Практика №1

1. Установите в postgresql.conf для параметра work_mem значение 8 Мбайт.

Обновите конфигурацию и проверьте, что изменения вступили в силу.

2. Запишите в файл ddl.sql команду CREATE TABLE

на создание любой таблицы.

Запишите в файл populate.sql команды на вставку строк в эту таблицу.

Войдите в psql, выполните оба скрипта и проверьте, что таблица создалась и в ней появились записи.

3. Найдите в журнале сервера строки за сегодняшний день.

Решение

Для выполнения практических заданий нужно войти в операционную систему под пользователем student (пароль student).

Для запуска psql в окне терминала наберите psql без параметров. Для подключения будут использованы настройки по умолчанию.

student:~$ **psql**
Для выполнения заданий каждой темы удобно создавать отдельную

базу данных:

student/student=# **CREATE DATABASE tools_overview;**CREATE DATABASE

student/student=# **\c tools_overview**
You are now connected to database "tools_overview" as user "student".
student/tools_overview=#

1. Воспользуйтесь любым текстовым редактором. В виртуальной машине установлены mousepad, gedit, vim, nano.

Обратите внимание: если вы запускаете редактор не из терминала, а из графической среды, он будет запущен под пользователем ОС student. В этом случае у вас не будет прав на редактирование файла postgresql.conf, который принадлежит пользователю postgres.

1. Параметры конфигурации

Добавим в конец конфигурационного файла строку:

```
student$ echo 'work_mem = 8MB' | sudo tee -a
/etc/postgresql/12/main/postgresql.confwork_mem = 8MB
```

Это можно сделать и любым текстовым редактором.
Обновляем конфигурацию:

```
student$ sudo pg_ctlcluster 12 main reloadПроверяем:
 student$ psql
 => SELECT current_setting('work_mem') AS work_mem;

 work_mem
----------

8MB
 (1 row)
```

2. Выполнение скриптов в psql

Запишем в файл ddl.sql команду на создание таблицы с ключевыми словами PostgreSQL (для этого можно использовать и любой текстовый редактор):

```
student$ cat >ddl.sql <<EOF
CREATE TABLE keywords (
   word text,
   category text,
   description text
);
EOF
```

Запишем команды для заполнения таблицы keywords в файл populate.sql:

```
student$ cat >populate.sql <<EOF
INSERT INTO keywords
   SELECT * FROM pg_get_keywords();
EOF
```

Создаем базу данных и подключаемся к ней:

```
=> CREATE DATABASE tools_overview;
 CREATE DATABASE
 => \c tools_overview
 You are now connected to database "tools_overview" as user "student".Выполняем
```

скрипты и проверяем записи в таблице:

```
=> \i ddl.sql
 CREATE TABLE
 => \i populate.sql
 INSERT 0 442
 => SELECT * FROM keywords LIMIT 10;

word     | category | description
----------+----------+-------------

abort |U | unreserved
absolute |U | unreserved
access |U | unreserved
action |U | unreserved
add |U | unreserved
admin |U | unreserved
after |U | unreserved
aggregate |U | unreserved
all |R | reserved
also |U| unreserved

(10 rows)
```

3. Просмотр журнала

Журнал можно открыть любым текстовым редактором. Каждая запись в журнале начинается с даты (настройки журнала после установки из пакета) и может состоять из нескольких строк. Записи за сегодняшний день будут в конце файла.

```
student$ tail /var/log/postgresql/postgresql-12-main.log


2021-10-19 23:26:45.143 MSK [83742] bob@access_overview CONTEXT: SQL
function "foo" statement 1
2021-10-19 23:26:45.143 MSK [83742] bob@access_overview STATEMENT: SELECT
foo();
 2021-10-19 23:26:45.482 MSK [83742] bob@access_overview ERROR: permission
denied for function foo
2021-10-19 23:26:45.482 MSK [83742] bob@access_overview STATEMENT: SELECT
foo();
2021-10-19 23:26:45.701 MSK [83742] bob@access_overview ERROR: permission
denied for function baz
2021-10-19 23:26:45.701 MSK [83742] bob@access_overview STATEMENT: SELECT
baz();
 2021-10-19 23:27:07.534 MSK [86674] student@student ERROR: database
"tools_overview" does not exist
```

