

Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

ОТЧЕТ
по лабораторной работе №3
на тему

Освоение прикладного интерфейса СУБД BerkeleyDB. Разработка
конвертора базы данных PostgreSQL в набор баз данных Berkeley DB.
Адаптация спецификаций приложения

Вариант 3 (Аэропорт)

Студент:

И.И. Божко

Преподаватель:

Ю.Ю. Желтко

МИНСК 2024

1 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.

2 ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. **Извлечение схемы и данных из PostgreSQL:**
 1. Используя подключение к PostgreSQL, извлеките структуру (схему) и данные из таблиц базы данных PostgreSQL. Для этого используйте SQL-запросы и выполните сериализацию данных.
2. **Конвертация данных в Berkeley DB:**
 1. Для каждой таблицы в PostgreSQL создайте соответствующую базу данных Berkeley DB.
 2. Используйте первичные ключи таблиц PostgreSQL в качестве ключей для Berkeley DB, а значения столбцов — в виде сериализованных структур
3. **Запись данных в Berkeley DB:**
 1. Реализуйте запись данных в формате ключ-значение в Berkeley DB. Для этого используйте соответствующие функции библиотеки Berkeley DB.
4. **Адаптация спецификаций приложения:**
 1. Проанализируйте спецификации приложения, которые ранее работали с PostgreSQL, и адаптируйте их для работы с Berkeley DB.
 2. Убедитесь, что операции вставки, обновления, удаления и поиска данных выполняются корректно с использованием нового формата хранения (ключ-значение).

3 СТРУКТУРА ДАННЫХ

3.1 Модель данных в postgresql

Модель данных “Аэропорт” содержит сущности “Airline” — авиакомпания, “Destination” — направление, “Flight” — рейс, “Airplane” — самолёт, “Passenger” — пассажир.

В таблице 3.1 представлено описание сущности “Airline”.

Таблица 3.1 – Описание сущности “Airline”.

| Название поля | Описание поля | Ключ |
|---------------|-------------------|----------------|
| code | Код компании | Первичный ключ |
| name | Название компании | — |
| country | Страна | — |

В таблице 3.2 представлено описание сущности “Destination”.

Таблица 3.2 – Описание сущности “Destination”.

| Название поля | Описание поля | Ключ |
|---------------|--------------------|----------------|
| code | Код аэропорта | Первичный ключ |
| name | Название аэропорта | — |
| country | Страна | — |

В таблице 3.3 представлено описание сущности “Destination”.

Таблица 3.3 – Описание сущности “Flight”.

| Название поля | Описание поля | Ключ |
|----------------|-------------------|----------------|
| flight_number | Номер рейса | Первичный ключ |
| airline_code | Код авиакомпании | Внешний ключ 1 |
| airplane_code | Код аэропорта | Внешний ключ 2 |
| days | Дни недели | — |
| arrival_time | Время прибытия | — |
| departure_time | Время отправления | — |

В таблице 3.4 представлено описание сущности “Airplane”.

Таблица 3.4 – Описание сущности “Airplane”.

| Название поля | Описание поля | Ключ |
|---------------|----------------------------|----------------|
| code | Код самолёта | Первичный ключ |
| name | Название самолёта | — |
| manufacturer | Производитель | — |
| capacity | Вместимость | — |
| width | Фюзеляж (широкий/узкий) | — |

В таблице 3.5 представлено описание сущности “Passenger”.

Таблица 3.5 – Описание сущности “Passenger”.

| Название поля | Описание поля | Ключ |
|-----------------|----------------|----------------|
| passport_number | Номер паспорта | Первичный ключ |
| first_name | Имя | — |
| last_name | Фамилия | — |
| email | Е-mail адрес | — |

В таблице 3.6 представлено описание связей в модели данных.

Таблица 3.6 – Описание связей.

| Название связи | Связываемые таблицы | Промежуточная таблица |
|---------------------------------|---------------------|-----------------------|
| Авиакомпания, выполняющая рейс | Рейс, Авиакомпания | — |
| Самолёт, выполняющий рейс | Рейс, Самолёт | — |
| Направление (направления) рейса | Рейс, Направление | Flights-Destinations |
| Пассажир (пассажиры) рейса | Рейс, Пассажир | Flights-Passengers |

3.2 Хранение данных в berkeleydb

Каждая таблица postgresql преобразуется в отдельную базу данных. В качестве ключей баз данных используются первичные ключи postgresql.

