Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования «СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Департамент цифровых, робототехнических систем и электроники института перспективной инженерии

Отчет по лабораторной работе №2

Организация данных и системный каталог

Сведения о студенте

Дата: 2025-10-26
Семестр: 7
Группа: ПИЖ-б-о-22-1
Дисциплина: Администрирование баз данных
Студент: Гойдь Олеся Яновна
Направление: 09.03.04 "Программная инженерия"
Профиль: Разработка и сопровождение программного обеспечения
Форма обучения: очная

Руководитель практики: Щеголев Алексей Алексеевич
старший преподаватель департамента цифровых, робототехнических систем и электроники института перспективной инженерии

Цель работы

Всестороннее изучение логической и физической структуры хранения данных в PostgreSQL. Получение практических навыков управления базами данных, схемами, табличными пространствами. Глубокое освоение работы с системным каталогом для извлечения метаинформации. Исследование низкоуровневых аспектов хранения, включая TOAST.

Теоретическая часть

Изученные концепции

- **Логическая структура данных в PostgreSQL:** Иерархия от кластера БД через базы данных и схемы к объектам (таблицам, представлениям)
- **Физическая структура хранения:** Табличные пространства связывают логические объекты с физическими файлами на диске

- **Системный каталог:** Набор системных таблиц и представлений, содержащих метаданные обо всех объектах БД
- **Механизм TOAST:** Технология хранения больших данных вне основной таблицы с поддержкой сжатия
- **Слои хранения:** Организация файлов данных (main, fsm, vm) и сегментация при превышении 1 ГБ
- **Нежурналируемые таблицы:** UNLOGGED таблицы для повышения производительности без гарантий сохранности

Ключевые термины

- **База данных (Database):** Логический контейнер для схем и объектов, изолированный от других баз в кластере
- **Схема (Schema):** Пространство имен для группировки объектов внутри базы данных
- **Табличное пространство (Tablespace):** Физическое расположение данных на диске, может использоваться несколькими базами
- **Системный каталог (System Catalog):** Метаданные PostgreSQL в виде системных таблиц (pg_class, pg_database, pg_namespace)
- **search_path:** Параметр, определяющий порядок поиска объектов по схемам
- **TOAST (The Oversized-Attribute Storage Technique):** Механизм хранения больших значений вне основной таблицы
- **Стратегии хранения:** plain, extended, external, main - различные подходы к хранению больших данных
- **Слои (forks):** Отдельные файлы для разных типов данных объекта (main, fsm, vm, init)
- **OID (Object Identifier):** Уникальный числовой идентификатор объекта в системном каталоге

Практическая часть

Модуль 1: Базы данных и схемы

Задача 1.1: Создание базы данных и проверка размера

Цель: Создать новую базу данных и изучить её начальный размер

Выполненные действия:

```
CREATE DATABASE lab02_db;
SELECT pg_database_size('lab02_db');
```

Результаты:

```
CREATE DATABASE
 pg_database_size
------------------
          7602703
(1 row)
```

Выводы:

Начальный размер новой базы данных составил 7602703 байт (примерно 7.3 МБ). Этот размер включает системный каталог и служебные структуры данных, необходимые для функционирования базы.

Задача 1.2: Создание схем и таблиц

Цель: Освоить работу со схемами как механизмом организации объектов в базе данных

Выполненные действия:

```
\c lab02_db

CREATE SCHEMA app;
CREATE SCHEMA student;

CREATE TABLE app.users(
    id SERIAL PRIMARY KEY,
    username VARCHAR(50)
);

CREATE TABLE student.users(
    id SERIAL PRIMARY KEY,
    email VARCHAR(100)
);

INSERT INTO app.users (username) VALUES ('name1'), ('name2'), ('name3');
INSERT INTO student.users (email) VALUES
    ('name1@gmail.com'),
    ('name2@gmail.com'),
    ('name3@gmail.com');

SELECT * FROM app.users;
SELECT * FROM student.users;
```

Результаты:

```
CREATE SCHEMA
CREATE SCHEMA
CREATE TABLE
CREATE TABLE
INSERT 0 3
INSERT 0 3

 id | username
----+----------
  1 | name1
```

```
  2 | name2
  3 | name3
(3 rows)

 id |     email
----+----------------
  1 | name1@gmail.com
  2 | name2@gmail.com
  3 | name3@gmail.com
(3 rows)
```

Выводы:

Успешно созданы две схемы (app и student) с идентичными по структуре таблицами users. Схемы позволяют изолировать объекты с одинаковыми именами в разных пространствах имен.