2021-10-19 23:27:07.534 MSK [86674] student@student STATEMENT: DROP DATABASE tools_overview;
2021-10-19 23:27:07.711 MSK [59805] LOG: received SIGHUP, reloading configuration files
2021-10-19 23:27:07.712 MSK [59805] LOG: parameter "work_mem" changed to "8MB"

Практика №2

1. Создайте таблицу с одной строкой.
Начните первую транзакцию на уровне изоляции Read Committed и выполните запрос к таблице.
Во втором сеансе удалите строку и зафиксируйте изменения. Сколько строк увидит первая транзакция, выполнив тот же запрос повторно? Проверьте.
Завершите первую транзакцию.

2. Повторите все то же самое, но пусть теперь транзакция работает на уровне изоляции Repeatable Read:
BEGIN ISOLATION LEVEL REPEATABLE READ;
Объясните отличия.

Решение

1. Уровень изоляции Read Committed

Создаем таблицу:

```
=> CREATE TABLE t(n integer);
CREATE TABLE
=> INSERT INTO t VALUES (42);
INSERT 0 1
```

Запрос из первой транзакции:

```
=> BEGIN;
 BEGIN
 => SELECT * FROM t;

n ----

42
 (1 row)
```

Удаляем строку во второй транзакции и фиксируем изменения:

```
   => DELETE FROM t;
DELETE 1
```

Повторим запрос:

```
=> SELECT * FROM t;

n ---

(0 rows)
```

Первая транзакция видит произошедшие изменения.

```
=> COMMIT;
COMMIT
```

2. Уровень изоляции Repeatable Read

Вернем строку:

```
=> INSERT INTO t VALUES (42);
 INSERT 0 1
```

Запрос из первой транзакции:

```
 => BEGIN ISOLATION LEVEL REPEATABLE READ;
BEGIN

=> SELECT * FROM t;

n ----

42
 (1 row)
```

Удаляем строку во второй транзакции и фиксируем изменения:

```
   => DELETE FROM t;
DELETE 1
```

Повторим запрос:

```
=> SELECT * FROM t;

n ----

42
 (1 row)
```

На этом уровне изоляции первая транзакция не видит изменений.

```
=> COMMIT;
COMMIT
```

Практика №3

1. Проверьте, как используется буферный кеш в случае обновления одной строки в обычной и во временной таблице. Попробуйте объяснить отличие.

2. Создайте нежурналируемую таблицу и вставьте в нее несколько строк. Сымитируйте сбой системы, остановив сервер в режиме immediate, как в демонстрации. Запустите сервер и проверьте, что произошло с таблицей. Найдите в журнале сообщений сервера упоминание о восстановлении после сбоя.

Решение

1. Временные таблицы выглядят так же, как обычные, но время их жизни — текущий сеанс. Такая таблица видна тоже только в текущем сеансе.

Воспользуйтесь командой

EXPLAIN (analyze, buffers, costs off, timing off)

2. Останов в режиме immediate выполняется так:

sudo pg_ctlcluster 12 main stop -m immediate --skip-systemctl-redirect

Ключ --skip-systemctl-redirect нужен здесь из-за того, что используется PostgreSQL, установленный в Ubuntu из пакета. Он управляется командой pg_ctlcluster, которая вызывает утилиту systemctl, которая в свою очередь вызывает pg_ctl, теряя при этом указанный режим. Ключ позволяет обойтись без systemctl и передать информацию pg_ctl.