4 ПРИМЕРЫ SQL-ЗАПРОСОВ ДЛЯ ИЗВЛЕЧЕНИЯ СХЕМЫ И ДАННЫХ ИЗ POSTGRESQL

Пример запроса для получения названий столбцов таблицы:

```
SELECT column_name
FROM information_schema.columns
WHERE table_name = '{table_name}';
```

Пример запроса для получения первичных ключей таблицы:

```
SELECT kcu.column_name
FROM information_schema.table_constraints tco
JOIN information_schema.key_column_usage kcu
ON kcu.constraint_name = tco.constraint_name
WHERE tco.constraint_type = 'PRIMARY KEY' AND
tco.table_name = '{table_name}';
```

5 КОД КОНВЕРТОРА ДАННЫХ

Для сериализации данных получают названия таблиц, столбцов и первичных ключей каждой таблицы. Первичный ключ обязательно переносится в начало таблицы. Затем для каждой таблицы создаётся JSON файл, содержащий все данные из таблицы.

Код для сериализации данных:

```
import psycopg2
import datetime
import json
import os

def get_all_tables(cursor):
    try:
        cursor.execute("""
            SELECT table_name
            FROM information_schema.tables
            WHERE table_schema = 'postgres';
        """)
        tables = cursor.fetchall()
        return [table[0] for table in tables]
    except Exception as e:
        print(f"Error getting tables: {e}")
        return []

def get_primary_keys(cursor, table_name):
    try:
        cursor.execute(f"""
            SELECT kcu.column_name
            FROM information_schema.table_constraints tco
            JOIN information_schema.key_column_usage kcu
            ON kcu.constraint_name = tco.constraint_name
            WHERE tco.constraint_type = 'PRIMARY KEY' AND tco.table_name
            = '{table_name}';
        """)
        primary_keys = cursor.fetchall()
        return [pk[0] for pk in primary_keys]
    except Exception as e:
```

```

        print(f"Error getting primary keys for {table_name}: {e}")
        return []

def get_column_names(cursor, table_name):
    try:
        cursor.execute(f"""
            SELECT column_name
            FROM information_schema.columns
            WHERE table_name = '{table_name}';
        """)
        columns = cursor.fetchall()
        return [column[0] for column in columns]
    except Exception as e:
        print(f"Error getting column names for {table_name}: {e}")
        return []

import datetime

def get_all_data(cursor, table_name, cnfk):
    try:
        cursor.execute(f"SELECT * FROM {table_name};")
        data = cursor.fetchall()
        column_names = [description[0] for description in
cursor.description]
        cnfk_index = column_names.index(cnfk)
        rearranged_column_names = [column_names[cnfk_index]] + [
            column_names[i] for i in range(len(column_names)) if i !=
cnfk_index
        ]
        filtered_data = []
        for row in data:
            new_row = (row[cnfk_index],) + tuple(
                item.strftime('%H:%M:%S') if isinstance(item,
datetime.time) else item
                for i, item in enumerate(row) if i != cnfk_index
            )
            filtered_data.append(new_row)
        return rearranged_column_names, filtered_data
    except Exception as e:

```

```

        print(f"Error getting data from {table_name}: {e}")
        return [], []

def create_json_from_tuples(tuples_array, file_path, column_names):
    data_list = [
        {column_names[i]: value for i, value in enumerate(tup)}
        for tup in tuples_array
    ]
    with open(file_path, 'w') as json_file:
        json.dump(data_list, json_file, indent=4)

def main():
    conn = None
    try:
        conn = psycopg2.connect(host='localhost', database='newairport',
                                user='postgres', password='admin')
    except Exception as e:
        print(f"Error connecting to database: {e}")
        return

    if conn is not None:
        cursor = conn.cursor()
        tables = get_all_tables(cursor)
        print("Tables in the database:", tables)
        if tables:
            try:
                os.makedirs('./dumps', exist_ok=True)
            except Exception as e:
                pass
            for table_name in tables:
                columns = get_column_names(cursor, table_name)
                primary_key = get_primary_keys(cursor, table_name)
                columns, data = get_all_data(cursor, table_name,
                primary_key[0])
                print(f"Data from {table_name}:", columns, data)
                create_json_from_tuples(data,
                f'dumps/{table_name}.json', columns)
            conn.close()

```

```

if __name__ == "__main__":
    main()

```

Для десериализации данных получаются данные из JSON файлов и для каждого файла создаётся база данных, в качестве ключа используется первое значение из json.

