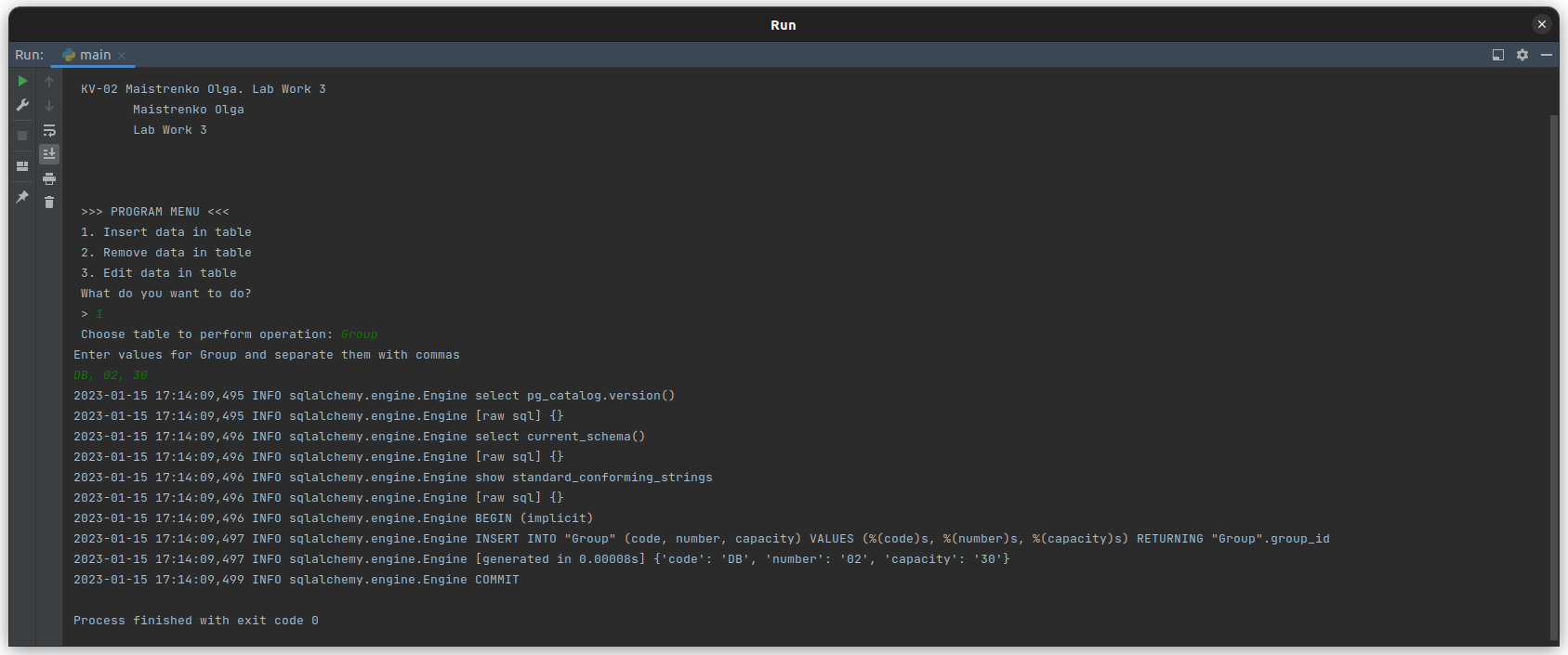


Рисунок 4. Внесення даних до обраної таблиці

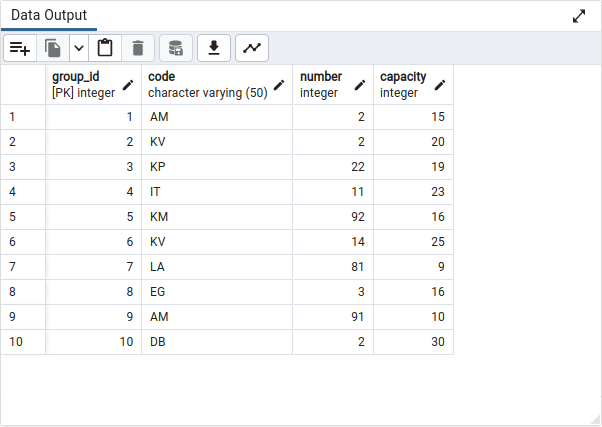


Рисунок 5. Стан таблиці Group

1. Delete

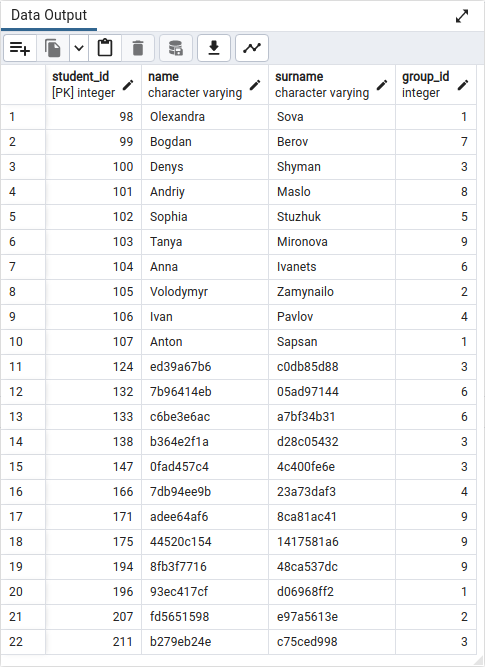


Рисунок 6. Стан таблиці Student

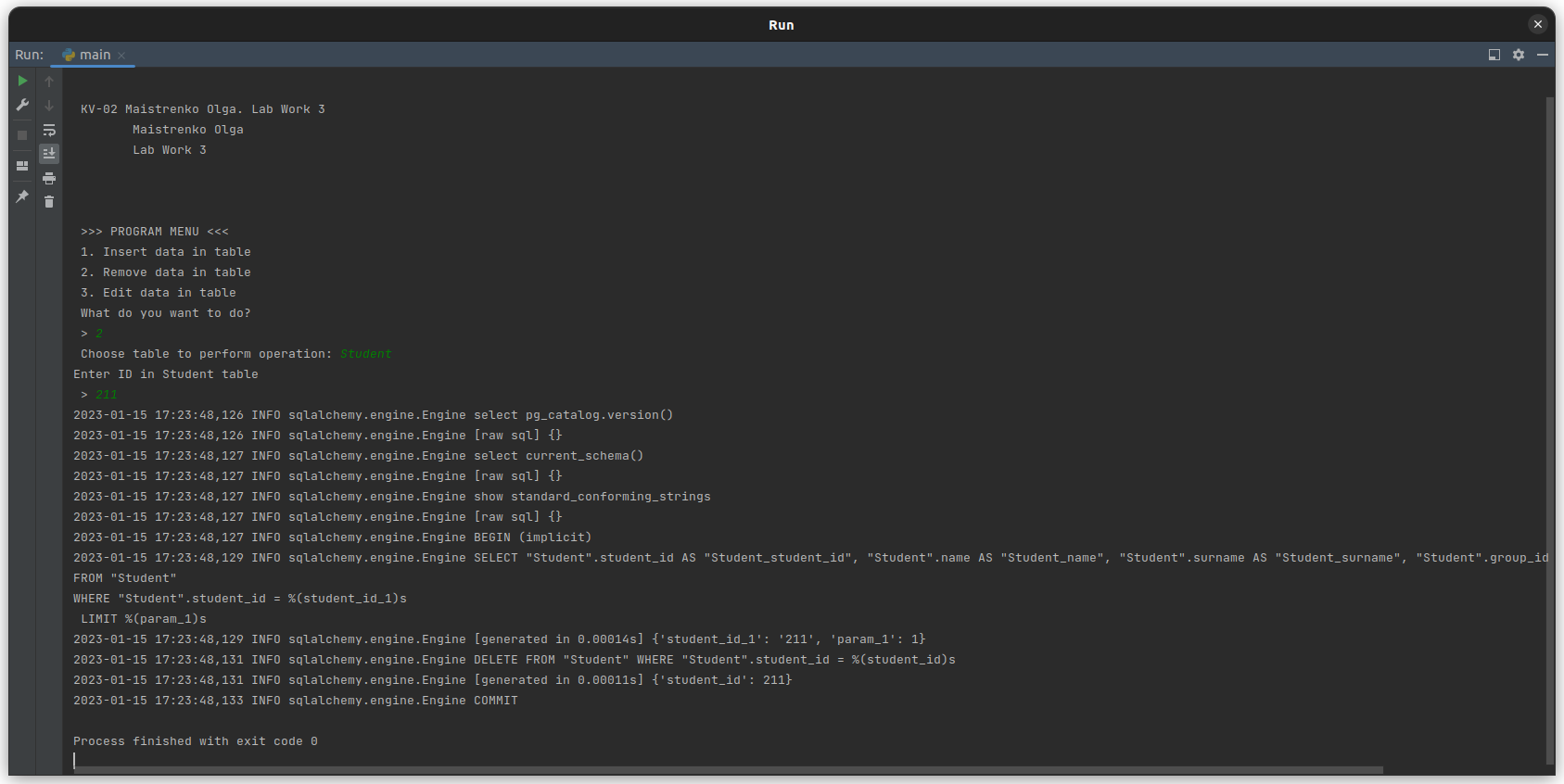


Рисунок 7. Видалення даних з обраної таблиці

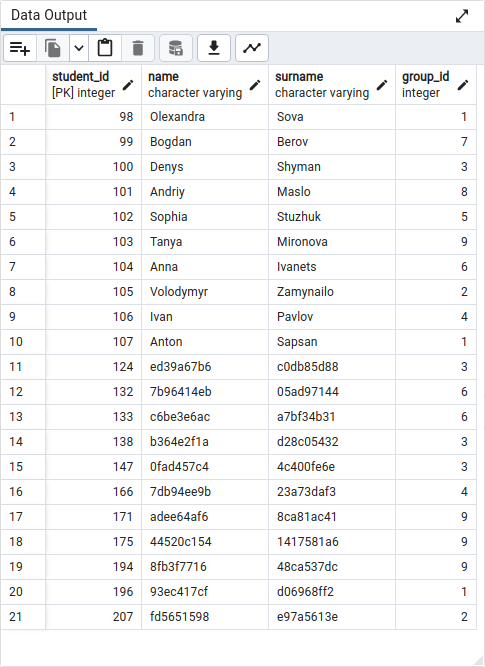


Рисунок 8. Стан таблиці Student