1. Использование кеша для обычных и временных таблиц

```
=> CREATE DATABASE arch_wal_overview;
 CREATE DATABASE
 => \c arch_wal_overview
 You are now connected to database "arch_wal_overview" as user "student".
```

Создадим обычную таблицу с одной строкой...

```
=> CREATE TABLE t(n integer);
CREATE TABLE
=> INSERT INTO t(n) VALUES (1);
INSERT 0 1
```

...и такую же временную таблицу.

```
=> CREATE TEMPORARY TABLE tt(n integer);
CREATE TABLE
=> INSERT INTO tt(n) VALUES (1);
INSERT 0 1
```

Обновление строки в обычной таблице:

```
=> EXPLAIN (analyze, buffers, costs off, timing off)
UPDATE t SET n = n + 1;
                QUERY PLAN
-------------------------------------------
 Update on t (actual rows=0 loops=1)
   Buffers: shared hit=2
   ->  Seq Scan on t (actual rows=1 loops=1)
         Buffers: shared hit=1
 Planning Time: 0.220 ms
 Execution Time: 0.082 ms
(6 rows)
```

При сканировании таблицы (Seq Scan) страница была найдена в буферном кеше (shared hit=1).
При обновлении потребовалось прочитать две страницы (shared hit=2). Вторая страница относится к карте видимости (с которой мы познакомимся в модуле «Организация данных»).

Обновление строки во временной таблице:

```
=> EXPLAIN (analyze, buffers, costs off, timing off)
UPDATE tt SET n = n + 1;
                QUERY PLAN
-------------------------------------------
 Update on tt (actual rows=0 loops=1)
   Buffers: local hit=2
   ->  Seq Scan on tt (actual rows=1 loops=1)
         Buffers: local hit=1
 Planning Time: 0.059 ms
 Execution Time: 0.120 ms
(6 rows)
```

Отличие в том, что вместо общего буферного кеша, расположенного в разделяемой памяти сервера, используется локальный кеш текущего сеанса (local).

2. Нежурналируемые таблицы при сбое

Создаем нежурналируемую таблицу:

```
=> CREATE UNLOGGED TABLE u(s text);
CREATE TABLE
=> INSERT INTO u VALUES ('Привет!');
INSERT 0 1
```

Имитируем сбой:

```
student$ sudo pg_ctlcluster 12 main stop -m immediate --skip-systemctl-redirect
```

Запускаем сервер:

```
 student$ sudo pg_ctlcluster 12 main start
 student$ psql arch_wal_overview
```

```
=> SELECT * FROM u;
```

```
s ---

(0 rows)
```

Таблица на месте, но она пуста. Содержимое нежурналируемых таблиц не восстанавливается при сбое; вместо этого такие таблицы «обнуляются».

Проверим журнал сообщений:

```
student$ tail -n 5 /var/log/postgresql/postgresql-12-main.log
```

```
2021-10-19 23:27:15.827 MSK [88060] LOG: database system was not properly shut down; automatic recovery in progress
2021-10-19 23:27:15.831 MSK [88060] LOG: redo starts at 0/3A9D3480
2021-10-19 23:27:15.832 MSK [88060] LOG: invalid record length at 0/3A9F24A0: wanted 24, got 0
2021-10-19 23:27:15.832 MSK [88060] LOG: redo done at 0/3A9F2038
2021-10-19 23:27:15.869 MSK [88059] LOG:  database system is ready to accept connections
```

Здесь мы видим, что при запуске был установлен факт аварийного завершения и было произведено автоматическое восстановление.

Практика №4

1. В новой базе данных создайте схему, названную так же, как и пользователь. Создайте схему app.

Создайте несколько таблиц в обеих схемах.

2. Получите в psql описание созданных схем и список всех таблиц в них.

3. Установите путь поиска так, чтобы при подключении к базе данных таблицы из обеих схем были доступны по неквалифицированному имени; приоритет должна иметь «пользовательская» схема.

Проверьте правильность настройки.

