Практика 6

Физическое расположение данных в PostgreSQL

**Цель работы:**

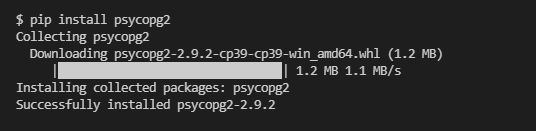
1. Научиться понимать, где физически находится база данных и таблицы.
2. Научиться работать с функциями размеров таблиц в PostgreSQL и определять размер таблицы.
3. Понять разницу между VACUUM и VACUUM FULL.
4. Ознакомиться с понятием TOAST

*Подготовка*

Чтобы иметь возможность оперировать данными в PostgreSQL, нам необходимо установить текущую версию psycopg2 для использования PostgreSQL в Python. С помощью команды pip можно установить модуль в любую ОС: Windows, macOS, Linux:

Для начала нужно установить текущую версию Psycopg2 для использования PostgreSQL в Python. С помощью команды pip можно установить модуль в любую операцию систему: Windows, macOS, Linux:

pip install psycopg2



**Часть 1**

Шаг 1. Подготовка БД

В PostgreSQL имеется уже предсозданная база данных ‘postgresdb’. Будем использовать её.

Шаг 2. Узнать расположение БД

Чтобы узнать, в каком файле находится база данных, выполняем команду

**show data\_directory;**

q = 'show data\_directory;'

cur.execute(q)

print(cur.fetchall())

В ответе увидим следующий путь: 'C:/Program Files/PostgreSQL/12/data'

Чтобы узнать размер БД, необходимо воспользоваться функцией

pg\_database\_size(db\_name),

где db\_name – название БД

В нашем коде это будет выглядеть так:

q = 'select pg\_database\_size(%s);'

cur.execute(q, ('postgresdb', ))

row = cur.fetchone()

print(str(row[0]))

Ответ получим в байтах:

**8209263**

**Часть 2**

Шаг 1. Создание таблицы

Создадим таблицу под названием ‘worker’. Синтаксис будет следующим:

CREATE TABLE worker (

    id BIGSERIAL NOT NULL PRIMARY KEY,

    first\_name text,

    age INTEGER

    );

*\*Примечание: следовало назвать таблицу с множественным значением ‘worker****s****’.*

Шаг 2. Посмотреть размер таблицы на момент создания

Узнаем, сколько занимает таблица ‘worker’. На этот раз будем использовать дополнительно функцию pg\_size\_pretty(), которая переводит байты в человекочитаемый формат

q = 'select pg\_size\_pretty(pg\_table\_size(%s));'

cur.execute(q, ('worker', ))

row = cur.fetchone()

print(str(row[0]))

В ответе получим:

8192 bytes

Заполним таблицу ‘worker’

Структуру нашей таблицы можно описать так:

**…**

age

first\_name

id

Где id – первичный ключ, позволяющий однозначно идентифицировать каждую запись в таблице,

first\_name - имя сотрудника [text]

age – его возраст [INTEGER]

Шаг 3. Заполнение таблицы

Итак, приступим к заполнению нашей таблицы, для этого выполним следующий блок кода:

q = (

    f"insert into worker ("

    f"id, first\_name, age"

    f")"

    f"select "

    f"i,"

    f"md5(random()::text),"

    f"random()\*10000+1"

    f"from generate\_series(1, 100000) i;"

    )

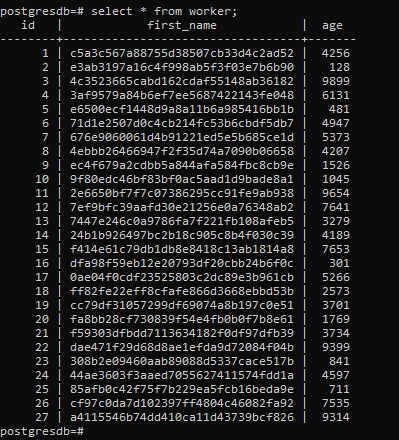
cur.execute(q)

con.commit() # сохраняем изменения

Для колонки ‘first\_name’ используем функцию random(), с преобразованием его в текстовый формат (::text) и последующим его вычислением MD5-хеш строки md5(), возвращающий результат в 16-ричном виде.