Код для десериализации данных:

```

import os
import json
import bsddb3

def process_json_files(directory):
    for filename in os.listdir(directory):
        base = os.path.splitext(filename)[0]
        new_filename = f"./databases/{base}.db"
        print(f'{filename} -> {new_filename}')
        try:
            os.makedirs('./databases', exist_ok=True)
        except Exception as e:
            pass
        if filename.endswith('.json'):
            file_path = os.path.join(directory, filename)
            db = bsddb3.db.DB()
            db.open(new_filename, None, bsddb3.db.DB_BTREE,
bsddb3.db.DB_CREATE)
            try:
                with open(file_path, 'r') as json_file:
                    data = json.load(json_file)
                    if isinstance(data, list):
                        for item in data:
                            if isinstance(item, dict):
                                key =
f'{list(item.values())[0]}.encode('utf-8')
                                value_data = {k: item[k] for k in item
if k != list(item.keys())[0]}
                                value_structure =
json.dumps(value_data).encode("utf-8")
                                print(key, value_structure)
                                db.put(key, value_structure)
                            else:
                                print("Item is not a dictionary:", item)
                        else:
                            print(f"Data in {filename} is not a list:
{data}")

```



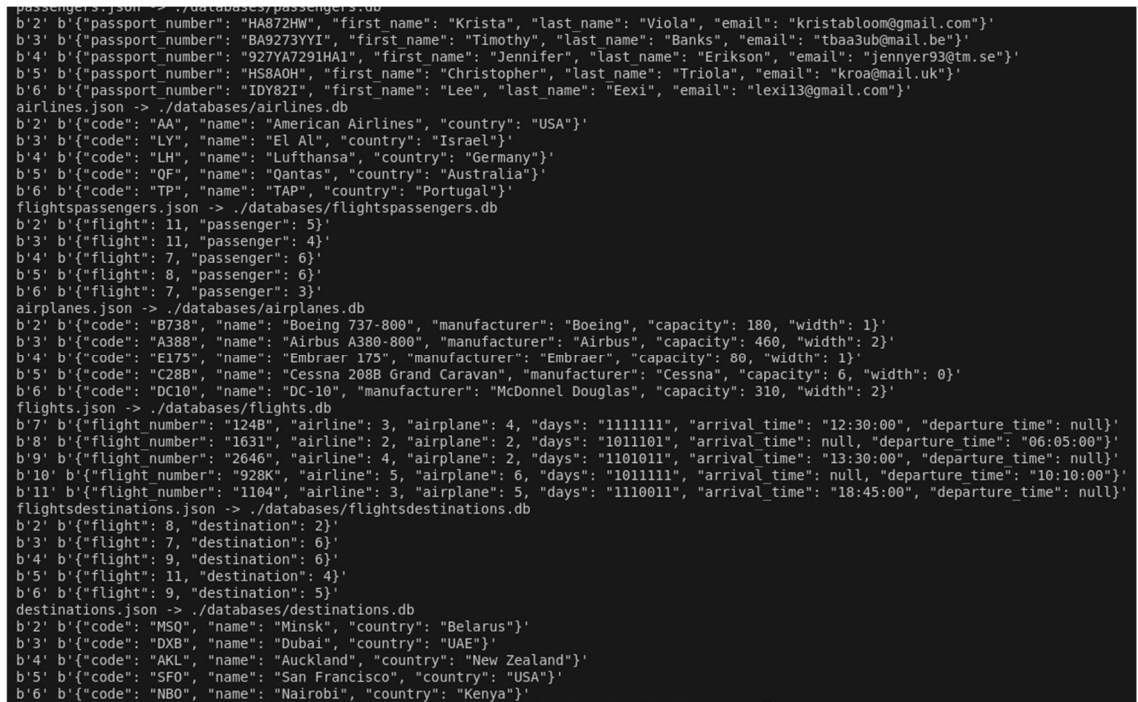
```

except Exception as e:
    print(f"Error processing file {filename}: {e}")
db.close()

directory_path = './dumps'
process_json_files(directory_path)