Задача 1.3: Контроль изменения размера базы данных

Цель: Проанализировать изменение размера БД после добавления объектов

Выполненные действия:

```
SELECT pg_database_size('lab02_db');
```

Результаты:

```
 pg_database_size
------------------
          7877091
(1 row)
```

Выводы и объяснения:

Начальный размер БД: **7602703 байт**
Текущий размер: **7877091 байт**
Разница: **274388 байт** (около 268 КБ)

Размер базы данных увеличился из-за создания новых объектов и данных. При создании двух схем (app и student) PostgreSQL добавил информацию о них в системный каталог. При создании таблиц СУБД выделила место для хранения структуры таблиц, включая метаданные о колонках, типах данных, первичных ключах и последовательностях для SERIAL. После вставки записей PostgreSQL сохранила сами данные на диске. Размер также увеличился за счет служебной информации: индексов для первичных ключей, статистики таблиц и внутренних структур данных для управления объектами.

Задача 1.4: Управление путем поиска (search_path)

Цель: Настроить приоритет поиска объектов по схемам

Выполненные действия:

```
-- Установка приоритета схеме app
SET search_path = app, student, public;
SELECT * FROM users;

-- Установка приоритета схеме student
SET search_path = student, app, public;
SHOW search_path;
SELECT current_schemas(true);
SELECT * FROM users;
```

Результаты:

```
SET
 id | username
----+----------
  1 | name1
  2 | name2
  3 | name3
(3 rows)

SET
     search_path
----------------------
 student, app, public
(1 row)

         current_schemas
---------------------------------
 {pg_catalog,student,app,public}
(1 row)

 id |     email
----+---------------
  1 | name1@gmail.com
  2 | name2@gmail.com
  3 | name3@gmail.com
(3 rows)
```

Выводы:

Параметр search_path успешно управляет приоритетом поиска объектов. При первой настройке (app, student, public) запрос к таблице users без указания схемы обратился к app.users. После изменения приоритета на (student, app, public) тот же запрос вернул данные из student.users. Функция current_schemas() показала полный список схем включая системную pg_catalog, которая всегда имеет наивысший приоритет.

Задача 1.5: Настройка параметра базы данных

Цель: Установить параметр temp_buffers на уровне базы данных

Выполненные действия:

```
SHOW temp_buffers;
ALTER DATABASE lab02_db SET temp_buffers = '32MB';

-- Переподключение
\c postgres
\c lab02_db

SHOW temp_buffers;
```

Результаты:

```
 temp_buffers
--------------
 8MB
(1 row)

ALTER DATABASE

You are now connected to database "postgres" as user "student".
You are now connected to database "lab02_db" as user "student".

 temp_buffers
--------------
 32MB
(1 row)
```

Выводы:

Параметр temp_buffers успешно установлен на уровне базы данных. Значение по умолчанию (8MB) было увеличено в 4 раза до 32MB. Изменение вступило в силу только после переподключения к базе данных, что подтверждает, что параметры уровня базы данных применяются при создании нового сеанса.

Модуль 2: Системный каталог

Задача 2.1: Исследование pg_class

Цель: Изучить структуру системной таблицы pg_class

Выполненные действия:

```
\d pg_class
```

Результаты:

```
                    Table "pg_catalog.pg_class"
       Column        |     Type     | Collation | Nullable | Default
---------------------+--------------+-----------+----------+---------
 oid                 | oid          |           | not null |
 relname             | name         |           | not null |
 relnamespace        | oid          |           | not null |
 reltype             | oid          |           | not null |
 reloftype           | oid          |           | not null |
 relowner            | oid          |           | not null |
 relam               | oid          |           | not null |
 relfilenode         | oid          |           | not null |
 reltablespace       | oid          |           | not null |
 relpages            | integer      |           | not null |
 reltuples           | real         |           | not null |
 relallvisible       | integer      |           | not null |
 reltoastrelid       | oid          |           | not null |
 relhasindex         | boolean      |           | not null |
 relisshared         | boolean      |           | not null |
 relpersistence      | "char"       |           | not null |
 relkind             | "char"       |           | not null |
 relnatts            | smallint     |           | not null |
 relchecks           | smallint     |           | not null |
 relhasrules         | boolean      |           | not null |
 relhastriggers      | boolean      |           | not null |
 relhassubclass      | boolean      |           | not null |
 relrowsecurity      | boolean      |           | not null |
 relforcerowsecurity | boolean      |           | not null |
 relispopulated      | boolean      |           | not null |
 relreplident        | "char"       |           | not null |
 relispartition      | boolean      |           | not null |
 relrewrite          | oid          |           | not null |
 relfrozenxid        | xid          |           | not null |
 relminmxid          | xid          |           | not null |
 relacl              | aclitem[]    |           |          |
 reloptions          | text[]       | C         |          |
 relpartbound        | pg_node_tree | C         |          |
```

```
Indexes:
    "pg_class_oid_index" PRIMARY KEY, btree (oid)
    "pg_class_relname_nsp_index" UNIQUE CONSTRAINT, btree (relname, relnamespace)
    "pg_class_tblspc_relfilenode_index" btree (reltablespace, relfilenode)
```

Выводы:

Таблица pg_class содержит информацию обо всех объектах базы данных: таблицах, индексах, представлениях, последовательностях. Ключевые поля: oid (уникальный идентификатор), relname (имя объекта), relkind (тип объекта), relnamespace (схема), reltablespace (табличное пространство), reltoastrelid (ссылка на TOAST-таблицу).