Решение

1. База данных, схемы, таблицы

Создаем базу данных:

```
=> CREATE DATABASE data_logical;
 CREATE DATABASE
 => \c data_logical
 You are now connected to database "data_logical" as user "student".
```

Схемы:

```
=> CREATE SCHEMA student;
CREATE SCHEMA
 => CREATE SCHEMA app;
CREATE SCHEMA
```

Таблицы для схемы student:

```
=> CREATE TABLE a(s text);
CREATE TABLE
 => INSERT INTO a VALUES ('student');
INSERT 0 1
 => CREATE TABLE b(s text);
 CREATE TABLE
 => INSERT INTO b VALUES ('student');
INSERT 0 1
```

Таблицы для схемы app:

```
 => CREATE TABLE app.a(s text);
CREATE TABLE
 => INSERT INTO app.a VALUES ('app');
INSERT 0 1
 => CREATE TABLE app.c(s text);
CREATE TABLE
 => INSERT INTO app.c VALUES ('app');
INSERT 0 1
```

2. Описание схем и таблиц

Описание схем:

```
=> \dn

  List of schemas
  Name   |  Owner
---------+----------
 app     | student
 public  | postgres
 student | student


(3 rows)
```

Описание таблиц:

```
=> \dt student.*

        List of relations
 Schema  | Name | Type  |  Owner
---------+------+-------+---------
 student | a    | table | student
 student | b    | table | student


(2 rows)
```

```
=> \dt app.*

        List of relations
 Schema | Name | Type  |  Owner
--------+------+-------+---------
 app    | a    | table | student
 app    | c    | table | student


(2 rows)
```

3. Путь поиска

С текущими настройками пути поиска видны таблицы только схемы student:

```
=> SELECT * FROM a;
   s
---------
 student


(1 row)

=> SELECT * FROM b;
   s
---------
 student


(1 row)

=> SELECT * FROM c;
ERROR:  relation "c" does not exist
LINE 1: SELECT * FROM c;


^

=> ALTER DATABASE data_logical SET search_path = "$user",app,public;
ALTER DATABASE
 => \c
 You are now connected to database "data_logical" as user "student".
=> SHOW search_path;

    search_path
----------------------
 "$user", app, public
(1 row)
```

Теперь видны таблицы из обеих схем, но приоритет остается за student:

```
=> SELECT * FROM a;
   s
---------
 student


(1 row)
```

```
=> SELECT * FROM b;
   s
---------
 student

(1 row)
```

Изменим путь поиска.

```
=> SELECT * FROM c;

s ----- app

(1 row)
```

Практика №5

1. Создайте новую базу данных и подключитесь к ней. Создайте табличное пространство ts.

Создайте таблицу t в табличном пространстве ts и добавьте в нее несколько строк.

2. Вычислите объем, занимаемый базой данных, таблицей и табличными пространствами ts и pg_default.

3. Перенесите таблицу в табличное пространство pg_default. Как изменился объем табличных пространств?

4. Удалите табличное пространство ts.

Решение

1. Табличные пространства и таблица

Создаем базу данных:

```
=> CREATE DATABASE data_physical;
 CREATE DATABASE
 => \c data_physical
 You are now connected to database "data_physical" as user "student".
```

Табличное пространство:

```
postgres$ mkdir /var/lib/postgresql/ts_dir
 => CREATE TABLESPACE ts LOCATION '/var/lib/postgresql/ts_dir';
CREATE TABLESPACE
```

Создаем таблицу:

```
 => CREATE TABLE t(n integer) TABLESPACE ts;
 CREATE TABLE
 => INSERT INTO t SELECT 1 FROM generate_series(1,1000);
 INSERT 0 1000
```

2. Размер данных

Объем базы данных:

```
=> SELECT pg_size_pretty(pg_database_size('data_physical')) AS db_size;
 db_size
---------
 8041 kB
```

(1 row)

Размер таблицы:

```
=> SELECT pg_size_pretty(pg_total_relation_size('t')) AS t_size;
 t_size
--------


64 kB (1 row)
```

Объем табличных пространств:

```
=> SELECT
    pg_size_pretty(pg_tablespace_size('pg_default')) AS pg_default_size,
    pg_size_pretty(pg_tablespace_size('ts')) AS ts_size;
 pg_default_size | ts_size
-----------------+---------
 188 MB          | 68 kB
(1 row)
```

Размер табличного пространства несколько больше размера таблицы за счет служебных файлов, хранящихся в каталоге табличного пространства.

