Министерство транспорта Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования «Российский университет транспорта»

РУТ (МИИТ)

Институт управления и цифровых технологий

Кафедра «Цифровые технологии управления транспортными процессами»

КУРСОВАЯ РАБОТА ПО ДИСЦИПЛИНЕ:

**“** Технологии хранения данных**”**

НА ТЕМУ:

“Кузня”

Выполнил: Веселков К.О.

Группа: УВП-312

Проверил: Заманов Д. А.

Москва 2023 г.

СОДЕРЖАНИЕ

[ПОСТАНОВКА ЗАДАЧИ 3](#_Toc148318475)

[ИСПОЛЬЗУЕМЫЕ ТЕХНОЛОГИИ 4](#_Toc148318476)

[ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ 5](#_Toc148318477)

[ER-МОДЕЛЬ БАЗЫ ДАННЫХ В НОТАЦИИ IDF1X 7](#_Toc148318478)

[ER-МОДЕЛЬ БАЗЫ ДАННЫХ В НОТАЦИИ ЧЕНА 8](#_Toc148318479)

[СХЕМА РЕЛЯЦИОННОЙ БАЗЫ ДАННЫХ 9](#_Toc148318480)

[ПРОЕКТИРОВАНИЕ РЕЛЯЦИОННОЙ МОДЕЛИ БАЗЫ ДАННЫХ ПО АЛГОРИТМАМ ПРОЕКТИРОВАНИЯ 10](#_Toc148318481)

[ОПИСАНИЕ СХЕМЫ БАЗЫ ДАННЫХ НА ЯЗЫКЕ SQL 11](#_Toc148318482)

[ЗАПРОС НА ЯЗЫКЕ РЕЛЯЦИОННОЙ АЛГЕБРЫ И РЕЛЯЦИОННОМ ИСЧИСЛЕНИИ 14](#_Toc148318483)

[Руководство пользователя по работе с приложением 15](#_Toc148318484)

[Исходный код программы 17](#_Toc148318485)

# ПОСТАНОВКА ЗАДАЧИ

Спроектировать базу данных и разработать приложение для работы с ней. Это приложение должно иметь визуальный интерфейс, позволяющий вводить, изменять и удалять записи из таблиц базы данных. Приложение также должно выполнять поисковые запросы, реализованные в лабораторной работе. На основе выбранной проблемной области должна быть реализована ER-модель в нотации Чена и IDEF1X, а также разработана реляционная модель базы данных и запросы к базе данных в соответствии с созданной ER-моделью. Для выполнения проекта выбирается схема реляционной базы данных, полученная по одному из алгоритмов проектирования. В курсовой работе будет использоваться та же система управления базами данных, что и в лабораторной работе.

# ИСПОЛЬЗУЕМЫЕ ТЕХНОЛОГИИ

1. Tkinter - это стандартная библиотека языка программирования Python, которая предоставляет интерфейс для создания графического пользовательского интерфейса (GUI). Tkinter базируется на библиотеке Tk, которая является кросс-платформенным инструментом для создания GUI.
2. PostgreSQL - это мощная и расширяемая система управления базами данных (СУБД), которая относится к классу реляционных баз данных. PostgreSQL была разработана с акцентом на надежность, расширяемость и соответствие стандартам SQL.
3. Python - это интерпретируемый, высокоуровневый язык программирования, который был разработан в конце 1980-х годов Гвидо ван Россумом. Он обладает простым и понятным синтаксисом, который способствует читаемости и ясности кода, делая его легким для изучения и использования.
4. SQLite - это встроенная система управления базами данных, которая предоставляет легковесный и простой способ хранения и управления данными. Модуль в Python обеспечивает интерфейс для взаимодействия с базой данных SQLite в Python.
5. pgAdmin 4 - это бесплатная и открытая интегрированная среда администрирования для баз данных PostgreSQL. Она предоставляет удобный веб-интерфейс для управления и администрирования серверов PostgreSQL, баз данных, схем, таблиц, пользователей и других объектов базы данных.
6. Draw.io — это бесплатный веб-приложение для создания диаграмм и схем. Оно предоставляет возможность рисовать различные типы диаграмм, включая блок-схемы, диаграммы классов, диаграммы потоков данных, организационные диаграммы, диаграммы баз данных и многие другие.

# ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

Предметная область – “Кузня”. Создана модель сущность-связь в среде Draw.io. Данная модель содержит сущности “Кузнец”, “Изделие”, “Сырье”, “Наковальня”, “Печь” “Поставщик”.

Сущности и их атрибуты:

1. **Blacksmith** (Кузнец)

Фамилия (surname) – строковое значение, отображающее фамилию человека.

Имя (name)– строковое значение, отображающее имя человека.

Дата рождения (birthdate) – формат даты, отображающий дату рождения кузнеца.

Опыт работы (work\_experience) – целочисленное значение, время, которое потратил мастер на улучшение качества и потраченного времени на изделие.

1. **Product** (Изделие)

Тип (type) – строковое значение, отображающее вид изделия.

Размер (size) – строковое значение, отображающее размер изделия.

Назначение (appointment) – строковое значение, отображающее применение изделия.

Вес (weight) – целочисленное значение, вес изделия.