Функция же generate\_series(1, 100000) будет выдавать ряд значений от 1 до 100000 с шагом 1 (есть возможность использовать третий параметр для определения шага)

Чтобы проверить результат выполнения данного кода, проверим в SQL Shell:



Действительно, наша таблица теперь хранит порядка 100 тыс. записей

Произведём вышеупомянутые команды для подсчёта размера данных таблицы сейчас.

Получим следующий результат:

7512 kB

Шаг 4. Удаление данных и перерасчёт

Теперь очистим нашу таблицу с помощью команды TRUNCATE (удалить все данные)

q = 'TRUNCATE worker;'

cur.execute(q)

con.commit()

и произведём расчёт размера таблицы заново, получив ответ:

8192 bytes

Очевидно, размер данных вернулся к прежнему состоянию (когда БД была пустой)

Наши созданные записи занимали 7,512\*1024 – 8192 = 7,684,096 байт

Шаги 5-6. Vacuum и Vacuum full

Сейчас же протестируем команды *vacuum* и *vacuum full*, для этого повторим действия для заполнения таблицы

Для vacuum будем использовать код:

# Очистка данных vacuum

con.set\_isolation\_level(psycopg2.extensions.ISOLATION\_LEVEL\_AUTOCOMMIT)

q = "VACUUM worker;"

cur.execute(q)

con.commit()

А для vacuum full следующий код:

# Очистка данных vacuum full

con.set\_isolation\_level(psycopg2.extensions.ISOLATION\_LEVEL\_AUTOCOMMIT)

q = "VACUUM FULL worker;"

cur.execute(q)

con.commit()

Необходимо отметить, что выполнение функции set\_isolation\_level(…) обязательно, и позволяет нам использовать команды vacuum и vacuum full, обеспечивая выполнение кода вне транзакционного кода.

**Vacuum**

До: 7512 kB

После: 7520 kB

**Vacuum full**

До: 7512 kB

После: 7488 kB

Самая большая разница между Vacuum Full и Vacuum заключается в том, что Vacuum Full физически удаляет мертвые кортежи и повторно освобождает освобожденное пространство операционной системе, поэтому после вакуумирования размер таблицы будет уменьшен до фактического размера. По сравнению с *vacuum* недостатком *vacuum full* является то, что во время выполнения невозможно получить доступ к таблице. Из-за необходимости импортировать данные кортежей в новую таблицу, эффективность ее выполнения будет очень низкой. Преимущество состоит в том, что после выполнения в табличном пространстве хранятся только «живые» кортежи, и отсутствуют избыточные «мертвые» кортежи, что повысит эффективность выполнения запроса.

**Часть 3**

**TOAST** – это отдельная таблица, связанная с основной таблицей. Она используется для хранения больних фрагментов данных из колонок, поддерживающих TOAST-обработку (например, **text**).

Попробуем найти такую таблицу, связанную с нашей таблицей woker. Для этого воспользуемся pg\_class и выполним:

q = 'SELECT relname, reltoastrelid FROM pg\_class WHERE relname = %s;'

cur.execute(q, ('worker', ))

print(cur.fetchall())

В ответе:

('worker', **24595**),

Затем

q = 'SELECT relname FROM pg\_class WHERE oid = 24595;'

cur.execute(q)

print(cur.fetchall())

В ответе получим:

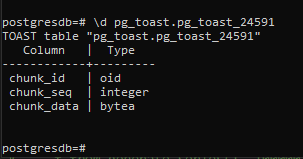
('pg\_toast\_24591',)

Как и можно было ожидать, была создана таблица pg\_toast\_24591

Посмотрим на таблицу TOAST, выполним функцию

\d pg\_toast.pg\_toast\_24591

В выводе должны получить:



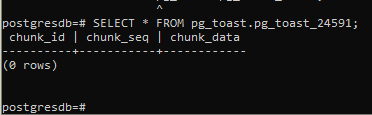
Она состоит из трёх столбцов:

chunk\_id: ссылка на значение, подвергнутое TOAST-обработке.