```

На рисунке 5.1 показан процесс десериализации данных.



```

passengers.json -> ./databases/passengers.db
b'2' b'{"passport_number": "HA872HW", "first_name": "Krista", "last_name": "Viola", "email": "kristabloom@gmail.com"}'
b'3' b'{"passport_number": "BA9273YYI", "first_name": "Timothy", "last_name": "Banks", "email": "tbba3ub@mail.be"}'
b'4' b'{"passport_number": "927YA7291HA1", "first_name": "Jennifer", "last_name": "Erikson", "email": "jennyer93@tm.se"}'
b'5' b'{"passport_number": "HS8A0H", "first_name": "Christopher", "last_name": "Triola", "email": "kroa@mail.uk"}'
b'6' b'{"passport_number": "IDY82I", "first_name": "Lee", "last_name": "Eexi", "email": "lexil3@gmail.com"}'
airlines.json -> ./databases/airlines.db
b'2' b'{"code": "AA", "name": "American Airlines", "country": "USA"}'
b'3' b'{"code": "LY", "name": "EL AL", "country": "Israel"}'
b'4' b'{"code": "LH", "name": "Lufthansa", "country": "Germany"}'
b'5' b'{"code": "QF", "name": "Qantas", "country": "Australia"}'
b'6' b'{"code": "TP", "name": "TAP", "country": "Portugal"}'
flightspassengers.json -> ./databases/flightspassengers.db
b'2' b'{"flight": 11, "passenger": 5}'
b'3' b'{"flight": 11, "passenger": 4}'
b'4' b'{"flight": 7, "passenger": 6}'
b'5' b'{"flight": 8, "passenger": 6}'
b'6' b'{"flight": 7, "passenger": 3}'
airplanes.json -> ./databases/airplanes.db
b'2' b'{"code": "B738", "name": "Boeing 737-800", "manufacturer": "Boeing", "capacity": 180, "width": 1}'
b'3' b'{"code": "A388", "name": "Airbus A380-800", "manufacturer": "Airbus", "capacity": 460, "width": 2}'
b'4' b'{"code": "E175", "name": "Embraer 175", "manufacturer": "Embraer", "capacity": 80, "width": 1}'
b'5' b'{"code": "C28B", "name": "Cessna 208B Grand Caravan", "manufacturer": "Cessna", "capacity": 6, "width": 0}'
b'6' b'{"code": "DC10", "name": "DC-10", "manufacturer": "McDonnell Douglas", "capacity": 310, "width": 2}'
flights.json -> ./databases/flights.db
b'7' b'{"flight_number": "124B", "airline": 3, "airplane": 4, "days": "1111111", "arrival_time": "12:30:00", "departure_time": null}'
b'8' b'{"flight_number": "1631", "airline": 2, "airplane": 2, "days": "1011101", "arrival_time": null, "departure_time": "06:05:00"}'
b'9' b'{"flight_number": "2646", "airline": 4, "airplane": 2, "days": "1101011", "arrival_time": "13:30:00", "departure_time": null}'
b'10' b'{"flight_number": "928K", "airline": 5, "airplane": 6, "days": "1011111", "arrival_time": null, "departure_time": "10:10:00"}'
b'11' b'{"flight_number": "1104", "airline": 3, "airplane": 5, "days": "1110011", "arrival_time": "18:45:00", "departure_time": null}'
flightsdestinations.json -> ./databases/flightsdestinations.db
b'2' b'{"flight": 8, "destination": 2}'
b'3' b'{"flight": 7, "destination": 6}'
b'4' b'{"flight": 9, "destination": 6}'
b'5' b'{"flight": 11, "destination": 4}'
b'6' b'{"flight": 9, "destination": 5}'
destinations.json -> ./databases/destinations.db
b'2' b'{"code": "MSQ", "name": "Minsk", "country": "Belarus"}'
b'3' b'{"code": "DXB", "name": "Dubai", "country": "UAE"}'
b'4' b'{"code": "AKL", "name": "Auckland", "country": "New Zealand"}'
b'5' b'{"code": "SFO", "name": "San Francisco", "country": "USA"}'
b'6' b'{"code": "NBO", "name": "Nairobi", "country": "Kenya"}'

```

Рисунок 5.1 – Десериализация данных

Для просмотра содержимого баз данных используется следующий код:

```

import bsddb3
import os

def print_all_records(db_path):
    try:
        db = bsddb3.btopen(db_path, 'r')
    except Exception as e:
        print(f"Error opening database: {e}")
        return

    try:
        for key in db.keys():

```

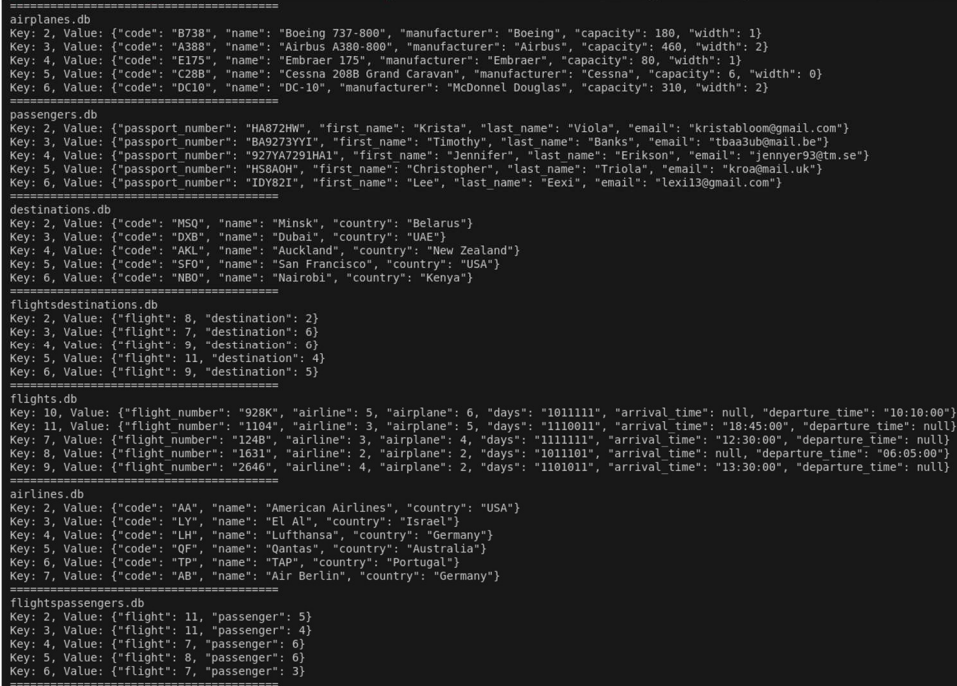
```

        value = db[key]
        key = key.decode("utf-8")
        value = value.decode("utf-8")
        print(f"Key: {key}, Value: {value}")
    except Exception as e:
        print(f"Error reading records: {e}")
    finally:
        db.close()

if __name__ == "__main__":
    print('=====')
    for filename in os.listdir('./databases'):
        print(filename)
        print_all_records(f'./databases/{filename}')
    print('=====')