3. Перенос таблицы

Перенесем таблицу:

```
=> ALTER TABLE t SET TABLESPACE pg_default;
ALTER TABLE
```

Новый объем табличных пространств:

```
=> SELECT
    pg_size_pretty(pg_tablespace_size('pg_default')) AS pg_default_size,
    pg_size_pretty(pg_tablespace_size('ts')) AS ts_size;

pg_default_size |  ts_size
-----------------+------------
 188 MB          | 4096 bytes
(1 row)
```

4. Удаление табличного пространства

Удаляем табличное пространство:

```
=> DROP TABLESPACE ts;
DROP TABLESPACE


 => \c bookstore
 You are now connected to database "bookstore" as user "student".

=> SHOW search_path;

search_path
------------------
bookstore, public

(1 row)
```

2. Таблицы

Авторы:

```
=> CREATE TABLE authors(
 author_id integer PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
 last_name text NOT NULL,
 first_name text NOT NULL,
 middle_name text
);

CREATE TABLE
```

Книги:

```
=> CREATE TABLE books(
 book_id integer PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
title text NOT NULL
);

CREATE TABLE
```

Авторство:

```
=> CREATE TABLE authorship(
 book_id integer REFERENCES books,
 author_id integer REFERENCES authors,
 seq_num integer NOT NULL,
 PRIMARY KEY (book_id,author_id)
);

CREATE TABLE
```

Операции:

```
=> CREATE TABLE operations(
 operation_id integer PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
book_id integer NOT NULL REFERENCES books,
 qty_change integer NOT NULL,
 date_created date NOT NULL DEFAULT current_date

);
```

CREATE TABLE

3. Данные

Авторы:

```
=> INSERT INTO authors(last_name, first_name, middle_name) VALUES

('Пушкин', 'Александр', 'Сергеевич'), ('Тургенев', 'Иван', 'Сергеевич'),
('Стругацкий', 'Борис', 'Натанович'), ('Стругацкий', 'Аркадий', 'Натанович'),
('Толстой', 'Лев', 'Николаевич'), ('Свифт', 'Джонатан', NULL);
```

INSERT 0 6

Книги:

```
=> INSERT INTO books(title) VALUES

('Сказка о царе Салтане'),
 ('Муму'),
 ('Трудно быть богом'),
 ('Война и мир'),
 ('Путешествия в некоторые удаленные страны мира в четырех частях:
сочинение Лемюэля Гулливера, сначала хирурга, а затем капитана
нескольких кораблей'),
('Хрестоматия');
```

INSERT 0 6

Авторство:

```
=> INSERT INTO authorship(book_id, author_id, seq_num) VALUES

(1, 1, 1), (2, 2, 1), (3, 3, 2), (3, 4, 1), (4, 5, 1), (5, 6, 1), (6, 1, 1), (6, 5, 2), (6, 2, 3);
```

INSERT 0 9

Операции.

Другой способ вставки данных в таблицу — команда COPY. Она обычно используется, если нужно загрузить большой объем информации. Но в этом случае надо не забыть «передвинуть» значение последовательности:

```
=> COPY operations (operation_id, book_id, qty_change) FROM stdin;

 1 1 10
 2 1 10
 3 1 -1

\.

COPY 3

=> SELECT pg_catalog.setval('operations_operation_id_seq', 3, true);

setval
--------

3 (1 row)
```

4. Представления

Представление для авторов:

```
=> CREATE VIEW authors_v AS

SELECT a.author_id,
 a.last_name || ' ' || a.first_name || coalesce(' ' || nullif(a.middle_name, ''), '') AS
display_name

FROM authors a;

CREATE VIEW
```

Представление для каталога:

```
=> CREATE VIEW catalog_v AS
SELECT b.book_id,
 b.title AS display_name
 FROM books b;

 CREATE VIEW
```

Представление для операций:

```
=> CREATE VIEW operations_v AS SELECT book_id,

CASE
 WHEN qty_change > 0 THEN 'Поступление' ELSE 'Покупка'
```

```
END op_type,
 abs(qty_change) qty_change,
 to_char(date_created, 'DD.MM.YYYY') date_created

FROM operations
ORDER BY operation_id;
```

CREATE VIEW