Прочность (endurance) – строковое значение, отображающее прочность изделия.

1. **Raw\_material** (Сырье)

Материал (material) – строковое значение, материала требуемого для создания изделия.

Теплоемкость (heat\_capacity)– целочисленное значение, количество тепла необходимое, чтобы расплавить материал.

Цена (price) – целочисленное значение, показывает цену на сырье.

1. **Anvil** (Наковальня)

Вид (genus) – строковое значение, вид наковальни.

Стойкость (durability) – целочисленное значение, отображающее крепкость наковальни, определяющее ее срок службы.

1. **Smelter** (Печь)

Размер (dimension) – строковое значение, отображающее размер печи.

Материал (fabric) – строковое значение, материал из которого сделана печь.

Срок службы (time\_last) – целочисленное значение, срок через который печь необходимо заменить.

Максимальная температура (maximum\_temperature) – целочисленное значение, максимальная допустимая температура.

1. **Provider** (Поставщик)

Фамилия (surname) – строковое значение, отображающее фамилию человека.

Имя (name)– строковое значение, отображающее имя человека.

Опыт работы (work\_experience) – целочисленное значение, время, которое потратил мастер на улучшение качества и потраченного времени на изделие.

Цена за килограмм (price\_per\_kg) – целочисленное значение, цена за килограмм покупаемого сырья

# ER-МОДЕЛЬ БАЗЫ ДАННЫХ В НОТАЦИИ IDF1X

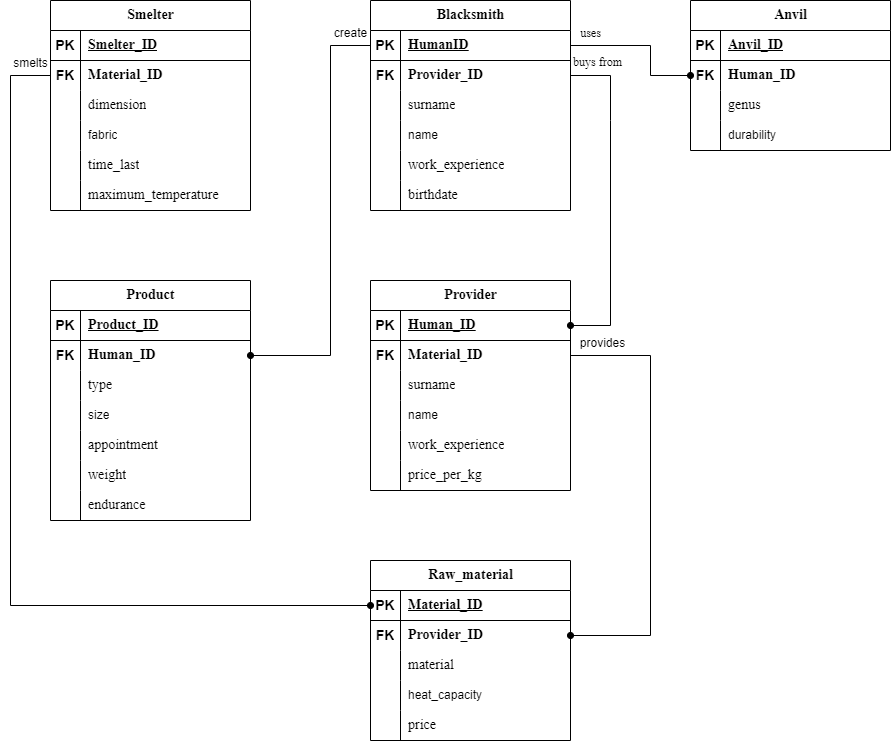


Рисунок 1– Нотация Чена

# ER-МОДЕЛЬ БАЗЫ ДАННЫХ В НОТАЦИИ ЧЕНА

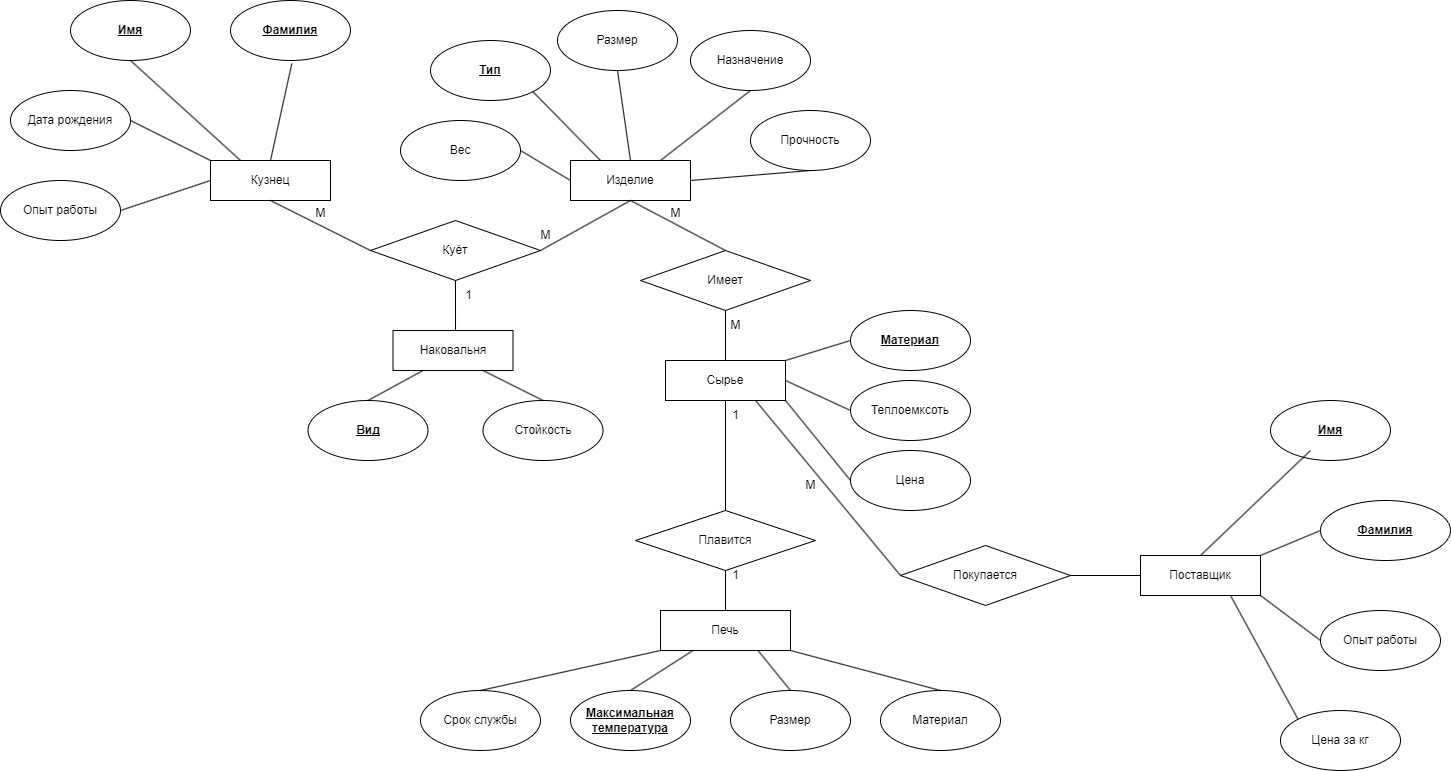


Рисунок 2– Нотация Чена

# СХЕМА РЕЛЯЦИОННОЙ БАЗЫ ДАННЫХ

**Blacksmith (**Human\_ID, Provider\_ID, name, surname, birthdate, work\_experience**)**

**Provider (**Human\_ID, Material\_ID, name, surname, work\_experience, price\_per\_kg**)**

**Raw\_Material (**Material\_ID**,** Provider\_ID, material, heat\_capacity, price**)**

**Anvil (**Anvil\_ID, Human\_ID, genus, durability**)**

**Product (**Product\_ID,Human\_ID, type, size, appointment, weight, endurance**)**

**Smelter (**Smelter\_ID, Product\_ID, dimension, fabric, time\_last, maximum\_temperature**)**

# ПРОЕКТИРОВАНИЕ РЕЛЯЦИОННОЙ МОДЕЛИ БАЗЫ ДАННЫХ ПО АЛГОРИТМАМ ПРОЕКТИРОВАНИЯ

Таблица находится в первой нормальной форме, когда каждый атрибут имеет атомарное значение, то есть не является массивом, списком или другой сложной структурой.

Вторая нормальная форма требует, чтобы каждый не ключевой атрибут был функционально зависим от всего первичного ключа, при этом таблица должна также быть в первой нормальной форме.

Третья нормальная форма требует, чтобы каждый не ключевой атрибут был функционально зависим от всего первичного ключа, а не только от его части, и не зависел от других не ключевых атрибутов. Таблица, находящаяся в третьей нормальной форме, также будет соответствовать второй и первой нормальной форме.

Основываясь на описании структуры таблицы, можно сделать вывод, что данная база данных соответствует второй нормальной форме.

Изображение выглядит как текст, Шрифт, снимок экрана, число

Автоматически созданное описание

# ОПИСАНИЕ СХЕМЫ БАЗЫ ДАННЫХ НА ЯЗЫКЕ SQL

1. Создание таблицы BlackSmith

CREATE TABLE blacksmith (

"human\_ID" integer NOT NULL,

surname VARCHAR(50) NOT NULL,

birthdate DATE NOT NULL,

work\_expirience integer NOT NULL

);

CREATE TABLE

1. Создание таблицы Provide

smith=# CREATE TABLE Provider(

"human\_ID" integer NOT NULL,

name VARCHAR(50) NOT NULL,

surname VARCHAR(50) NOT NULL,

work\_experience integer NOT NULL,

price\_per\_kg\_rub integer NOT NULL

);

CREATE TABLE

1. Создание таблицы Raw\_material

CREATE TABLE Raw\_material L(

"material\_ID" integer NOT NULL,

material VARCHAR(50) NOT NULL,

heat\_capacity\_°C INT NOT NULL,

price INT NOT NULL

);

CREATE TABLE

1. Создание таблицы Anvil

CREATE TABLE Anvil (

"anvil\_ID" integer NOT NULL,

genus INT NOT NULL,

durability INT NOT NULL

);