```

На рисунке 5.2 показан процесс просмотра содержимого баз данных.



```

airplanes.db
Key: 2, Value: {'code': 'B738', 'name': 'Boeing 737-800', 'manufacturer': 'Boeing', 'capacity': 180, 'width': 1}
Key: 3, Value: {'code': 'A388', 'name': 'Airbus A380-800', 'manufacturer': 'Airbus', 'capacity': 460, 'width': 2}
Key: 4, Value: {'code': 'E175', 'name': 'Embraer 175', 'manufacturer': 'Embraer', 'capacity': 80, 'width': 1}
Key: 5, Value: {'code': 'C28B', 'name': 'Cessna 208B Grand Caravan', 'manufacturer': 'Cessna', 'capacity': 6, 'width': 0}
Key: 6, Value: {'code': 'DC10', 'name': 'DC-10', 'manufacturer': 'McDonnell Douglas', 'capacity': 310, 'width': 2}
=====
passengers.db
Key: 2, Value: {'passport_number': 'HA872HW', 'first_name': 'Krista', 'last_name': 'Viola', 'email': 'kristabloom@gmail.com'}
Key: 3, Value: {'passport_number': 'BA9273YVI', 'first_name': 'Timothy', 'last_name': 'Banks', 'email': 'tbaa3ub@gmail.be'}
Key: 4, Value: {'passport_number': '927YA7291HA1', 'first_name': 'Jennifer', 'last_name': 'Erikson', 'email': 'jenmyer99@tm.se'}
Key: 5, Value: {'passport_number': 'HS8A0H', 'first_name': 'Christopher', 'last_name': 'Triola', 'email': 'kroa@mail.uk'}
Key: 6, Value: {'passport_number': 'IDY82I', 'first_name': 'Lee', 'last_name': 'Eexi', 'email': 'lexil3@gmail.com'}
=====
destinations.db
Key: 2, Value: {'code': 'MSQ', 'name': 'Minsk', 'country': 'Belarus'}
Key: 3, Value: {'code': 'DXB', 'name': 'Dubai', 'country': 'UAE'}
Key: 4, Value: {'code': 'AKL', 'name': 'Auckland', 'country': 'New Zealand'}
Key: 5, Value: {'code': 'SFO', 'name': 'San Francisco', 'country': 'USA'}
Key: 6, Value: {'code': 'NBO', 'name': 'Nairobi', 'country': 'Kenya'}
=====
flightsdestinations.db
Key: 2, Value: {'flight': 0, 'destination': 2}
Key: 3, Value: {'flight': 7, 'destination': 6}
Key: 4, Value: {'flight': 9, 'destination': 6}
Key: 5, Value: {'flight': 11, 'destination': 4}
Key: 6, Value: {'flight': 9, 'destination': 5}
=====
flights.db
Key: 10, Value: {'flight_number': '928K', 'airline': 5, 'airplane': 6, 'days': '1011111', 'arrival_time': null, 'departure_time': '10:10:00'}
Key: 11, Value: {'flight_number': '1104', 'airline': 3, 'airplane': 5, 'days': '1110011', 'arrival_time': '18:45:00', 'departure_time': null}
Key: 7, Value: {'flight_number': '124B', 'airline': 3, 'airplane': 4, 'days': '1111111', 'arrival_time': '12:30:00', 'departure_time': null}
Key: 8, Value: {'flight_number': '1631', 'airline': 2, 'airplane': 2, 'days': '1011101', 'arrival_time': null, 'departure_time': '06:05:00'}
Key: 9, Value: {'flight_number': '2646', 'airline': 4, 'airplane': 2, 'days': '1101011', 'arrival_time': '13:30:00', 'departure_time': null}
=====
airlines.db
Key: 2, Value: {'code': 'AA', 'name': 'American Airlines', 'country': 'USA'}
Key: 3, Value: {'code': 'LY', 'name': 'El Al', 'country': 'Israel'}
Key: 4, Value: {'code': 'LH', 'name': 'Lufthansa', 'country': 'Germany'}
Key: 5, Value: {'code': 'QF', 'name': 'Qantas', 'country': 'Australia'}
Key: 6, Value: {'code': 'TP', 'name': 'TAP', 'country': 'Portugal'}
Key: 7, Value: {'code': 'AB', 'name': 'Air Berlin', 'country': 'Germany'}
=====
flightspassengers.db
Key: 2, Value: {'flight': 11, 'passenger': 5}
Key: 3, Value: {'flight': 11, 'passenger': 4}
Key: 4, Value: {'flight': 7, 'passenger': 6}
Key: 5, Value: {'flight': 8, 'passenger': 6}
Key: 6, Value: {'flight': 7, 'passenger': 3}

```

Рисунок 5.2 – Просмотр данных

Для добавления данных используется следующий код:

```

import bsddb3
import os

```

```

maxkey = 1

def print_all_records(db_path):
    global maxkey
    try:
        db = bsddb3.btopen(db_path, 'r')
    except Exception as e:
        print(f"Error opening database: {e}")
        return
    try:
        for key in db.keys():
            value = db[key]
            key = key.decode("utf-8")
            ikey = int(key)
            value = value.decode("utf-8")
            print(f"Key: {key}, Value: {value}")
            if ikey > maxkey:
                maxkey = ikey
    except Exception as e:
        print(f"Error reading records: {e}")
    finally:
        db.close()

def add_new_record(db_path, data):
    global maxkey
    try:
        db = bsddb3.btopen(db_path, 'c')
    except Exception as e:
        print(f"Error opening database: {e}")
        return

```


```

try:
    key = f'{maxkey + 1}'.encode("utf-8")
    value = data.encode("utf-8")
    db[key] = value
except Exception as e:
    print(f"Error adding record: {e}")
finally:
    db.close()

if __name__ == "__main__":
    user_input = input("Enter table name: ")
    if os.path.exists(f'./databases/{user_input}.db'):
        print_all_records(f'./databases/{user_input}.db')
        print('=====')
        new_data = input("Enter new data: ")
        add_new_record(f'./databases/{user_input}.db', new_data)
        print('=====')
        print_all_records(f'./databases/{user_input}.db')
    else:
        print(f"The file ./databases/{user_input}.db does not exist.")

```

На рисунке 5.3 показан процесс добавления данных.



```

Enter table name: airlines
Key: 2, Value: {"code": "AA", "name": "American Airlines", "country": "USA"}
Key: 3, Value: {"code": "LY", "name": "El Al", "country": "Israel"}
Key: 4, Value: {"code": "LH", "name": "Lufthansa", "country": "Germany"}
Key: 5, Value: {"code": "QF", "name": "Qantas", "country": "Australia"}
Key: 6, Value: {"code": "TP", "name": "TAP", "country": "Portugal"}
=====
Enter new data: {"code": "AB", "name": "Air Berlin", "country": "Germany"}
=====
Key: 2, Value: {"code": "AA", "name": "American Airlines", "country": "USA"}
Key: 3, Value: {"code": "LY", "name": "El Al", "country": "Israel"}
Key: 4, Value: {"code": "LH", "name": "Lufthansa", "country": "Germany"}
Key: 5, Value: {"code": "QF", "name": "Qantas", "country": "Australia"}
Key: 6, Value: {"code": "TP", "name": "TAP", "country": "Portugal"}
Key: 7, Value: {"code": "AB", "name": "Air Berlin", "country": "Germany"}

```

Рисунок 5.3 – Добавление данных

Для удаления данных используется следующий код:

```
import bsddb3
import os

maxkey = 1

def print_all_records(db_path):
    global maxkey
    try:
        db = bsddb3.btopen(db_path, 'r')
    except Exception as e:
        print(f"Error opening database: {e}")
        return
    try:
        for key in db.keys():
            value = db[key]
            key = key.decode("utf-8")
            ikey = int(key)
            value = value.decode("utf-8")
            print(f"Key: {key}, Value: {value}")
            if ikey > maxkey:
                maxkey = ikey
    except Exception as e:
        print(f"Error reading records: {e}")
    finally:
        db.close()

def delete_record(db_path, orkey):
    global maxkey
    k = None
```

```

try:
    db = bsddb3.btopen(db_path, 'c')
except Exception as e:
    print(f"Error opening database: {e}")
    return

try:
    k = f'{orkey}'.encode("utf-8")
    value = db[k]
except Exception as e:
    print(f"Record with key '{orkey}' does not exist.")
    return

try:
    del db[k]
except Exception as e:
    print(f"Error editing record: {e}")

finally:
    db.close()

if __name__ == "__main__":
    user_input = input("Enter table name: ")
    if os.path.exists(f'./databases/{user_input}.db'):
        print_all_records(f'./databases/{user_input}.db')
        print('=====')
        orig_key = input("Enter the key: ")
        delete_record(f'./databases/{user_input}.db', orig_key)
        print('=====')
        print_all_records(f'./databases/{user_input}.db')
    else:
        print(f"The file ./databases/{user_input}.db does not exist.")

```

На рисунке 5.4 показан процесс удаления данных.

```
Enter table name: airlines
Key: 2, Value: {"code": "AA", "name": "American Airlines", "country": "USA"}
Key: 3, Value: {"code": "LY", "name": "El Al", "country": "Israel"}
Key: 4, Value: {"code": "LH", "name": "Lufthansa", "country": "Germany"}
Key: 5, Value: {"code": "QF", "name": "Qantas", "country": "Australia"}
Key: 6, Value: {"code": "LO", "name": "TAP", "country": "Portugal"}
Key: 7, Value: {"code": "AB", "name": "Air Berlin", "country": "Germany"}
=====
Enter the key: 4
=====
Key: 2, Value: {"code": "AA", "name": "American Airlines", "country": "USA"}
Key: 3, Value: {"code": "LY", "name": "El Al", "country": "Israel"}
Key: 5, Value: {"code": "QF", "name": "Qantas", "country": "Australia"}
Key: 6, Value: {"code": "LO", "name": "TAP", "country": "Portugal"}
Key: 7, Value: {"code": "AB", "name": "Air Berlin", "country": "Germany"}
=====
```

Рисунок 5.4 – Удаление данных

Для редактирования данных используется следующий код:

```
import bsddb3
import os

maxkey = 1

def print_all_records(db_path):
    global maxkey
    try:
        db = bsddb3.btopen(db_path, 'r')
    except Exception as e:
        print(f"Error opening database: {e}")
        return
    try:
        for key in db.keys():
            value = db[key]
            key = key.decode("utf-8")
            ikey = int(key)
            value = value.decode("utf-8")
            print(f"Key: {key}, Value: {value}")
            if ikey > maxkey:
                maxkey = ikey
    except Exception as e:
        print(f"Error reading records: {e}")
```

```

        finally:
            db.close()

def edit_record(db_path, data, orkey):
    global maxkey
    k = None
    try:
        db = bsddb3.btopen(db_path, 'c')
    except Exception as e:
        print(f"Error opening database: {e}")
        return
    try:
        k = f'{orkey}'.encode("utf-8")
        value = db[k]
    except Exception as e:
        print(f"Record with key '{orkey}' does not exist.")
        return
    try:
        value = data.encode("utf-8")
        db[k] = value
    except Exception as e:
        print(f"Error editing record: {e}")
    finally:
        db.close()

if __name__ == "__main__":
    user_input = input("Enter table name: ")
    if os.path.exists(f'./databases/{user_input}.db'):
        print_all_records(f'./databases/{user_input}.db')
        print('=====')
        orig_key = input("Enter the key: ")
        new_data = input("Enter new data: ")
        edit_record(f'./databases/{user_input}.db', new_data, orig_key)
        print('=====')
        print_all_records(f'./databases/{user_input}.db')
    else:
        print(f"The file ./databases/{user_input}.db does not exist.")

```


На рисунке 5.5 показан процесс редактирования данных.

```
Enter table name: airlines
Key: 2, Value: {"code": "AA", "name": "American Airlines", "country": "USA"}
Key: 3, Value: {"code": "LY", "name": "El Al", "country": "Israel"}
Key: 4, Value: {"code": "LH", "name": "Lufthansa", "country": "Germany"}
Key: 5, Value: {"code": "QF", "name": "Qantas", "country": "Australia"}
Key: 6, Value: {"code": "TP", "name": "TAP", "country": "Portugal"}
Key: 7, Value: {"code": "AB", "name": "Air Berlin", "country": "Germany"}
=====
Enter the key: 6
Enter new data: {"code": "LO", "name": "TAP", "country": "Portugal"}
=====
Key: 2, Value: {"code": "AA", "name": "American Airlines", "country": "USA"}
Key: 3, Value: {"code": "LY", "name": "El Al", "country": "Israel"}
Key: 4, Value: {"code": "LH", "name": "Lufthansa", "country": "Germany"}
Key: 5, Value: {"code": "QF", "name": "Qantas", "country": "Australia"}
Key: 6, Value: {"code": "LO", "name": "TAP", "country": "Portugal"}
Key: 7, Value: {"code": "AB", "name": "Air Berlin", "country": "Germany"}
```

Рисунок 5.5 – Редактирование данных

6 ИЗМЕНЕНИЯ СПЕЦИФИКАЦИИ ПРИЛОЖЕНИЯ

В связи с заменой postgresql на berkeleydb приложению необходимо выполнять дополнительные функции:

- 1) Контроль и генерация первичных ключей
- 2) Обработка внешних ключей
- 3) Валидация ввода
- 4) Приведение данных к единому формату

ЗАКЛЮЧЕНИЕ

В ходе лабораторной работы были выполнены ключевые шаги по преобразованию данных из реляционной модели PostgreSQL в формат ключ-значение, используемый в BerkeleyDB.

Были разработаны скрипты для сериализации и десериализации данных, просмотра, вставки, удаления, редактирования данных в BerkeleyDB.

Работа показала возможности и ограничения Berkeley DB в сравнении с PostgreSQL, а также расширила представление о выборе архитектуры хранения данных для разных типов приложений.