Задача 2.2: Исследование pg_tables и разница между таблицами и представлениями

Цель: Изучить представление pg_tables и понять разницу между таблицами и представлениями

Выполненные действия:

```
\d+ pg_tables
```

Объяснение разницы между таблицей и представлением:

Таблица — это физическая структура данных, которая реально хранит данные на диске. При создании таблицы и вставке записей PostgreSQL выделяет место на диске и сохраняет данные в файлах. Таблица имеет собственную структуру хранения, индексы и занимает физическое пространство.

Представление (view) — это виртуальная таблица, которая не хранит данные физически. Представление — это сохраненный SQL-запрос, который выполняется каждый раз при обращении к нему. При выполнении SELECT из представления PostgreSQL автоматически выполняет определенный в представлении запрос и возвращает результат. Представления используются для упрощения сложных запросов, обеспечения безопасности данных (показывая только определенные колонки или строки) и создания удобных интерфейсов для работы с данными.

Задача 2.3: Временные таблицы и схемы

Цель: Изучить механизм временных схем в PostgreSQL

Выполненные действия:

```
CREATE TEMP TABLE temp_test (
    id SERIAL PRIMARY KEY,
    data VARCHAR(50)
);

SELECT nspname AS schema_name
```

```
FROM pg_catalog.pg_namespace
ORDER BY nspname;
```

Результаты:

```
CREATE TABLE

    schema_name
--------------------
 app
 information_schema
 pg_catalog
 pg_temp_3
 pg_toast
 pg_toast_temp_3
 public
 student
(8 rows)
```

Объяснение наличия временной схемы:

После создания временной таблицы появилась схема `pg_temp_3` (где 3 — номер сессии). PostgreSQL автоматически создает временную схему для каждой сессии, в которой создаются временные объекты. Эта схема существует только в рамках текущего сеанса подключения и автоматически удаляется после завершения сессии. Временные таблицы размещаются в этой схеме, что изолирует их от обычных таблиц и делает видимыми только для текущего подключения. Также присутствует схема `pg_toast_temp_3` для TOAST-данных временных таблиц.

Задача 2.4: Представления information_schema

Цель: Получить список всех представлений стандартной схемы information_schema

Выполненные действия:

```
SELECT table_name
FROM information_schema.views
WHERE table_schema = 'information_schema'
ORDER BY table_name;
```

Результаты:

```
              table_name
---------------------------------------
 _pg_foreign_data_wrappers
```

```
_pg_foreign_servers
_pg_foreign_table_columns
_pg_foreign_tables
_pg_user_mappings
administrable_role_authorizations
applicable_roles
attributes
character_sets
check_constraint_routine_usage
check_constraints
collation_character_set_applicability
collations
column_column_usage
column_domain_usage
column_options
column_privileges
column_udt_usage
columns
constraint_column_usage
constraint_table_usage
data_type_privileges
domain_constraints
domain_udt_usage
domains
element_types
enabled_roles
foreign_data_wrapper_options
foreign_data_wrappers
foreign_server_options
foreign_servers
foreign_table_options
foreign_tables
information_schema_catalog_name
key_column_usage
parameters
referential_constraints
role_column_grants
role_routine_grants
role_table_grants
role_udt_grants
role_usage_grants
routine_column_usage
routine_privileges
routine_routine_usage
routine_sequence_usage
routine_table_usage
routines
schemata
sequences
table_constraints
table_privileges
```

```
 tables
 transforms
 triggered_update_columns
 triggers
 udt_privileges
 usage_privileges
 user_defined_types
 user_mapping_options
 user_mappings
 view_column_usage
 view_routine_usage
 view_table_usage
 views
(65 rows)
```

Выводы:

В схеме information_schema содержится 65 представлений, предоставляющих стандартизированный SQL-доступ к метаданным базы данных. Эти представления соответствуют стандарту SQL и обеспечивают переносимость запросов между различными СУБД.

Задача 2.5: Анализ метакоманды \d+ pg_views

Цель: Понять, какие запросы к системному каталогу выполняются за метакомандами psql

Выполненные действия:

```
\d+ pg_views
```

Результаты:

```
                      View "pg_catalog.pg_views"
   Column   | Type | Collation | Nullable | Default | Storage  | Description
------------+------+-----------+----------+---------+----------+-------------
 schemaname | name |           |          |         | plain    |
 viewname   | name |           |          |         | plain    |
 viewowner  | name |           |          |         | plain    |
 definition | text |           |          |         | extended |
View definition:
 SELECT n.nspname AS schemaname,
    c.relname AS viewname,
    pg_get_userbyid(c.relowner) AS viewowner,
    pg_get_viewdef(c.oid) AS definition
   FROM pg_class c
     LEFT JOIN pg_namespace n ON n.oid = c.relnamespace
  WHERE c.relkind = 'v'::"char";
```

Объяснение запросов к системному каталогу:

Представление pg_views выполняет JOIN между двумя системными таблицами:

- **pg_class** — содержит информацию обо всех объектах БД (таблицах, индексах, представлениях)
- **pg_namespace** — содержит информацию о схемах

Запрос фильтрует записи по условию `c.relkind = 'v'`, где 'v' означает view (представление), что позволяет отобрать только представления из всех объектов. Функция `pg_get_userbyid()` преобразует OID владельца в имя пользователя, а `pg_get_viewdef()` извлекает SQL-определение представления. Таким образом, простая команда `\d+ pg_views` транслируется в сложный запрос, объединяющий данные из нескольких системных таблиц.

Модуль 3: Табличные пространства

Задача 3.1: Создание табличного пространства

Цель: Создать пользовательское табличное пространство

Выполненные действия:

Команды в терминале:

```
mkdir /home/student/lab2space
sudo chown postgres:postgres /home/student/lab2space/
```

SQL-команды:

```
CREATE TABLESPACE lab02_ts LOCATION '/home/student/lab2space';
```

Результаты:

```
CREATE TABLESPACE
```

Выводы:

Успешно создано табличное пространство lab02_ts, указывающее на каталог /home/student/lab2space. Важно, что каталог должен принадлежать пользователю postgres и иметь соответствующие права доступа.

Задача 3.2: Изменение табличного пространства по умолчанию для template1

Цель: Настроить наследование табличного пространства для новых баз данных

Выполненные действия:

```
ALTER DATABASE template1 SET TABLESPACE lab02_ts;

SELECT datname, spcname
FROM pg_database d
JOIN pg_tablespace t ON d.dattablespace = t.oid
WHERE datname = 'template1';
```

Результаты:

```
ALTER DATABASE

  datname  | spcname
-----------+----------
 template1 | lab02_ts
(1 row)
```

Объяснение цели действия:

База данных template1 используется как шаблон при создании новых баз данных командой CREATE DATABASE. Когда изменяется её табличное пространство по умолчанию, все новые базы данных, создаваемые на основе этого шаблона, автоматически наследуют это табличное пространство. Это позволяет централизованно управлять размещением данных для всех будущих баз данных без необходимости явно указывать табличное пространство при каждом CREATE DATABASE.

Задача 3.3: Наследование свойств табличного пространства

Цель: Проверить механизм наследования табличного пространства от шаблона

Выполненные действия:

```
CREATE DATABASE lab02_db_new;
\c lab02_db_new

SELECT spcname
FROM pg_tablespace
WHERE oid = (SELECT dattablespace FROM pg_database WHERE datname =
'lab02_db_new');
```

Результаты:

```
CREATE DATABASE
You are now connected to database "lab02_db_new" as user "student".

 spcname
----------
 lab02_ts
(1 row)
```

Объяснение результата:

Новая база данных lab02_db_new унаследовала табличное пространство lab02_ts от базы-шаблона template1, так как мы изменили настройку по умолчанию для шаблона. Это демонстрирует механизм наследования свойств от шаблона к создаваемым базам данных, что является мощным инструментом для стандартизации конфигурации новых баз.

Задача 3.4: Символические ссылки табличных пространств

Цель: Понять механизм связи логических табличных пространств с физическими каталогами

Выполненные действия:

SQL-команды:

```
SELECT oid AS tsoid FROM pg_tablespace WHERE spcname = 'lab02_ts';
SHOW data_directory;
```

Команды в терминале:

```
sudo ls -la /var/lib/postgresql/16/main/pg_tblspc/16418
```

Результаты:

```
 tsoid
-------
 16418
(1 row)

       data_directory
-----------------------------
 /var/lib/postgresql/16/main
(1 row)
```

```
lrwxrwxrwx 1 postgres postgres 23 окт 19 16:57
/var/lib/postgresql/16/main/pg_tblspc/16418 -> /home/student/lab2space
```

Объяснение:

В каталоге `PGDATA/pg_tblspc/` находится символьная ссылка с именем `16418` (OID табличного пространства lab02_ts), которая указывает на физический путь `/home/student/lab2space`. PostgreSQL использует механизм символьных ссылок для связи логических табличных пространств с их физическим расположением на диске. При создании табличного пространства PostgreSQL создает запись в системном каталоге pg_tablespace с уникальным OID и создает символьную ссылку в каталоге pg_tblspc/ с именем, равным этому OID. Данные табличного пространства физически хранятся в указанном каталоге, а PostgreSQL обращается к ним через символьную ссылку.

Задача 3.5: Удаление табличного пространства

Цель: Изучить процесс удаления табличного пространства и зависимостей

Выполненные действия:

```
\c postgres
DROP TABLESPACE lab02_ts;
-- ERROR: tablespace "lab02_ts" is not empty

-- Поиск зависимых баз данных
SELECT oid AS tsoid FROM pg_tablespace WHERE spcname = 'lab02_ts';

SELECT datname
FROM pg_database
WHERE oid IN (SELECT pg_tablespace_databases(16418));

-- Проверка объектов в базе lab02_db_new
\c lab02_db_new
SELECT relnamespace::regnamespace, relname, relkind
FROM pg_class
WHERE reltablespace = 16418;

-- Перемещение баз данных на pg_default
\c postgres
ALTER DATABASE lab02_db_new SET TABLESPACE pg_default;
ALTER DATABASE template1 SET TABLESPACE pg_default;

-- Удаление табличного пространства
DROP TABLESPACE lab02_ts;

SELECT spcname FROM pg_tablespace WHERE spcname = 'lab02_ts';
```

Результаты:

```
ERROR:  tablespace "lab02_ts" is not empty

 tsoid
-------
 16418
(1 row)

   datname
--------------
 lab02_db_new
 template1
(2 rows)

 relnamespace | relname | relkind
--------------+---------+---------
(0 rows)

ALTER DATABASE
ALTER DATABASE
DROP TABLESPACE

 spcname
---------
(0 rows)
```

Объяснение необходимости CASCADE и процесса удаления:

PostgreSQL **не поддерживает опцию CASCADE** для команды DROP TABLESPACE. Это сделано намеренно из соображений безопасности данных — табличные пространства могут содержать критически важные данные целых баз данных, и автоматическое каскадное удаление было бы слишком опасным. Администратор должен явно и осознанно удалить или переместить все зависимые объекты, что предотвращает случайную потерю больших объемов данных.

Процесс удаления требует:

1. Найти все базы данных, использующие табличное пространство (через pg_tablespace_databases)
2. Проверить наличие объектов в этих базах (через pg_class)
3. Удалить базы данных или переместить их на другое табличное пространство (ALTER DATABASE ... SET TABLESPACE)
4. Только после этого можно удалить само табличное пространство

Задача 3.6: Настройка параметров табличного пространства

Цель: Установить параметр random_page_cost для оптимизации работы с SSD

Выполненные действия:

```
ALTER TABLESPACE pg_default SET (random_page_cost = 1.1);

\db+
```

Результаты:

```
ALTER TABLESPACE

                                       List of tablespaces
    Name    |  Owner   | Location | Access privileges |        Options         |
Size  | Description
------------+----------+----------+-------------------+------------------------+--
------+-------------
 pg_default | postgres |          |                   | {random_page_cost=1.1} |
44 MB  |
 pg_global  | postgres |          |                   |                        |
605 kB |
(2 rows)
```

Объяснение параметра random_page_cost:

Параметр `random_page_cost` определяет оценочную стоимость произвольного (случайного) чтения страницы данных с диска для планировщика запросов PostgreSQL. По умолчанию значение равно 4.0, что соответствует традиционным жестким дискам (HDD), где произвольный доступ значительно медленнее последовательного чтения.

Для современных SSD-накопителей, где произвольное чтение выполняется почти так же быстро, как последовательное, рекомендуется устанавливать значение 1.0-1.5. Установка `random_page_cost = 1.1` сообщает планировщику PostgreSQL, что произвольный доступ на этом табличном пространстве выполняется быстро. Это влияет на выбор планов запросов: планировщик будет чаще использовать индексы вместо последовательного сканирования таблиц, что может значительно ускорить выполнение запросов на SSD-дисках.

Модуль 4: Низкий уровень

Задача 4.1: Нежурналируемые таблицы и init-файлы

Цель: Изучить механизм нежурналируемых таблиц и слой init

Выполненные действия:

Команды в терминале:

```
mkdir /tmp/unlogged_ts
sudo chown postgres:postgres /tmp/unlogged_ts
```

SQL-команды:

```
CREATE TABLESPACE unlogged_ts LOCATION '/tmp/unlogged_ts';

CREATE UNLOGGED TABLE test_unlogged (
    id SERIAL PRIMARY KEY,
    data TEXT
) TABLESPACE unlogged_ts;

SELECT pg_relation_filepath('test_unlogged');

INSERT INTO test_unlogged (data) VALUES ('some cool data');
```

Проверка файлов в терминале:

```
sudo ls -la /tmp/unlogged_ts/PG_16_202307071/16385
```

Очистка:

```
DROP TABLE test_unlogged;
DROP TABLESPACE unlogged_ts;
sudo rm -rf /tmp/unlogged_ts
```

Результаты:

```
CREATE TABLESPACE
CREATE TABLE

            pg_relation_filepath
-----------------------------------------------
 pg_tblspc/16420/PG_16_202307071/16385/16422
(1 row)

INSERT 0 1
```

Вывод ls:

```
total 32
drwx------ 2 postgres postgres 4096 окт 19 18:11 .
drwx------ 3 postgres postgres 4096 окт 19 18:11 ..
-rw------- 1 postgres postgres 8192 окт 19 18:12 16422
```

```
-rw------- 1 postgres postgres    0 окт 19 18:11 16422_init
-rw------- 1 postgres postgres    0 окт 19 18:11 16426
-rw------- 1 postgres postgres    0 окт 19 18:11 16426_init
-rw------- 1 postgres postgres 8192 окт 19 18:11 16427
-rw------- 1 postgres postgres 8192 окт 19 18:12 16427_init
```

Объяснение:

Нежурналируемые таблицы (UNLOGGED) не записывают изменения в WAL (журнал транзакций), что ускоряет работу, но делает их небезопасными при сбоях. При перезапуске сервера такие таблицы очищаются. Файл с суффиксом `_init` (16422_init) хранит пустую структуру таблицы и используется для быстрой инициализации таблицы после сбоя. Также видны init-файлы для индекса (16426_init) и последовательности (16427_init).

Задача 4.2: Стратегии хранения TOAST

Цель: Изучить механизм TOAST и различные стратегии хранения больших данных

Выполненные действия:

```
\c lab02_db

CREATE TABLE toast_test (
    id SERIAL PRIMARY KEY,
    long_text TEXT
);

-- Проверить стратегию по умолчанию
SELECT attname, attstorage
FROM pg_attribute
WHERE attrelid = 'toast_test'::regclass
  AND attname = 'long_text';

-- Изменить стратегию на external
ALTER TABLE toast_test ALTER COLUMN long_text SET STORAGE external;

-- Проверить изменение
SELECT attname, attstorage
FROM pg_attribute
WHERE attrelid = 'toast_test'::regclass
  AND attname = 'long_text';

-- Узнать OID TOAST-таблицы
SELECT oid, reltoastrelid
FROM pg_class
WHERE relname = 'toast_test';

-- Вставить данные
```

```
INSERT INTO toast_test (long_text) VALUES (repeat('short', 100));
INSERT INTO toast_test (long_text) VALUES (repeat('long text here ', 1000));

-- Проверить размер данных
SELECT id, length(long_text) as text_length, pg_column_size(long_text) as
stored_size
FROM toast_test;

-- Найти имя TOAST-таблицы
SELECT n.nspname, c.relname
FROM pg_class c
JOIN pg_namespace n ON c.relnamespace = n.oid
WHERE c.oid = (SELECT reltoastrelid FROM pg_class WHERE relname = 'toast_test');

-- Проверить содержимое TOAST-таблицы
SELECT count(*) FROM pg_toast.pg_toast_16431;

-- Детальный просмотр TOAST-чанков
SELECT chunk_id, chunk_seq, length(chunk_data) as chunk_size
FROM pg_toast.pg_toast_16431
ORDER BY chunk_id, chunk_seq;
```

Результаты:

```
CREATE TABLE

  attname  | attstorage
-----------+------------
 long_text | x
(1 row)

ALTER TABLE

  attname  | attstorage
-----------+------------
 long_text | e
(1 row)

  oid  | reltoastrelid
-------+---------------
 16431 |         16435
(1 row)

INSERT 0 1
INSERT 0 1

 id | text_length | stored_size
----+-------------+-------------
  1 |         500 |         504
```

```
  2 |       15000 |       15000
(2 rows)

 nspname  |    relname
----------+----------------
 pg_toast | pg_toast_16431
(1 row)

 count
-------
     8
(1 row)
```

Объяснение результата:

Стратегия хранения по умолчанию для TEXT — 'x' (extended), что означает сжатие и TOAST. После изменения на 'e' (external) данные не сжимаются, но большие значения выносятся в TOAST-таблицу.

Короткая строка (500 байт) осталась в основной таблице, так как не превышает пороговое значение. Длинная строка (15000 байт) была вынесена в TOAST-таблицу и разбита на 8 чанков (кусков) размером примерно 2000 байт каждый. TOAST разбивает большие значения на чанки для эффективного хранения и извлечения данных. Каждый чанк имеет chunk_id (идентификатор строки) и chunk_seq (порядковый номер от 0 до 7).

Задача 4.3: Сравнение размеров базы данных

Цель: Понять разницу между размером базы данных и размером таблиц

Выполненные действия:

```
\c lab02_db

-- Суммарный размер всех таблиц
SELECT sum(pg_total_relation_size(oid)) as tables_total_size
FROM pg_class
WHERE NOT relisshared
  AND relkind = 'r';

-- Общий размер базы данных
SELECT pg_database_size('lab02_db') as database_size;
```

Результаты:

```
 tables_total_size
-------------------
           7798784
```

```
(1 row)

 database_size
---------------
       7983587
(1 row)
```

Объяснение расхождения:

Разница: **184803 байт** (около 180 КБ)

Размер базы данных больше суммы размеров таблиц по следующим причинам:

Функция `pg_database_size` возвращает размер всего каталога базы данных в файловой системе. В этом каталоге, кроме файлов таблиц, находятся служебные файлы:

- **pg_filenode.map** — отображение OID таблиц в имена файлов
- **pg_internal.init** — кеш системного каталога
- **PG_VERSION** — файл с версией PostgreSQL

Функция `pg_total_relation_size` учитывает только сами таблицы с их индексами и TOAST-данными, но не учитывает эти служебные файлы и возможное неиспользуемое пространство в каталоге.

Задача 4.4: Проверка методов сжатия

Цель: Проверить, какие методы сжатия TOAST поддерживает PostgreSQL

Выполненные действия:

```
-- Проверить параметры сборки PostgreSQL
SELECT * FROM (
  SELECT string_to_table(setting, ' ') AS setting
  FROM pg_config WHERE name = 'CONFIGURE'
) AS config
WHERE setting ~ '(lz4|pglz)';

-- Проверить метод по умолчанию
\dconfig *toast*

-- Проверить доступные методы
SELECT setting, enumvals
FROM pg_settings
WHERE name = 'default_toast_compression';
```

Результаты:

```
   setting
--------------
 '--with-lz4'
(1 row)

 List of configuration parameters
         Parameter         | Value
---------------------------+-------
 default_toast_compression | pglz
(1 row)

 setting |  enumvals
---------+------------
 pglz    | {pglz,lz4}
(1 row)
```

Объяснение:

Представление pg_config показывает, что PostgreSQL был собран с опцией `--with-lz4`, что означает поддержку метода сжатия lz4. Метод pglz встроен по умолчанию и доступен всегда. Параметр `default_toast_compression` показывает, что по умолчанию используется метод pglz. Поле `enumvals` из pg_settings подтверждает, что доступны оба метода: pglz и lz4.

Задача 4.5: Сравнение методов сжатия

Цель: Сравнить эффективность различных методов сжатия TOAST

Выполненные действия:

Создание тестового файла в терминале:

```
sudo cat /usr/lib/postgresql/16/bin/postgres | base32 -w0 >
/tmp/compress_test.input
ls -lh /tmp/compress_test.input
```

SQL-команды:

```
\c lab02_db

-- 1) Таблица БЕЗ сжатия (external)
CREATE TABLE compress_external (txt TEXT STORAGE EXTERNAL);

\timing on
COPY compress_external FROM '/tmp/compress_test.input';
\timing off
```

```
SELECT pg_table_size('compress_external')/1024 as size_kb;

-- 2) Таблица со сжатием pglz
TRUNCATE TABLE compress_external;
ALTER TABLE compress_external
    ALTER COLUMN txt SET STORAGE EXTENDED,
    ALTER COLUMN txt SET COMPRESSION pglz;

\timing on
COPY compress_external FROM '/tmp/compress_test.input';
\timing off

SELECT pg_table_size('compress_external')/1024 as size_kb;

-- 3) Таблица со сжатием lz4
TRUNCATE TABLE compress_external;
ALTER TABLE compress_external
    ALTER COLUMN txt SET COMPRESSION lz4;

\timing on
COPY compress_external FROM '/tmp/compress_test.input';
\timing off

SELECT pg_table_size('compress_external')/1024 as size_kb;
```

Результаты:

```
-rw-rw-r-- 1 student student 16M окт 19 18:47 /tmp/compress_test.input

CREATE TABLE
Timing is on.
COPY 1
Time: 164,342 ms
Timing is off.
 size_kb
---------
   10720
(1 row)

TRUNCATE TABLE
ALTER TABLE
Timing is on.
COPY 1
Time: 468,825 ms
Timing is off.
 size_kb
---------
   10376
```

```
(1 row)

TRUNCATE TABLE
ALTER TABLE
Timing is on.
COPY 1
Time: 170,241 ms
Timing is off.
 size_kb
---------
   10712
(1 row)
```

Сравнительная таблица результатов:

Метод сжатия	Размер таблицы (КБ)	Время загрузки (мс)	Степень сжатия
EXTERNAL (без сжатия)	10720	164	-
PGLZ	10376	469	3.2%
LZ4	10712	170	0.07%

Анализ результатов:

Метод **EXTERNAL** (без сжатия) обеспечивает самую быструю загрузку данных — всего 164 миллисекунды, но таблица занимает 10720 КБ на диске. Данные хранятся в TOAST-таблице без какой-либо компрессии.

Метод **pglz** показал лучшую степень сжатия, уменьшив размер таблицы до 10376 КБ (экономия около 3.2%). Однако время загрузки увеличилось почти в 3 раза — до 469 миллисекунд. Это связано с необходимостью выполнять сжатие данных во время записи.

Метод **lz4** продемонстрировал оптимальный баланс между скоростью и эффективностью. Время загрузки составило всего 170 миллисекунд (практически как без сжатия), при этом размер таблицы — 10712 КБ. Степень сжатия оказалась минимальной (0.07%), что объясняется особенностями тестовых данных — текст, полученный кодированием base32, уже имеет высокую энтропию и плохо поддается сжатию.

Общий вывод:

Для данного типа данных (base32-кодированный текст) методы сжатия оказались малоэффективными из-за высокой энтропии исходных данных. Метод lz4 показал наилучший результат, обеспечивая высокую скорость работы при незначительных накладных расходах. В реальных условиях с текстовыми данными естественного языка или повторяющимися структурами степень сжатия была бы существенно выше. Метод lz4 рекомендуется для большинства практических применений благодаря оптимальному балансу производительности и компрессии.

Результаты выполнения

Сводная таблица результатов

Модуль	Задача	Статус	Ключевые наблюдения
1	1.1	✅ Выполнено	Начальный размер БД 7.3 МБ
1	1.2	✅ Выполнено	Созданы схемы app и student с таблицами
1	1.3	✅ Выполнено	Размер увеличился на 268 КБ
1	1.4	✅ Выполнено	search_path управляет приоритетом схем
1	1.5	✅ Выполнено	temp_buffers увеличен в 4 раза до 32MB
2	2.1	✅ Выполнено	Изучена структура pg_class (33 колонки)
2	2.2	✅ Выполнено	Понята разница между таблицами и представлениями
2	2.3	✅ Выполнено	Обнаружены временные схемы pg_temp_3
2	2.4	✅ Выполнено	В information_schema 65 представлений
2	2.5	✅ Выполнено	Изучен SQL-запрос за \d+ pg_views
3	3.1	✅ Выполнено	Создано табличное пространство lab02_ts
3	3.2	✅ Выполнено	template1 настроен на lab02_ts
3	3.3	✅ Выполнено	Новая БД унаследовала lab02_ts
3	3.4	✅ Выполнено	Найдена символьная ссылка OID 16418
3	3.5	✅ Выполнено	CASCADE не поддерживается, удаление вручную
3	3.6	✅ Выполнено	random_page_cost установлен в 1.1 для SSD
4	4.1	✅ Выполнено	Найден init-файл для UNLOGGED таблицы
4	4.2	✅ Выполнено	Длинная строка разбита на 8 TOAST-чанков
4	4.3	✅ Выполнено	Разница 180 КБ из-за служебных файлов
4	4.4	✅ Выполнено	Поддержка pglz и lz4 подтверждена
4	4.5	✅ Выполнено	lz4 показал лучший баланс скорости/сжатия

Анализ и выводы

Основные наблюдения

1. **Организация данных в PostgreSQL имеет четкую иерархию:** Кластер содержит базы данных, каждая база содержит схемы, схемы группируют объекты. Эта структура обеспечивает логическую организацию и изоляцию данных. Схемы особенно полезны для создания отдельных пространств имен внутри одной базы данных.

2. **Физическое хранение данных гибко настраивается:** Табличные пространства позволяют размещать данные на разных дисках или разделах. Механизм символьных ссылок обеспечивает

прозрачное отображение логических объектов на физические каталоги. Это критично для оптимизации производительности и управления ростом данных.

3. **Системный каталог PostgreSQL — это мощный инструмент:** Все метаданные доступны через стандартные SQL-запросы к системным таблицам. