Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Хранение и управление данными

ОТЧЕТ

по лабораторной работе № 3

ОСВОЕНИЕ ПРИКЛАДНОГО ИНТЕРФЕЙСА СУБД BERKELEYDB. РАЗРАБОТКА КОНВЕРТОРА БАЗЫ ДАННЫХ POSTGRESQL В НАБОР БАЗ ДАННЫХ BERKELEY DB. АДАПТАЦИЯ СПЕЦИФИКАЦИЙ ПРИЛОЖЕНИЯ.

Студент: Л.А. Мацкевич

Преподаватель: Ю.Ю. Желтко

МИНСК 2024

**ЦЕЛЬ РАБОТЫ**

1. Научиться преобразовывать реляционные базы данных (PostgreSQL) в формат ключ-значение (Berkeley DB).
2. Освоить процесс сериализации и десериализации данных для хранения в нереляционной базе данных.
3. Выполнить адаптацию существующих спецификаций приложения для работы с Berkeley DB.

**1 СТРУКТУРЫ ХРАНЕНИЯ ДАННЫХ В BERKELEY DB**

На рисунке 1.1 представлена схема базы данных «Больница».

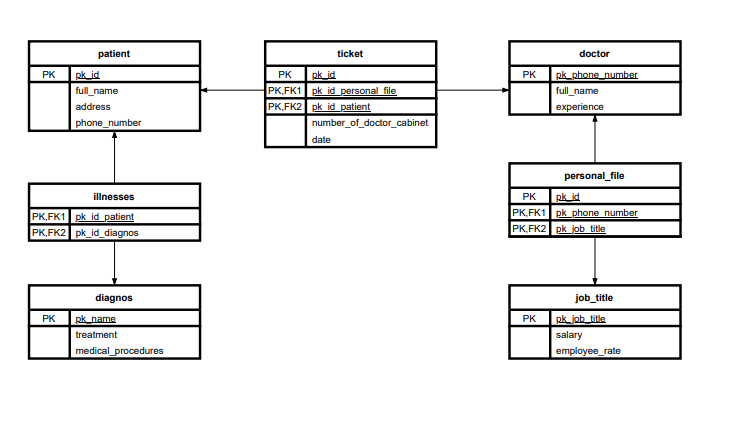


Рисунок 2.1 – схема базы данных

В BDB таблицу patinet можно представить как пару «ключ-значение», где в качестве ключа используется pk\_id, а значение − это оставшаяся информация о пациенте (например, full\_name, address, full\_name).

Таблицу ticket можно представить как пару «ключ-значение», где ключ − это pk\_id, а значение будет содержать поля pk\_id\_patinet, pk\_id\_doctor, date и number\_of\_doctor\_cabinet

Для таблицы diagnosis каждый диагноз будет иметь уникальный pk\_id в качестве ключа, а значением будет информация о диагнозе (например, medical\_procedures).

Для таблицы doctor каждый доктор будет иметь уникальный pk\_id в качестве ключа, а значением будет информация о докторе (full\_name, medical\_procedures).

Для таблицы illnesses в качестве ключа будет использоваться pk\_id, а значение будет содержать информацию о заболевании пациента (pk\_id\_patient,pk\_id\_diagnosis).

**2 ПРИМЕР SQL-ЗАПРОСОВ ДЛЯ ИЗВЛЕЧЕНИЯ СХЕМЫ И ДАННЫХ ИЗ POSTGRESQL**

Для извлечения схемы и данных из PostgreSQL можно использовать несколько типов SQL-запросов.

Чтобы извлечь список всех таблиц в базе данных, можно использовать следующий запрос:

SELECT table\_schema, table\_name

FROM information\_schema.tables

WHERE table\_type = 'BASE TABLE'

ORDER BY table\_schema, table\_name;

Чтобы получить информацию о столбцах конкретной таблицы, включая типы данных и дополнительные атрибуты, можно использовать запрос к information\_schema.columns:

SELECT column\_name, data\_type, is\_nullable, character\_maximum\_length, numeric\_precision, numeric\_scale

FROM information\_schema.columns

WHERE table\_schema = 'public'

AND table\_name = 'clients'

ORDER BY ordinal\_position;

Для извлечения информации о первичных ключах таблиц можно использовать запрос к таблице information\_schema.table\_constraints и information\_schema.key\_column\_usage:

SELECT tc.table\_schema, tc.table\_name, kcu.column\_name

FROM information\_schema.table\_constraints AS tc

JOIN information\_schema.key\_column\_usage AS kcu

ON tc.constraint\_name = kcu.constraint\_name

WHERE tc.constraint\_type = 'PRIMARY KEY'

AND tc.table\_schema = 'public'

AND tc.table\_name = 'clients';

Чтобы извлечь все данные из таблицы, можно использовать стандартный SQL-запрос:

SELECT \* FROM clients;

SELECT \* FROM contracts;

SELECT \* FROM services;

SELECT \* FROM contents;

SELECT \* FROM employee;

SELECT \* FROM maintenance;

**3 РАБОТА С BERKELEY DB**

Сериализация и десериализация — это процессы, которые используются для преобразования данных между различными форматами, чтобы их можно было передавать или хранить. Эти процессы особенно важны в контексте работы с данными, которые необходимо сохранять в файлах, передавать по сети или между различными приложениями.

Код конвертора данных из PostgreSQL в Berkeley DB, включая сериализацию и десериализацию, находится в приложении А.

Для вставки данных в Berkeley DB используется метод put(), который сохраняет пару ключ-значение в базе данных.

Для обновления данных в Berkeley DB необходимо использовать метод put(), как для вставки. Если ключ уже существует, новое значение просто перезапишет старое.

Пример вставки и обновления данных в Berkeley DB:

db\_env = bdb.DBEnv()

db\_env.open(".", bdb.DB\_CREATE | bdb.DB\_INIT\_MPOOL)

db = bdb.DB(db\_env)

db.open("example.db", None, bdb.DB\_HASH, bdb.DB\_CREATE)

db.put(b"key1", b"value1")

db.put(b"key2", b"value2")

db.put(b"key3", b"value3")

db.put(b"key1", b"new\_value1")

db.put(b"key2", b"new\_value2")

Для чтения данных из Berkeley DB используется метод get(), который возвращает значение по заданному ключу. Пример чтения данных:

db\_env = bdb.DBEnv()

db\_env.open(".", bdb.DB\_CREATE | bdb.DB\_INIT\_MPOOL)

db = bdb.DB(db\_env)

db.open("example.db", None, bdb.DB\_HASH)

value = db.get(b"key1")

Для удаления данных из Berkeley DB используется метод delete(), который удаляет пару ключ-значение по заданному ключу:

db\_env = bdb.DBEnv()

db\_env.open(".", bdb.DB\_CREATE | bdb.DB\_INIT\_MPOOL)

db = bdb.DB(db\_env)

db.open("example.db", None, bdb.DB\_HASH)

db.delete(b"key2")

**4 ОПИСАНИЕ ИЗМЕНЕНИЙ В СПЕЦИФИКАЦИЯХ ПРИЛОЖЕНИЯ, АДАПТИРОВАННЫХ ДЛЯ РАБОТЫ С BERKELEY DB.**

При адаптации приложения для работы с Berkeley DB вместо традиционной реляционной базы данных (PostgreSQL), необходимо внести изменения в спецификации приложения, чтобы учесть особенности хранения данных в Berkeley DB, такой как работа с парой ключ-значение, а также API для выполнения операций с данными.

def convert\_all(self):

        try:

            pg\_conn = psycopg2.connect(\*\*self.pg\_conn\_params)

            pg\_cursor = pg\_conn.cursor()

            for entity, db\_filename in self.entities.items():

                table\_name = entity.lower().replace(" ", "\_")

                pg\_cursor.execute(f"SELECT \* FROM {table\_name};")

                rows = pg\_cursor.fetchall()

                columns = [desc[0] for desc in pg\_cursor.description]

                berkeley\_db = db.DB()

                berkeley\_db.open(db\_filename, None, db.DB\_HASH, db.DB\_CREATE)

                for row in rows:

                    primary\_key = row[0]  *# Предполагаем, что первичный ключ — первый столбец*

*# Исключаем первичный ключ из value*

                    value = {columns[i]: row[i] for i in range(1, len(columns))}

                    serialized\_value = pickle.dumps(value)

                    berkeley\_db.put(str(primary\_key).encode(), serialized\_value)

                berkeley\_db.close()

            pg\_cursor.close()

            pg\_conn.close()

            messagebox.showinfo("Success", "All tables have been converted successfully.")

        except Exception as e:

            messagebox.showerror("Error", f"Failed to convert tables: {e}")

**5 ВЫВОД**

Целью работы было освоить процесс преобразования данных из реляционной базы данных PostgreSQL в формат ключ-значение, используемый в Berkeley DB, а также изучить принципы сериализации и десериализации данных для их эффективного хранения в нереляционной базе данных. В ходе работы было проведено преобразование структуры данных и разработаны методы для адаптации существующих спецификаций приложения под новый формат хранения.

На основе выполнения задачи было показано, что преобразование реляционных данных в формат ключ-значение требует не только изменения способа хранения информации, но и адаптации логики работы с данными, поскольку Berkeley DB не поддерживает SQL-запросы и сложные операции над данными, такие как объединение таблиц или агрегации. Вместо этого, для работы с данными используется простой и быстрый механизм записи и чтения по ключу.

Процесс сериализации и десериализации данных также оказался важным этапом работы, поскольку данные из PostgreSQL, представляющие собой сложные структуры, необходимо было привести к удобному для хранения виду и впоследствии восстановить их в исходный формат. Это требовало использования механизмов сериализации объектов и работы с байтовыми потоками.

PostgreSQL − это мощная и универсальная СУБД, идеально подходящая для работы с большими объемами структурированных данных, поддерживающая сложные запросы и транзакции. Однако, она требует больше системных ресурсов и опыта для администрирования, что может быть проблемой для небольших и ограниченных по ресурсам приложений.

С другой стороны, Berkeley DB предлагает высокую производительность и простоту в использовании для приложений, где не требуется сложная логика работы с данными. Она идеально подходит для встроенных решений, приложений с высокими требованиями к скорости обработки данных, но не поддерживает SQL и имеет ограниченные возможности для работы с более сложными данными и многопользовательскими системами.

**ПРИЛОЖЕНИЕ А**

*(обязательное)*

Исходный текст программы

Содержимое файла main.py:

import tkinter as tk

from tkinter import ttk, messagebox

from bsddb3 import db

import pickle

import psycopg2

class BerkeleyDBApp:

    def \_\_init\_\_(self, root):

        self.root = root

        self.root.title("Berkeley DB Management")

        self.root.geometry("600x500")

*# PostgreSQL connection parameters*

        self.pg\_conn\_params = {

            "host": "172.19.32.1",

            "port": 5432,

            "database": "postgres",

            "user": "postgres",

            "password": "7969930n"

        }

        self.entities = {

            "Patient": "patient.db",

            "Doctor": "doctor.db",

            "Ticket": "ticket.db",

            "Diagnosis": "diagnosis.db",

            "Illnesses": "illnesses.db",

        }

        self.current\_entity = None

        self.db\_filename = None

        self.create\_widgets()

    def create\_widgets(self):

*# Dropdown menu to select entity*

        self.entity\_var = tk.StringVar()

        self.entity\_menu = ttk.Combobox(

            self.root, textvariable=self.entity\_var, values=list(self.entities.keys()), state="readonly"

        )

        self.entity\_menu.bind("<<ComboboxSelected>>", self.load\_entity)

        self.entity\_menu.pack(pady=10)

*# Scrollable Table to display data*

        self.table\_frame = tk.Frame(self.root)

        self.table\_frame.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

        self.table\_canvas = tk.Canvas(self.table\_frame)

        self.scrollbar\_y = ttk.Scrollbar(self.table\_frame, orient="vertical", command=self.table\_canvas.yview)

        self.scrollbar\_x = ttk.Scrollbar(self.table\_frame, orient="horizontal", command=self.table\_canvas.xview)

        self.scrollbar\_y.pack(side=tk.RIGHT, fill=tk.Y)

        self.scrollbar\_x.pack(side=tk.BOTTOM, fill=tk.X)

        self.table\_canvas.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

        self.table\_canvas.configure(yscrollcommand=self.scrollbar\_y.set, xscrollcommand=self.scrollbar\_x.set)

        self.table\_canvas.bind("<Configure>", lambda e: self.table\_canvas.configure(scrollregion=self.table\_canvas.bbox("all")))

        self.table\_inner\_frame = tk.Frame(self.table\_canvas)

        self.table\_canvas.create\_window((0, 0), window=self.table\_inner\_frame, anchor="nw", width=780, height=500)

        self.table = ttk.Treeview(self.table\_inner\_frame, show='headings')

        self.table.pack(fill=tk.BOTH, expand=True)

        self.table.bind("<ButtonPress-1>", self.disable\_table\_dragging)

        self.table.bind("<ButtonRelease-1>", self.fill\_fields\_on\_row\_select)

*# Entry fields for new data*

        self.entry\_frame = tk.Frame(self.root)

        self.entry\_frame.pack(pady=10)

        self.entry\_labels = []

        self.entry\_fields = []

*# Buttons*

        self.button\_frame = tk.Frame(self.root)

        self.button\_frame.pack(pady=10)

        self.add\_button = tk.Button(self.button\_frame, text="Add Record", command=self.add\_record)

        self.add\_button.pack(side=tk.LEFT, padx=5)

        self.delete\_button = tk.Button(self.button\_frame, text="Delete Record", command=self.delete\_record)

        self.delete\_button.pack(side=tk.LEFT, padx=5)

        self.update\_button = tk.Button(self.button\_frame, text="Update Record", command=self.update\_record)

        self.update\_button.pack(side=tk.LEFT, padx=5)

        self.convert\_button = tk.Button(self.button\_frame, text="Convert All", command=self.convert\_all)

        self.convert\_button.pack(side=tk.LEFT, padx=5)

*# Search bar for filtering records*

        self.search\_label = tk.Label(self.root, text="Search:")

        self.search\_label.pack(pady=5)

        self.search\_entry = tk.Entry(self.root)

        self.search\_entry.pack(pady=5)

        self.search\_entry.bind("<KeyRelease>", self.search\_records)

    def disable\_table\_dragging(self, event):

*# Prevent the table headings from being draggable*

        if self.table.identify\_region(event.x, event.y) == "separator":

            return "break"

    def load\_entity(self, event):

        self.current\_entity = self.entity\_var.get()

        self.db\_filename = self.entities[self.current\_entity]

        self.load\_data\_structure()

        self.load\_data()

    def load\_data\_structure(self):

*# Столбцы без 'ID'*

        columns = {

            "Patient": ["Full Name", "Address", "Phone Number"],

            "Doctor": ["Name", "Specialty", "Phone Number"],

            "Ticket": [ "Patient ID", "Doctor ID", "Date","Number Cabinet"],

            "Diagnosis": ["Description"],

            "Illnesses": ["id\_patient", "id\_diagnosis"],

        }

        for label in self.entry\_labels:

            label.destroy()

        for field in self.entry\_fields:

            field.destroy()

        self.entry\_labels = []

        self.entry\_fields = []

        self.table.delete(\*self.table.get\_children())

        self.table.configure(columns=columns[self.current\_entity])

        for col in columns[self.current\_entity]:

            self.table.heading(col, text=col)

            self.table.column(col, width=100)

*# Создание текстовых полей ввода*

            label = tk.Label(self.entry\_frame, text=col)

            label.pack(side=tk.LEFT, padx=5)

            self.entry\_labels.append(label)

            entry = tk.Entry(self.entry\_frame)

            entry.pack(side=tk.LEFT, padx=5)

            self.entry\_fields.append(entry)

    def load\_data(self):

*# Удалить старые данные из таблицы*

        self.table.delete(\*self.table.get\_children())

        try:

            berkeley\_db = db.DB()

            berkeley\_db.open(self.db\_filename, None, db.DB\_HASH, db.DB\_RDONLY)

            cursor = berkeley\_db.cursor()

            record = cursor.first()

            while record is not None:

                key, value = record

                deserialized\_value = pickle.loads(value)

*# Преобразуем dict\_values в список*

                self.table.insert('', 'end',iid=key.decode(), values=list(deserialized\_value.values()))

                record = cursor.next()

            cursor.close()

            berkeley\_db.close()

        except Exception as e:

            messagebox.showerror("Error", f"Failed to load data: {e}")

    def add\_record(self):

        try:

            berkeley\_db = db.DB()

            berkeley\_db.open(self.db\_filename, None, db.DB\_HASH, db.DB\_CREATE)

*# Get the last ID by scanning all records and finding the maximum ID*

            cursor = berkeley\_db.cursor()

            last\_id = 0

            record = cursor.first()

            while record is not None:

                key, value = record

                current\_id = int(key.decode('utf-8'))

                last\_id = max(last\_id, current\_id)  *# Update last\_id if current ID is greater*

                record = cursor.next()

            cursor.close()

*# Increment the maximum ID by 1 to get a new unique ID for the new record*

            new\_id = last\_id + 1

*# Create a new record with the new ID*

            new\_record = {}

            for i, field in enumerate(self.entry\_fields):

                new\_record[self.table["columns"][i]] = field.get()

*# Save the new record with the unique ID*

            berkeley\_db.put(str(new\_id).encode('utf-8'), pickle.dumps(new\_record))

            berkeley\_db.close()

            self.load\_data()

        except Exception as e:

            messagebox.showerror("Error", f"Failed to add record: {e}")

    def delete\_record(self):

        selected\_item = self.table.focus()

        if not selected\_item:

            messagebox.showwarning("Warning", "Please select a record to delete.")

            return

        try:

            record\_id = selected\_item

            berkeley\_db = db.DB()

            berkeley\_db.open(self.db\_filename, None, db.DB\_HASH, db.DB\_CREATE)

            berkeley\_db.delete(str(record\_id).encode('utf-8'))

            berkeley\_db.close()

            if self.current\_entity in ["Patient", "Doctor"]:

                berkeley\_db = db.DB()

                berkeley\_db.open(f"ticket.db", None, db.DB\_HASH, db.DB\_CREATE)

                cursor = berkeley\_db.cursor()

                record = cursor.first()

                while record:

                    ticket\_id, ticket\_data = record

                    ticket\_fields = pickle.loads(ticket\_data)

                    if (self.current\_entity == "Patient" and list(ticket\_fields.values())[0] ==  int(selected\_item)) or \

                       (self.current\_entity == "Doctor" and list(ticket\_fields.values())[1] ==  int(selected\_item)):

                        berkeley\_db.delete(ticket\_id)

                    record = cursor.next()

                cursor.close()

                berkeley\_db.close()

            if self.current\_entity in ["Patient", "Diagnosis"]:

                berkeley\_db = db.DB()

                berkeley\_db.open(f"illnesses.db", None, db.DB\_HASH, db.DB\_CREATE)

                cursor = berkeley\_db.cursor()

                record = cursor.first()

                while record:

                    ticket\_id, ticket\_data = record

                    ticket\_fields = pickle.loads(ticket\_data)

                    if (self.current\_entity == "Patient" and list(ticket\_fields.values())[0] ==  int(selected\_item)) or \

                       (self.current\_entity == "Diagnosis" and list(ticket\_fields.values())[1] ==  int(selected\_item)):

                        berkeley\_db.delete(ticket\_id)

                    record = cursor.next()

                cursor.close()

                berkeley\_db.close()

            self.load\_data()

        except Exception as e:

            messagebox.showerror("Error", f"Failed to delete record: {e}")

    def fill\_fields\_on\_row\_select(self, event):

*# Get the selected row from the table*

        selected\_item = self.table.selection()

        if not selected\_item:

            return  *# If no row is selected, do nothing*

        try:

*# Retrieve the values of the selected row*

            row\_values = self.table.item(selected\_item, "values")

*# Populate the entry fields with the corresponding row values*

            for i, field in enumerate(self.entry\_fields):

*# Clear existing content in the entry field*

                field.delete(0, tk.END)

*# Insert the value from the selected row*

                field.insert(0, row\_values[i])

        except Exception as e:

            messagebox.showerror("Error", f"Failed to populate fields: {e}")

    def update\_record(self):

        selected\_item = self.table.focus()

        if not selected\_item:

            messagebox.showwarning("Warning", "Please select a record to update.")

            return

        try:

            record\_id = selected\_item

            berkeley\_db = db.DB()

            berkeley\_db.open(self.db\_filename, None, db.DB\_HASH, db.DB\_CREATE)

            updated\_record = {}

            for i, field in enumerate(self.entry\_fields):

                updated\_record[self.table["columns"][i]] = field.get()

            berkeley\_db.put(str(record\_id).encode('utf-8'), pickle.dumps(updated\_record))

            berkeley\_db.close()

            self.load\_data()

        except Exception as e:

            messagebox.showerror("Error", f"Failed to update record: {e}")

    def convert\_all(self):

        try:

            pg\_conn = psycopg2.connect(\*\*self.pg\_conn\_params)

            pg\_cursor = pg\_conn.cursor()

            for entity, db\_filename in self.entities.items():

                table\_name = entity.lower().replace(" ", "\_")

                pg\_cursor.execute(f"SELECT \* FROM {table\_name};")

                rows = pg\_cursor.fetchall()

                columns = [desc[0] for desc in pg\_cursor.description]

                berkeley\_db = db.DB()

                berkeley\_db.open(db\_filename, None, db.DB\_HASH, db.DB\_CREATE)

                for row in rows:

                    primary\_key = row[0]  *# Предполагаем, что первичный ключ — первый столбец*

*# Исключаем первичный ключ из value*

                    value = {columns[i]: row[i] for i in range(1, len(columns))}

                    serialized\_value = pickle.dumps(value)

                    berkeley\_db.put(str(primary\_key).encode(), serialized\_value)

                berkeley\_db.close()

            pg\_cursor.close()

            pg\_conn.close()

            messagebox.showinfo("Success", "All tables have been converted successfully.")

        except Exception as e:

            messagebox.showerror("Error", f"Failed to convert tables: {e}")

    def search\_records(self, event):

        search\_term = self.search\_entry.get().lower()

*# First, clear all previous highlights*

        self.table.tag\_configure("match", background="lightgreen")

        self.table.tag\_configure("no\_match", background="white")

*# Iterate through the rows and find matching ones*

        for row in self.table.get\_children():

            row\_values = self.table.item(row, "values")

            match = False

            for value in row\_values:

                if search\_term in str(value).lower():

                    match = True

                    break

            if match:

                self.table.item(row, tags=("match",))

            else:

                self.table.item(row, tags=("no\_match",))

*# Optional: Scroll to the first match*

        for row in self.table.get\_children():

            if "match" in self.table.item(row, "tags"):

                self.table.see(row)  *# Scroll to the matched row*

                break  *# Stop after the first match*

if \_\_name\_\_ == "\_\_main\_\_":

    root = tk.Tk()

    app = BerkeleyDBApp(root)

    root.mainloop()