

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития
Кафедра инфокоммуникаций

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №6
дисциплины «анализ данных»

Выполнил:
Середа Кирилл Витальевич
2 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и
вычислительная техника», очная
форма обучения

(подпись)

Руководитель практики:
Воронкин Роман Александрович

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2024 г.

Тема: Основы работы с SQLite3

Цель работы: исследовать базовые возможности системы управления базами данных SQLite3.

Ход работы:

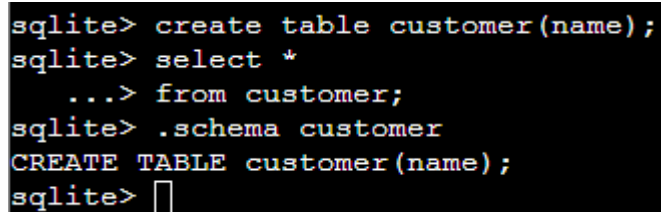
Задача 1 (рис.1):

create table customer(name);

select *

from customer;

.schema customer

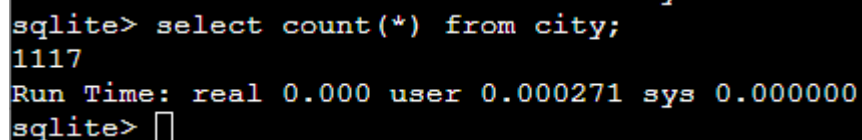


```
sqlite> create table customer(name);  
sqlite> select *  
...> from customer;  
sqlite> .schema customer  
CREATE TABLE customer(name);  
sqlite> 
```

Рисунок 1 – Результат выполнения первой задачи

Задача 2:

Решите задачу: с помощью команды .help найдите в песочнице команду, которая отвечает за вывод времени выполнения запроса. Если ее включить, в результатах запроса добавится строка.

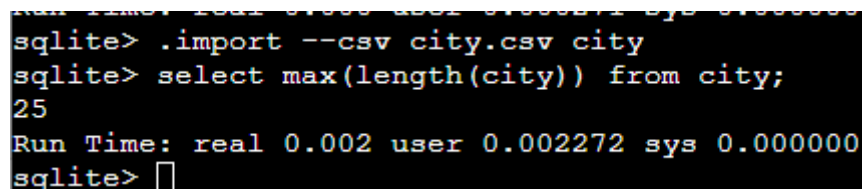


```
sqlite> select count(*) from city;  
1117  
Run Time: real 0.000 user 0.000271 sys 0.000000  
sqlite> 
```

Рисунок 2 – Результат выполнения второй задачи

Задача 3:

Решите задачу: загрузите файл city.csv в песочнице (рис. 3)



```
sqlite> .import --csv city.csv city  
sqlite> select max(length(city)) from city;  
25  
Run Time: real 0.002 user 0.002272 sys 0.000000  
sqlite> 
```

Рисунок 3 – Результат выполнения третьей задачи

Задача 4:

Решите задачу: загрузите файл city.csv в песочнице с помощью команды `.import` , но без использования опции `--csv` . Эта опция появилась только в недавней версии SQLite (3.32, май 2020), так что полезно знать способ, подходящий для старых версий. (рис. 4)

```
sqlite> .separator ,
sqlite> .import city.csv city;
sqlite> 
```

Рисунок 4 – Результат выполнения четвертой задачи

Задача 5:

Решите задачу: напишите в песочнице запрос, который посчитает количество городов для каждого часового пояса в Сибирском и Приволжском федеральных округах. Выведите столбцы `timezone` и `city_count` , отсортируйте по значению часового пояса (рис.5)

```
sqlite> .mode box
sqlite> SELECT
...> DISTINCT timezone AS timee,
...> COUNT() AS count_city
...> FROM city
...>
...> GROUP BY timee
...> ORDER BY timee DESC;
```

timee	count_city
timezone	1
UTC+9	62
UTC+8	56
UTC+7	172
UTC+6	12
UTC+5	346
UTC+4	132
UTC+3	1320
UTC+2	44
UTC+12	12
UTC+11	34
UTC+10	44

```
Run Time: real 0.003 user 0.003355 sys 0.000000
sqlite> 
```