CREATE TABLE

1. Создание таблицы Smelter

CREATE TABLE Smelter(

"smelter\_ID" integer NOT NULL,

dimension INT NOT NULL,

fabric VARCHAR(50) NOT NULL,

time\_last INT NOT NULL,

maximum\_temperature INT NOT NULL

);

CREATE TABLE

1. Создание таблицы Product

CREATE TABLE Product(

"smelter\_ID" integer NOT NULL,

type VARCHAR(12) NOT NULL,

size INT NOT NULL,

appointment VARCHAR(9) NOT NULL,

weight INT NOT NULL,

endurance INT NOT NULL

);

CREATE TABLE

Описание используемых запросов на языке SQL

1. Найти кузнеца с возрастом: 30 лет

SELECT \* FROM blacksmiths where age = '30'

1. Определить поставщика, работающего 26 лет с ценой за килограмм материала в 32035

SELECT \* FROM Provider where work\_experience = '26' AND price\_per\_kg\_rub = 32035

1. Выбор имен и фамилий кузнецов, тип, размер и вес продуктов, а также размер, материал и время последней плавки в плавильне

SELECT Blacksmith.name AS BlacksmithName, Blacksmith.surname AS BlacksmithSurname,

Product.type AS ProductType, Product.size AS ProductSize, Product.weight AS ProductWeight,

Smelter.dimension AS SmelterDimension, Smelter.fabric AS SmelterFabric, Smelter.time\_last AS SmelterTimeLast

FROM Blacksmith

JOIN Product ON Blacksmith.Human\_ID = Product.Human\_ID

JOIN Smelter ON Product.Product\_ID = Smelter.Product\_ID

WHERE Blacksmith.work\_experience > 5 AND Product.weight > 10;

1. Поиск по именам и фамилиям кузнецов, типам, размерам и весу продуктов, а также виды и прочность наковален.

SELECT Blacksmith.name AS BlacksmithName, Blacksmith.surname AS BlacksmithSurname,

Product.type AS ProductType, Product.size AS ProductSize, Product.weight AS ProductWeight,

Anvil.genus AS AnvilGenus, Anvil.durability AS AnvilDurability

FROM Blacksmith

JOIN Product ON Blacksmith.Human\_ID = Product.Human\_ID

JOIN Anvil ON Blacksmith.Human\_ID = Anvil.Human\_ID

WHERE Blacksmith.work\_experience > 5 AND Product.weight > 10 AND Anvil.durability > 100;

1. Поиск имен и фамилий поставщиков, материалов, их теплоемкость и цены, а также вид и прочность наковален.

SELECT Provider.name AS ProviderName, Provider.surname AS ProviderSurname,

Raw\_Material.material AS RawMaterial, Raw\_Material.heat\_capacity AS HeatCapacity, Raw\_Material.price AS MaterialPrice,

Anvil.genus AS AnvilGenus, Anvil.durability AS AnvilDurability

FROM Provider

JOIN Raw\_Material ON Provider.Material\_ID = Raw\_Material.Material\_ID

JOIN Anvil ON Provider.Human\_ID = Anvil.Human\_ID

WHERE Provider.work\_experience > 3 AND Raw\_Material.price < 50;

1. Найти продукт который является военным, является мечом, весит 35 кг

SELECT \* FROM Product WHERE type = 'Sword' AND appointment = 'War' AND weight = '35';

# ЗАПРОС НА ЯЗЫКЕ РЕЛЯЦИОННОЙ АЛГЕБРЫ И РЕЛЯЦИОННОМ ИСЧИСЛЕНИИ

*Определение:*

1. Получить имена и фамилии кузнецов с опытом работы более 5 лет:  
   σ work\_experience > 5​(Blacksmith)
2. Определить кузнеца для изготовления военного изделия:

π name, surname, type​(Blacksmith⋈Blacksmith.Human\_ID = Product.Human\_ID​Product)

1. Получить только имена, фамилии и типы продуктов от кузнецов:

π product\_ID, type σ appointment =”war” ∧ size ∧ weight = ”product” (Product)

1. Объединение таблицы поставщиков и таблицы сырья для получения всех поставщиков и материалов:

Provider **∪** Raw\_Material)

1. Определить тип наковальни для создания изделия:

πAnvil\_ID σ Product\_ID = durability ∧ type ∧ size ∧ appointment = “anvil\_ID” (Anvil)

1. Определить температуру плавильни для переплавки сырья:

πSmelter\_ID σ Raw\_material\_ID = maximum\_temperature ∧ dimension ∧ heat\_capacity ∧ material = “Product\_ID” (Product)

# Руководство пользователя по работе с приложением

От пользователя требуется знание как минимум простейших SQL – запросов

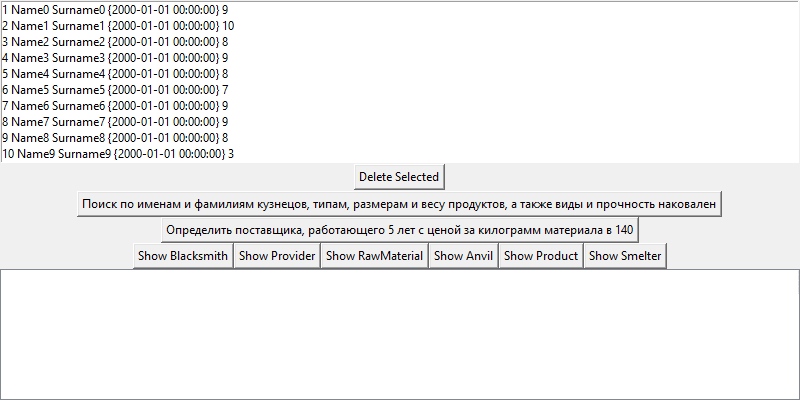


Рисунок 3 – Главное окно

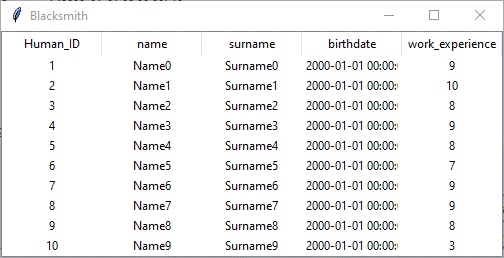


Рисунок 4 – Пример вывода одной из таблиц

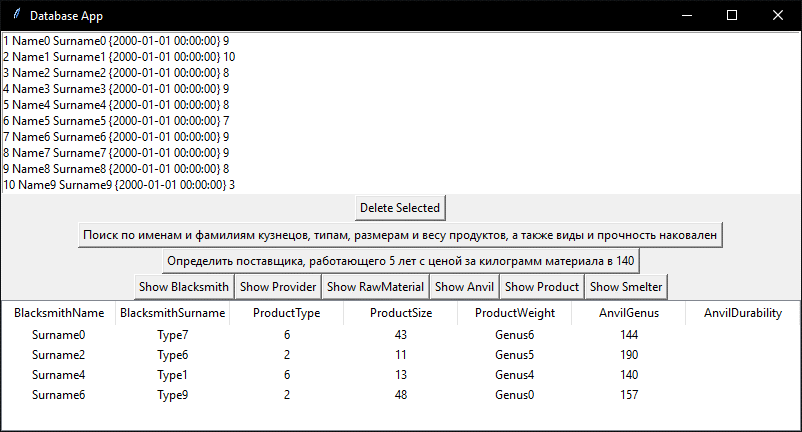


Рисунок 5 – Пример выполнения запроса

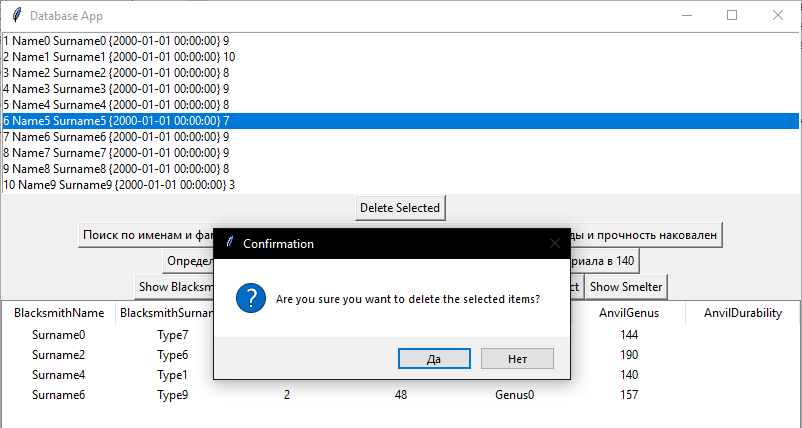
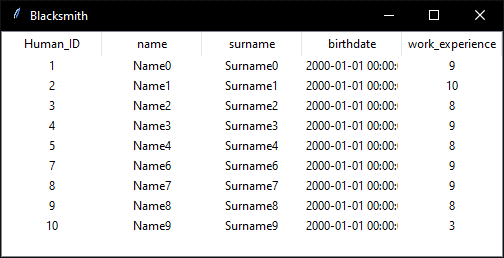
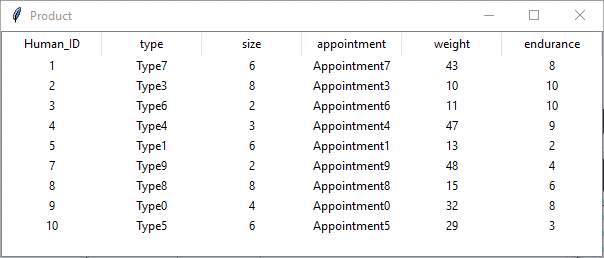


Рисунок 6 – Пример каскадного удаления





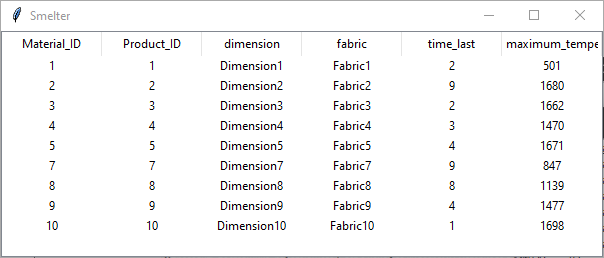


Рисунок 7 – Успешное каскадное удаление

# Исходный код программы

import random  
import sqlite3  
import tkinter  
from tkinter import messagebox, ttk  
import tkinter as tk  
import datetime  
import time  
  
with sqlite3.connect('blacksmith.db') as conn:  
 cursor = conn.cursor()  
  
# Создаем таблицу Blacksmith  
cursor.execute('''  
 CREATE TABLE IF NOT EXISTS Blacksmith (  
 Human\_ID INTEGER PRIMARY KEY,  
 name TEXT,  
 surname TEXT,  
 birthdate DATE,  
 work\_experience INTEGER  
 )  
''')  
  
# Создаем таблицу RawMaterial  
cursor.execute('''  
 CREATE TABLE IF NOT EXISTS RawMaterial (  
 Material\_ID INTEGER PRIMARY KEY AUTOINCREMENT,  
 Provider\_ID INTEGER,  
 material TEXT,  
 heat\_capacity INTEGER,  
 price INTEGER,  
 FOREIGN KEY (Provider\_ID) REFERENCES Provider (Provider\_ID) ON DELETE CASCADE  
 )  
 ''')  
  
# Создаем таблицу Anvil  
cursor.execute('''  
 CREATE TABLE IF NOT EXISTS Anvil (  
 Human\_ID INTEGER PRIMARY KEY,  
 genus TEXT,  
 durability INTEGER,  
 FOREIGN KEY (Human\_ID) REFERENCES Blacksmith(Human\_ID) ON DELETE CASCADE  
 )  
''')  
  
# Создаем таблицу Product  
cursor.execute('''  
 CREATE TABLE IF NOT EXISTS Product (  
 Human\_ID INTEGER PRIMARY KEY,  
 type TEXT,  
 size INTEGER,  
 appointment TEXT,  
 weight INTEGER,  
 endurance INTEGER,  
 FOREIGN KEY (Human\_ID) REFERENCES Blacksmith(Human\_ID) ON DELETE CASCADE  
 )  
''')  
  
# Создаем таблицу Smelter  
cursor.execute('''  
 CREATE TABLE IF NOT EXISTS Smelter (  
 Material\_ID INTEGER PRIMARY KEY,  
 Product\_ID INTEGER,  
 dimension TEXT,  
 fabric TEXT,  
 time\_last INTEGER,  
 maximum\_temperature INTEGER,  
 FOREIGN KEY (Material\_ID) REFERENCES RawMaterial(Material\_ID) ON DELETE CASCADE,  
 FOREIGN KEY (Product\_ID) REFERENCES Product(Human\_ID) ON DELETE CASCADE  
 )  
''')  
  
# Создаем таблицу Provider  
cursor.execute('''  
 CREATE TABLE IF NOT EXISTS Provider (  
 Human\_ID INTEGER PRIMARY KEY AUTOINCREMENT,  
 Material\_ID INTEGER,  
 name TEXT,  
 surname TEXT,  
 work\_experience INTEGER,  
 price\_per\_kg INTEGER,  
 FOREIGN KEY (Material\_ID) REFERENCES RawMaterial (Material\_ID) ON DELETE CASCADE  
 )  
 ''')  
  
conn.commit()  
  
conn = sqlite3.connect('blacksmith.db')  
cursor = conn.cursor()  
  
  
class DatabaseApp:  
 def \_\_init\_\_(self, master):  
 self.master = master  
 self.master.title("Database App")  
 self.master.geometry("800x400")  
  
 self.conn = sqlite3.connect('blacksmith.db')  
 self.cursor = self.conn.cursor()  
  
 self.listbox = tk.Listbox(self.master)  
 self.listbox.pack(expand="yes", fill="both")  
  
 # Кнопка для удаления выбранных записей  
 self.delete\_button = tk.Button(self.master, text="Delete Selected", command=self.delete\_selected)  
 self.delete\_button.pack()  
  
 # Кнопки для открытия каждой таблицы  
 table\_buttons = tk.Frame(self.master)  
 table\_buttons.pack()  
  
 # Заполнение таблиц при запуске приложения  
 self.populate\_tables()  
  
 # Каскадное удаление  
 self.cursor.execute('PRAGMA foreign\_keys = ON')  
  
 # Завершение программы  
 master.protocol("WM\_DELETE\_WINDOW", self.close\_connection)  
  
 # Одна функция create\_widgets  
 self.create\_widgets()  
  
 def populate\_tables(self):  
 # Проверяем наличие данных в таблицах,  
 existing\_data = self.cursor.execute('SELECT COUNT(\*) FROM Blacksmith').fetchone()[0]  
  
 if existing\_data == 0:  
 # Заполняем таблицу Blacksmith  
 for \_ in range(10):  
 name = f"Name{\_}"  
 surname = f"Surname{\_}"  
 birthdate = datetime.datetime(2000, 1, 1)  
 work\_experience = random.randint(1, 10)  
 self.cursor.execute('''  
 INSERT INTO Blacksmith (name, surname, birthdate, work\_experience)  
 VALUES (?, ?, ?, ?)  
 ''', (name, surname, birthdate, work\_experience))  
  
 # Заполняем таблицу RawMaterial (пример заполнения, вы можете изменить по вашему усмотрению)  
 materials = ["Steel", "Iron", "Aluminum", "Copper", "Gold"]  
 used\_provider\_ids = set()  
 for provider\_id in range(1, 11):  
 while provider\_id in used\_provider\_ids:  
 provider\_id = random.randint(1, 10)  
  
 material = random.choice(materials)  
 heat\_capacity = random.randint(200, 1000)  
 price = random.randint(100, 500)  
 used\_provider\_ids.add(provider\_id)  
  
 self.cursor.execute('''  
 INSERT INTO RawMaterial (Provider\_ID, material, heat\_capacity, price)  
 VALUES (?, ?, ?, ?)  
 ''', (provider\_id, material, heat\_capacity, price))  
  
 self.conn.commit()  
  
 # Заполняем Listbox  
 self.execute\_query()  
  
 # Заполняем таблицу Provider  
 for \_ in range(10):  
 name = f"ProviderName{\_}"  
 surname = f"ProviderSurname{\_}"  
 work\_experience = random.randint(1, 10)  
 price\_per\_kg = random.randint(50, 200)  
  
 existing\_human\_ids = [row[0] for row in self.cursor.execute('SELECT Human\_ID FROM Provider').fetchall()]  
 used\_ids = set(existing\_human\_ids)  
 while True:  
 human\_id = random.randint(1, 10)  
 if human\_id not in used\_ids:  
 used\_ids.add(human\_id)  
 break  
  
 material\_id = random.randint(1, 10)  
  
 self.cursor.execute('''  
 INSERT INTO Provider (Human\_ID, Material\_ID, name, surname, work\_experience, price\_per\_kg)  
 VALUES (?, ?, ?, ?, ?, ?)  
 ''', (human\_id, material\_id, name, surname, work\_experience, price\_per\_kg))  
  
 # Заполняем таблицу Anvil  
 for \_ in range(10):  
 genus = f"Genus{\_}"  
 durability = random.randint(50, 200)  
 existing\_human\_ids = [row[0] for row in self.cursor.execute('SELECT Human\_ID FROM Anvil').fetchall()]  
  
 human\_id = random.randint(1, 10)  
 while human\_id in existing\_human\_ids:  
 human\_id = random.randint(1, 10)  
  
 self.cursor.execute('''  
 INSERT INTO Anvil (Human\_ID, genus, durability)  
 VALUES (?, ?, ?)  
 ''', (human\_id, genus, durability))  
  
 # Заполняем таблицу Product  
 for \_ in range(10):  
 type\_ = f"Type{\_}"  
 size = random.randint(1, 10)  
 appointment = f"Appointment{\_}"  
 weight = random.randint(5, 50)  
 endurance = random.randint(1, 10)  
  
 existing\_human\_ids = [row[0] for row in self.cursor.execute('SELECT Human\_ID FROM Product').fetchall()]  
 human\_id = random.randint(1, 10)  
 while human\_id in existing\_human\_ids:  
 human\_id = random.randint(1, 10)  
  
 self.cursor.execute('''  
 INSERT INTO Product (Human\_ID, type, size, appointment, weight, endurance)  
 VALUES (?, ?, ?, ?, ?, ?)  
 ''', (human\_id, type\_, size, appointment, weight, endurance))  
  
 # Заполняем таблицу Smelter  
 for product\_id in range(1, 11):  
 dimension = f"Dimension{product\_id}"  
 fabric = f"Fabric{product\_id}"  
 time\_last = random.randint(1, 10)  
 maximum\_temperature = random.randint(500, 2000)  
 time.sleep(0.1)  
 self.cursor.execute('''  
 INSERT INTO Smelter (Product\_ID, dimension, fabric, time\_last, maximum\_temperature)  
 VALUES (?, ?, ?, ?, ?)  
 ''', (product\_id, dimension, fabric, time\_last, maximum\_temperature))  
  
 self.conn.commit()  
  
 def close\_connection(self):  
 # Закрытие соединения при выходе из программы  
 self.conn.commit()  
 self.conn.close()  
 self.master.destroy()  
  
 def create\_widgets(self):  
 self.custom\_query\_button = tk.Button(self.master, text="Поиск по именам и фамилиям кузнецов, "  
 "типам, размерам и весу продуктов, "  
 "а также виды и прочность наковален",  
 command=self.execute\_custom\_query\_custom)  
 self.custom\_query\_button.pack()  
  
 self.new\_query\_button = tk.Button(self.master, text="Определить поставщика, работающего 5 лет с "  
 "ценой за килограмм материала в 140",  
 command=self.execute\_new\_query)  
 self.new\_query\_button.pack()  
  
 # Кнопки для открытия каждой таблицы  
 table\_buttons = tk.Frame(self.master)  
 table\_buttons.pack()  
  
 tables = ["Blacksmith", "Provider", "RawMaterial", "Anvil", "Product", "Smelter"]  
 for table\_name in tables:  
 button = tk.Button(table\_buttons, text=f"Show {table\_name}",  
 command=lambda name=table\_name: self.display\_table(name))  
 button.pack(side="left")  
  
 self.treeview = ttk.Treeview(self.master)  
 self.treeview.pack(expand="yes", fill="both")  
  
 # Заполнение Treeview при создании  
 self.execute\_query()  
  
 def execute\_new\_query(self):  
 query\_text = "SELECT \* FROM Provider WHERE work\_experience = '5' AND price\_per\_kg = 140"  
  
 try:  
 # Результаты запроса  
 result = self.cursor.execute(query\_text).fetchall()  
  
 result\_window = tk.Toplevel(self.master)  
 result\_window.title("New Query Results")  
  
 result\_window.geometry("400x200")  
  
 result\_listbox = tk.Listbox(result\_window, selectmode=tk.SINGLE)  
 result\_listbox.pack(expand=True, fill="both")  
  
 for i, row in enumerate(result, 1):  
 result\_listbox.insert(tk.END, f"{i}: {row}")  
  
 except Exception as e:  
 messagebox.showerror("Error", f"Query execution error: {str(e)}")  
  
 def execute\_query(self, query\_text=None):  
 if not query\_text:  
 query\_text = "SELECT \* FROM Blacksmith"  
 self.cursor.execute(query\_text)  
 rows = self.cursor.fetchall()  
  
 # Очищаем Listbox перед заполнением новыми данными  
 self.listbox.delete(0, tk.END)  
  
 # Заполняем данными  
 for row in rows:  
 self.listbox.insert(tk.END, row)  
  
 def delete\_selected(self):  
 # Получаем выделенные элементы Listbox  
 selected\_items = self.listbox.curselection()  
  
 if not selected\_items:  
 messagebox.showinfo("Information", "No items selected for deletion.")  
 return  
  
 # Запрос пользователя на подтверждение удаления  
 confirmation = messagebox.askyesno("Confirmation", "Are you sure you want to delete the selected items?")  
  
 if confirmation:  
 for item\_index in reversed(selected\_items):  
 # Получаем ID записи из Listbox  
 item\_id = self.listbox.get(item\_index)[0]  
 self.cursor.execute(f"DELETE FROM Blacksmith WHERE Human\_ID = ?", (item\_id,))  
  
 # Сохраняем изменения в базе данных  
 self.conn.commit()  
  
 # Обновляем Listbox после удаления  
 self.execute\_query()  
  
 def execute\_custom\_query\_custom(self):  
 query\_text = """  
 SELECT Blacksmith.name AS BlacksmithName, Blacksmith.surname AS BlacksmithSurname,  
 Product.type AS ProductType, Product.size AS ProductSize, Product.weight AS ProductWeight,  
 Anvil.genus AS AnvilGenus, Anvil.durability AS AnvilDurability  
 FROM Blacksmith  
 JOIN Product ON Blacksmith.Human\_ID = Product.Human\_ID  
 JOIN Anvil ON Blacksmith.Human\_ID = Anvil.Human\_ID  
 WHERE Blacksmith.work\_experience > 5 AND Product.weight > 10 AND Anvil.durability > 100  
 """  
  
 try:  
 result = self.cursor.execute(query\_text).fetchall()  
  
 result\_window = tk.Toplevel(self.master)  
 result\_window.title("Custom Query Results")  
  
 result\_window.geometry("400x200")  
  
 result\_listbox = tk.Listbox(result\_window, selectmode=tk.SINGLE)  
 result\_listbox.pack(expand=True, fill="both")  
  
 for i, row in enumerate(result, 1):  
 result\_listbox.insert(tk.END, f"{i}: {row}")  
  
 except Exception as e:  
 messagebox.showerror("Error", f"Query execution error: {str(e)}")  
  
 for item in self.treeview.get\_children():  
 self.treeview.delete(item)  
  
 columns = [desc[0] for desc in self.cursor.description]  
 self.treeview["columns"] = columns  
 self.treeview["show"] = "headings"  
  
 for col in columns:  
 self.treeview.heading(col, text=col)  
 self.treeview.column(col, anchor=tk.CENTER, width=100)  
  
 for i, row in enumerate(result):  
 self.treeview.insert("", i, text=row[0], values=row[1:])  
  
 def display\_table(self, table\_name):  
 rows = self.cursor.execute(f'SELECT \* FROM {table\_name}').fetchall()  
  
 # Получаем имена столбцов из описания таблицы  
 self.cursor.execute(f'PRAGMA table\_info({table\_name})')  
 column\_names = [column[1] for column in self.cursor.fetchall()]  
  
 table\_window = tk.Toplevel(self.master)  
 table\_window.title(table\_name)  
  
 treeview = ttk.Treeview(table\_window)  
  
 treeview["columns"] = column\_names  
 treeview.column("#0", width=0, stretch="no")  
 for col in column\_names:  
 treeview.heading(col, text=col)  
 treeview.column(col, anchor="center", width=100)  
  
 for row in rows:  
 values = [str(i) if i is not None else "" for i in row]  
 treeview.insert("", "end", values=values)  
  
 treeview.pack(fill="both", expand=True)  
  
  
root = tk.Tk()  
app = DatabaseApp(root)  
root.mainloop()