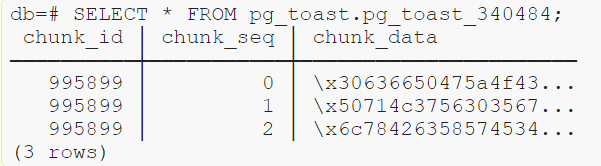
chunk\_seq: последовательный номер порции данных, представляющей часть значения.

chunk\_data: порция данных.

К сожалению, хотя и добавив большой текст размером 5000 символов, у меня не вышло спровоцировать обновление TOAST-таблицы



Однако ниже приведу пример из интернета заполненной TOAST-таблицы:



Как и ожидалось, большое значение хранится за пределами основной таблицы, в TOAST-таблице. Так как это значение слишком велико и не помещается в одну строку, PostgreSQL разделил это значение на несколько порций.

Записи вида \x063 – это то, как psql выводит двоичные данные.

Исходный код python:

import psycopg2

# Узнать в каком файле находится база данных

def get\_db\_placement(self):

    q = 'show data\_directory;'

    cur.execute(q)

    print(cur.fetchall())

#  Узнать размер БД

def get\_db\_size(self):

    q = 'select pg\_database\_size(%s);'

    cur.execute(q, ('postgresdb', ))

    row = cur.fetchone()

    print(str(row[0]))

# Создание таблицы

def create\_table(self):

    cur.execute("CREATE TABLE worker (id SERIAL PRIMARY KEY, " +

        "first\_name text, age INTEGER)")

    con.commit()

# Размер таблицы

def get\_table\_size(self):

    q = 'select pg\_size\_pretty(pg\_table\_size(%s));'

    cur.execute(q, ('worker', ))

    row = cur.fetchone()

    print(str(row[0]))

# Очистить всё содержимое таблицы

def truncate\_table(self):

    q = 'TRUNCATE worker;'

    cur.execute(q)

    con.commit()

# Поиск таблицы TOAST

def search\_toast(self):

    q = 'SELECT relname, reltoastrelid FROM pg\_class WHERE relname = %s;'

    cur.execute(q, ('worker', ))

    print(cur.fetchall())

# Заполнение данных случайными значениями

def insert\_rand(self):

    q = (

        f"insert into worker ("

        f"id, first\_name, age"

        f")"

        f"select "

        f"i,"

        f"md5(random()::text),"

        f"random()\*10000+1"

        f"from generate\_series(1, 100000) i;"

        )

    cur.execute(q)

    con.commit() # сохраняем изменения

# Показать таблицу TOAST

def show\_toast(self):

    q = 'SELECT \* FROM pg\_toast.pg\_toast\_24591;'

    cur.execute(q)

    print(cur.fetchall())

# Узнать название таблицы TOAST

def get\_toast\_name():

    q = 'SELECT relname FROM pg\_class WHERE oid = 24595;'

    cur.execute(q)

    print(cur.fetchall())

# Очистка данных vacuum

def vacuum(self):

    con.set\_isolation\_level(psycopg2.extensions.ISOLATION\_LEVEL\_AUTOCOMMIT)

    q = "VACUUM FULL worker"

    cur.execute(q)

    con.commit()

con = psycopg2.connect(

    database="postgresdb",

    user="postgres",

    password="1234",

    host="localhost",

    port="5432"

)

print("Database opened successfully")

#cursor

cur = con.cursor()

# Закрыть курсор

cur.close()

# Закрыть подключение

con.close()