Рисунок 5 – Результат выполнения пятой задачи

Задача 6: решите задачу: напишите в песочнице запрос, который найдет три ближайших к Самаре города, не считая саму Самару. (рис.6)

```

sqlite> .mode box
sqlite> WITH samara AS (SELECT geo_lat AS lat, geo_lon AS lon FROM city WHERE city="Самара")
...> SELECT
...> city,
...>   SORT(
...>     POW (geo_lon - lon, 2) + POWER(geo_lat - lat, 2)
...>   ) * 69.09 AS euclidean_distance
...> FROM
...>   city, samara
...> WHERE euclidean_distance > 0
...> ORDER BY euclidean_distance
...> LIMIT 3;

```

city	euclidean_distance
Новокуйбышевск	12.8298063265514
Чапаевск	24.7389598092284
Кинель	36.4840951529617

Рисунок 6 – Результат выполнения шестой задачи

Задача 7:

Решите задачу: напишите в песочнице запрос, который посчитает количество городов в каждом часовом поясе. (рис.7)

```

sqlite> .mode csv
sqlite> .headers ON
sqlite> .separator |
sqlite> .mode box
sqlite> SELECT
...>   DISTINCT timezone AS timee,
...>   COUNT() AS count_city
...> FROM city
...>
...> GROUP BY timee
...> ORDER BY timee DESC;

```

timee	count_city
UTC+9	31
UTC+8	28
UTC+7	86
UTC+6	6
UTC+5	173
UTC+4	66
UTC+3	660
UTC+2	22
UTC+12	6
UTC+11	17
UTC+10	22

Рисунок 7 - Результат выполнения седьмой задачи

Индивидуальное задание: загрузите в SQLite выбранный вами датасет в формате CSV (Kaggle). Сформируйте более пяти запросов к таблицам БД. Выгрузите результат выполнения запросов в форматы CSV и JSON.

```
sqlite> .output z1.json
sqlite> .schema blood
sqlite> .mode csv
sqlite> .output z1.csv
sqlite> .schema blood
sqlite> .output z2.json
sqlite> .mode json
sqlite> SELECT * FROM blood;
sqlite> .mode csv
sqlite> .output z2.csv
sqlite> SELECT * FROM blood;
sqlite> .output z3.csv
sqlite> SELECT * FROM blood limit 4;
sqlite> .output z4.csv
sqlite> SELECT * FROM blood limit 10;
sqlite> .output z5.csv
sqlite> SELECT * FROM blood limit 15;
sqlite> .mode json
sqlite> .output z3.json
sqlite> SELECT * FROM blood limit 4;
sqlite> .output z4.json
sqlite> SELECT * FROM blood limit 10;
sqlite> .output z5.json
sqlite> SELECT * FROM blood limit 15;
sqlite>
```

Рисунок 8 – Индивидуальное задание

Вывод: в ходе практической работы были исследованы базовые возможности системы управления базами данных SQLite3

Контрольные вопросы:

1. Назначение реляционных баз данных и СУБД

Реляционные базы данных предназначены для хранения и управления данными в структурированном формате, используя таблицы, ключи и отношения между данными. СУБД (системы управления базами данных) обеспечивают доступ к этим данным, их обновление, а также выполнение запросов и аналитику.

2. Назначение языка SQL

Язык SQL (Structured Query Language) предназначен для управления данными в реляционных базах данных. Он используется для создания, изменения и управления данными в таблицах, выполнения запросов, агрегации данных и манипуляций с базой данных.

3. Состав языка SQL

Язык SQL состоит из различных элементов, включая операторы для создания и управления таблицами, операторы для выполнения запросов (например, SELECT), операторы для вставки, обновления и удаления данных, а также операторы для управления правами доступа к данным.

4. Отличие СУБД SQLite от клиент-серверных СУБД

СУБД SQLite отличается от клиент-серверных СУБД тем, что она представляет собой встроенную базу данных, которая не требует отдельного сервера для своей работы. Это означает, что SQLite работает непосредственно с файлами на устройстве, в то время как клиент-серверные СУБД требуют наличия отдельного сервера для управления данными.

5. Установка SQLite в Windows и Linux

Для установки SQLite в Windows и Linux можно воспользоваться официальным сайтом SQLite и загрузить соответствующие установочные файлы. Для Windows это может быть исполняемый файл, а для Linux - установка через пакетный менеджер операционной системы.

6. Создание базы данных SQLite

Для создания базы данных SQLite можно воспользоваться командой `sqlite3` в командной строке. Например, для создания базы данных с именем "mydatabase.db" можно использовать следующую команду:

```
sqlite3 mydatabase.db
```

После этого можно создавать таблицы и вставлять данные в эту базу данных.

7. Определение текущей базы данных в SQLite

В SQLite можно определить текущую базу данных, выполнив следующую команду:

```
SELECT db_name()
```

Это позволит узнать, какая база данных в данный момент является текущей.

8. Создание и удаление таблицы в SQLite

Для создания таблицы в SQLite используется оператор `CREATE TABLE`, а для удаления - оператор `DROP TABLE`. Например:

```
CREATE TABLE mytable (  
    id INTEGER PRIMARY KEY,  
    name TEXT
```

```
);  
DROP TABLE mytable;
```

Эти операторы позволяют создавать и удалять таблицы в базе данных SQLite.

9. Первичный ключ в таблице

Первичный ключ в таблице является уникальным идентификатором каждой записи в таблице. Он позволяет однозначно идентифицировать каждую запись и обеспечивает уникальность данных в таблице.

10. Сделание первичного ключа таблицы автоинкрементным

Для того чтобы сделать первичный ключ таблицы автоинкрементным в SQLite, можно использовать ключевое слово `AUTOINCREMENT` при определении поля, которое будет являться первичным ключом. Например:

```
CREATE TABLE mytable (  
    id INTEGER PRIMARY KEY AUTOINCREMENT,  
    name TEXT  
);
```

Это позволит автоматически увеличивать значение первичного ключа при вставке новых записей.

11. Назначение инструкций NOT и DEFAULT при создании таблиц

При создании таблицы в SQLite, инструкция `NOT NULL` указывает, что поле не может содержать значение `NULL`, тогда как инструкция `DEFAULT` позволяет задать значение по умолчанию для поля, если при вставке записи значение не указано явно.

12. Назначение внешних ключей в таблице и их создание

Внешние ключи в таблице используются для установления связи между данными в разных таблицах. Они обеспечивают целостность данных и связывают записи в одной таблице с записями в другой. Для создания внешнего ключа в таблице SQLite можно использовать следующий синтаксис:

```
CREATE TABLE orders (
```

```
order_id INTEGER PRIMARY KEY,  
product_id INTEGER,  
FOREIGN KEY (product_id) REFERENCES products (product_id)  
);
```

Это создаст внешний ключ `product_id`, который ссылается на поле `product_id` в таблице `products`.

13. Вставка строки в таблицу базы данных SQLite

Для выполнения вставки строки в таблицу базы данных SQLite используется оператор `INSERT INTO`. Например:

```
INSERT INTO mytable (column1, column2, ...)  
VALUES (value1, value2, ...);
```

Это позволит вставить новую запись в таблицу `mytable` с указанными значениями.

14. Выбор данных из таблицы SQLite

Для выборки данных из таблицы SQLite используется оператор `SELECT`. Например:

```
SELECT column1, column2, ...  
FROM mytable;
```

Этот оператор позволяет выбрать определенные столбцы из таблицы `mytable`.

15. Ограничение выборки данных с помощью условия WHERE

Для ограничения выборки данных с помощью условия `WHERE` в SQLite используется следующий синтаксис:

```
SELECT column1, column2, ...  
FROM mytable  
WHERE condition;
```

Это позволяет выбрать только те строки, которые удовлетворяют указанному условию.

16. Упорядочивание выбранных данных

Для упорядочивания выбранных данных в SQLite используется оператор ORDER BY. Например:

```
SELECT column1, column2, ...  
FROM mytable  
ORDER BY column1 ASC;
```

Это позволяет упорядочить выбранные данные по значению столбца column1 в порядке возрастания (ASC) или убывания (DESC).

17. Обновление записей в таблице SQLite

Для обновления записей в таблице SQLite используется оператор UPDATE. Например:

```
UPDATE mytable  
SET column1 = value1, column2 = value2, ...  
WHERE condition;
```

Это позволяет обновить значения указанных столбцов в таблице mytable для строк, которые удовлетворяют указанному условию.

18. Удаление записей из таблицы SQLite

Для удаления записей из таблицы SQLite используется оператор DELETE. Например:

```
DELETE FROM mytable  
WHERE condition;
```

Это позволяет удалить строки из таблицы mytable, которые удовлетворяют указанному условию.

19. Группировка данных из выборки из таблицы SQLite

Для группировки данных из выборки из таблицы SQLite используется оператор GROUP BY. Например:

```
SELECT column1, aggregate_function(column2)  
FROM mytable  
GROUP BY column1;
```

Это позволяет сгруппировать данные по значению столбца column1 и применить агрегатные функции к данным в каждой группе.

20. Получение значения агрегатной функции в выборке из таблицы SQLite

Для получения значения агрегатной функции (например, минимум, максимум, количество записей и т. д.) в выборке из таблицы SQLite используются функции, такие как COUNT, MAX, MIN, SUM и AVG. Например:

```
SELECT COUNT(*)  
FROM mytable;
```

Это позволяет получить количество записей в таблице mytable. Аналогично, можно использовать другие агрегатные функции для получения различных статистических данных из таблицы.

21. Выполнение объединения нескольких таблиц в операторе SELECT

Для выполнения объединения нескольких таблиц в операторе SELECT в SQLite используются операторы JOIN. Например:

```
SELECT *  
FROM table1  
INNER JOIN table2 ON table1.column = table2.column;
```

Это позволяет объединить данные из двух таблиц на основе определенного условия.

22. Назначение подзапросов и шаблонов при работе с таблицами SQLite

Подзапросы в SQLite позволяют включать один запрос внутри другого запроса. Они могут использоваться для выполнения более сложных операций выборки данных или для создания вычисляемых столбцов. Шаблоны в SQLite могут использоваться для создания структуры таблицы или для определения правил валидации данных.

23. Назначение представлений VIEW в SQLite

Представления VIEW в SQLite представляют собой виртуальные таблицы, которые можно использовать для упрощения сложных запросов или для скрытия деталей структуры базы данных от пользователей.

24. Средства для импорта данных в SQLite

Существует несколько средств для импорта данных в SQLite, такие как использование команды `.import` в интерфейсе командной строки SQLite, использование инструментов для работы с базами данных, таких как DB Browser for SQLite, а также программирование с использованием языков программирования, поддерживающих SQLite.

25. Назначение команды `.schema`

Команда `.schema` в SQLite используется для отображения схемы базы данных, включая определения таблиц, индексов, представлений и т. д. Это позволяет получить обзор структуры базы данных.

26. Экспорт данных из SQLite в форматы CSV и JSON

Для экспорта данных из SQLite в форматы CSV и JSON можно использовать инструменты командной строки, такие как `sqlite3` с командой `.mode` для установки режима вывода в CSV или JSON, а затем выполнить запрос и перенаправить вывод в файл.

27. Другие форматы для экспорта данных

Помимо CSV и JSON, данные из SQLite можно экспортировать в другие форматы, такие как XML, Excel, SQL-скрипты и т. д. Это можно сделать с помощью различных инструментов и методов, включая использование сторонних программ или библиотек для работы с данными.