1. Edit

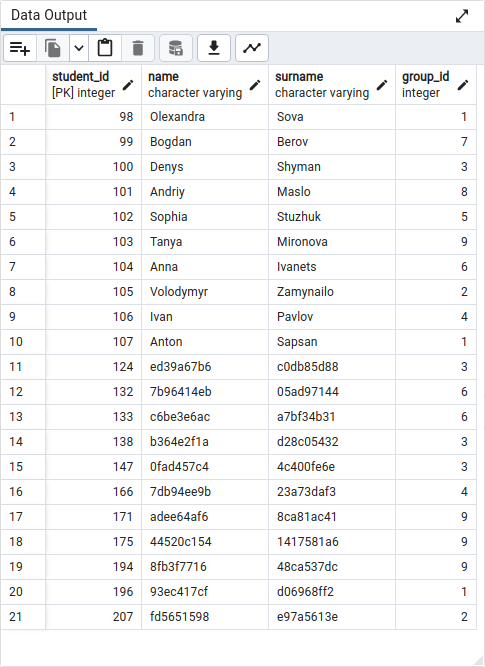


Рисунок 9. Стан таблиці Student

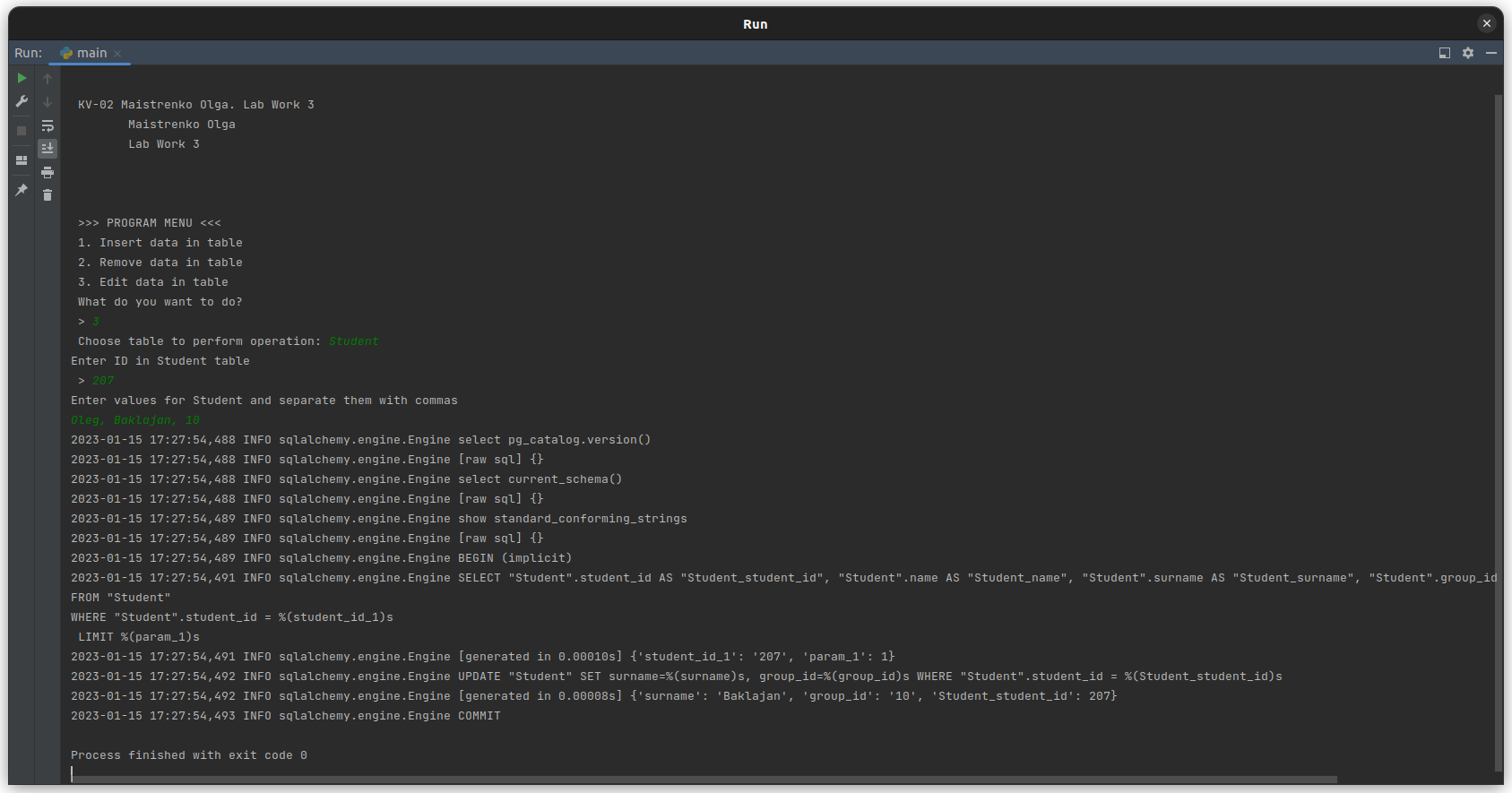


Рисунок 10. Редагування даних в обраній таблиці

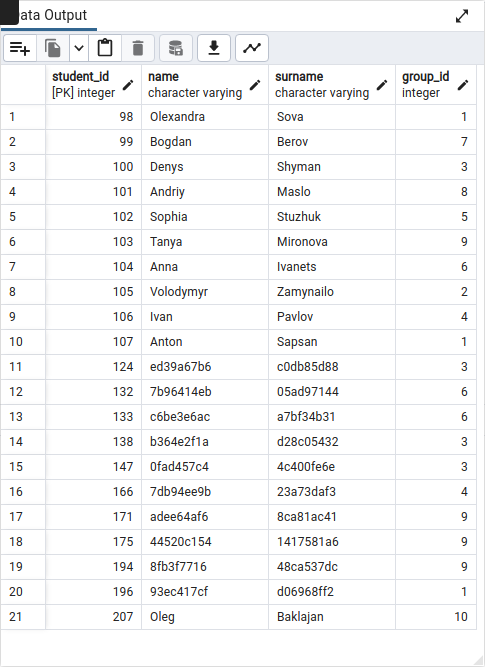


Рисунок 11. Стан таблиці Student

**Тригер бази даних PostgreSQL. Умова для тригера – before update, delete**

Цей тригер спрацьовуватиме щоразу перед тим, як дані в таблиці Student будуть видалені або змінені. При успішному виконанні запиту дані зміненого або віддаленого рядка в таблиці Student будуть записані в таблицю Student\_audit.

**Функція тригера:**

**BEGIN**

**IF** (TG\_OP = 'DELETE') **THEN**

**INSERT** **INTO** public."Student\_audit" ( "student\_id", "log", "timestamp", "name", "surname", "group\_id")

VALUES(**OLD**."student\_id",'Student DELETED', **CURRENT\_TIMESTAMP**, **OLD**."name", **OLD**."surname", **OLD**."group\_id");

ELSIF (TG\_OP = 'UPDATE') **THEN**

**INSERT** **INTO** public."Student\_audit" ( "student\_id", "log", "timestamp", "name", "surname", "group\_id")

VALUES(**NEW**."student\_id",'Student UPDATED', **CURRENT\_TIMESTAMP**, **NEW**."name", **NEW**."surname", **NEW**."group\_id");

**END** **IF**;

**RETURN** **NEW**;

**END**;

Створення тригера:

**CREATE** **TRIGGER** "StudentTrigger"

**BEFORE** **DELETE** **OR** **UPDATE**

**ON** public."Student"

**FOR** **EACH** **ROW**

**EXECUTE** **FUNCTION** public."StudAuditFunc"();

**Демонстрація роботи тригера**

**Видалення даних**

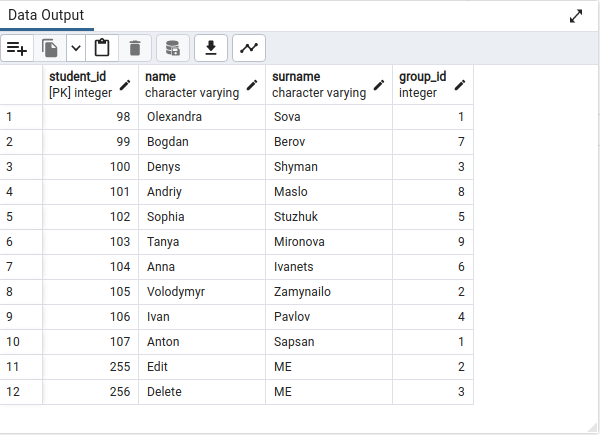


Рисунок 12. Стан таблиці Student

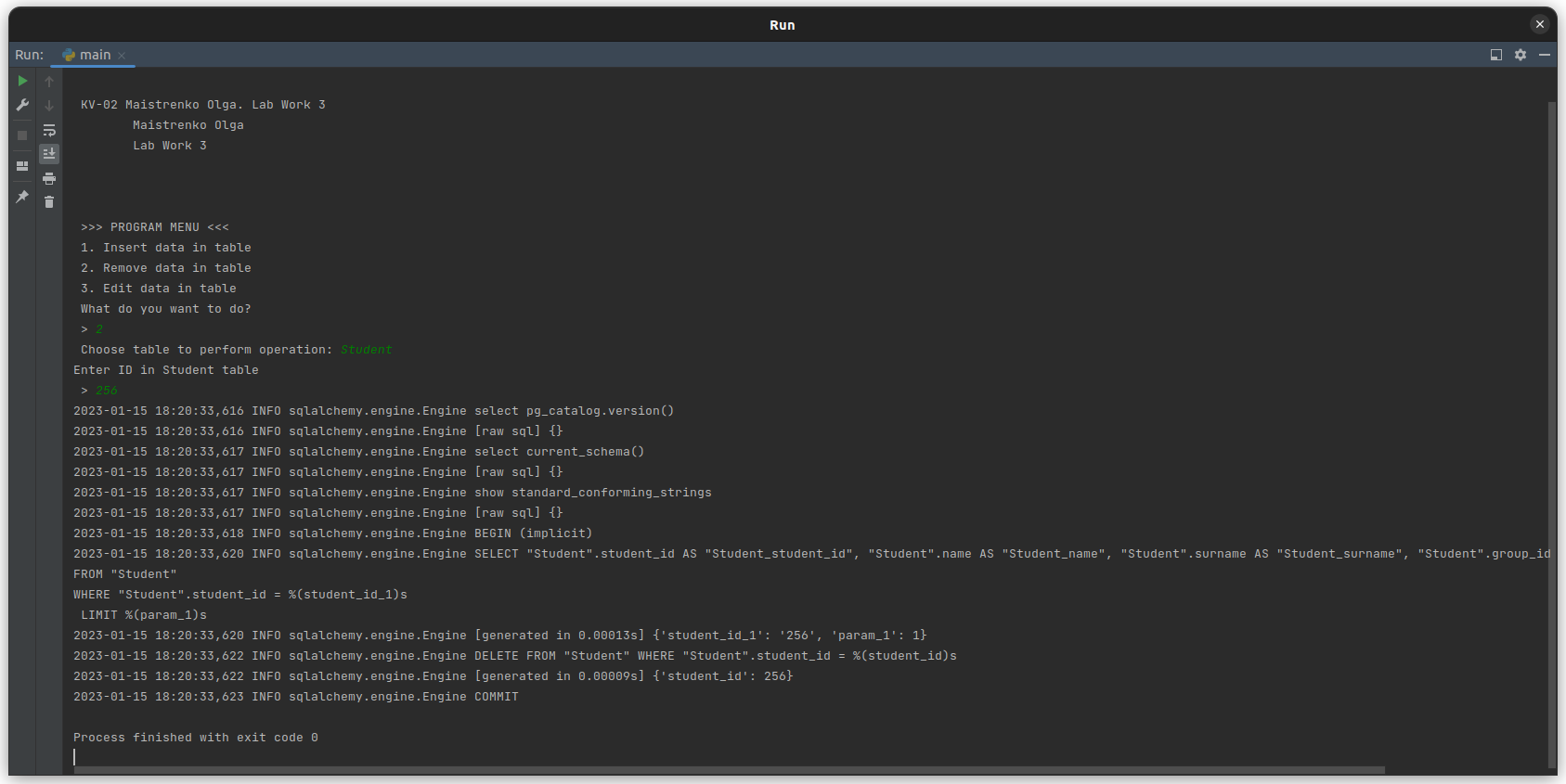


Рисунок 13. Видалення даних з таблиці Student

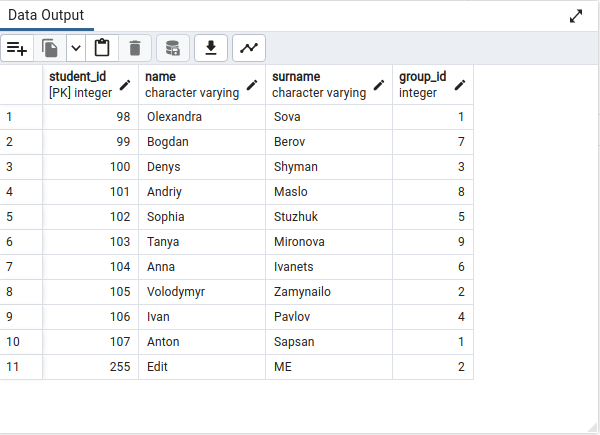


Рисунок 14. Стан таблиці Student

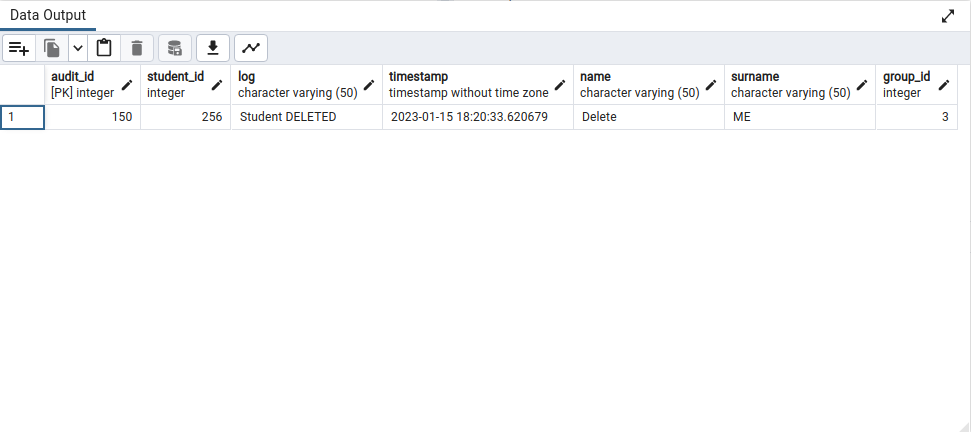


Рисунок 15. Стан таблиці Student\_audit

**Редагування даних**

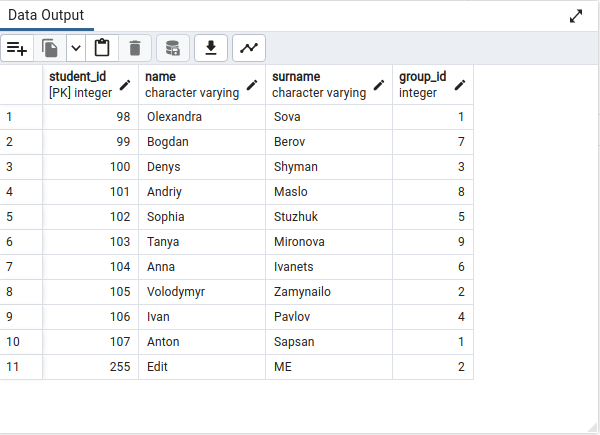


Рисунок 15. Стан таблиці Student

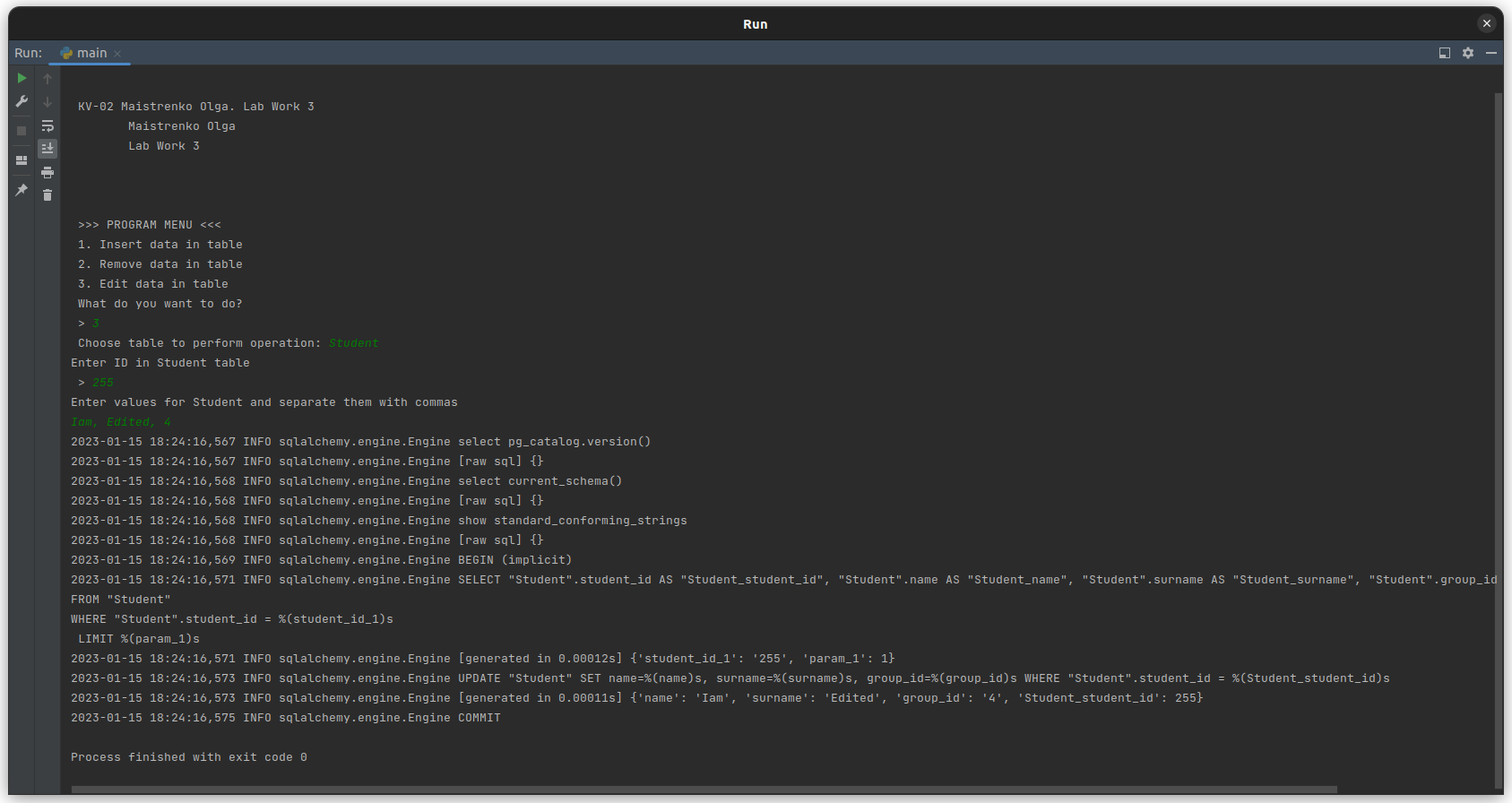


Рисунок 16. Редагування таблиці Student

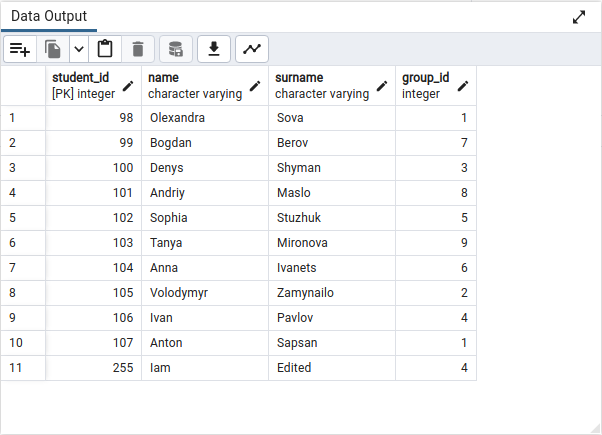


Рисунок 17. Стан таблиці Student

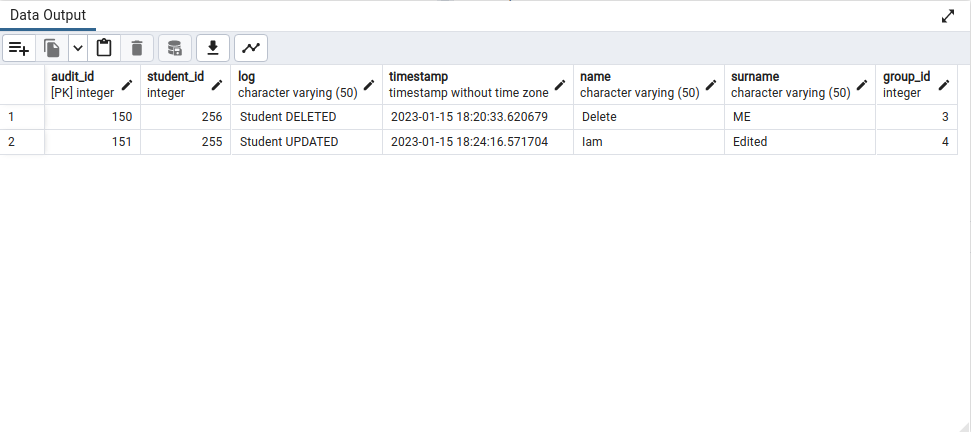


Рисунок 15. Стан таблиці Student\_audit

Як ми бачимо, тригер справно працює і додає до таблиці Student\_audit дані про віддалені або змінені рядки в таблиці Student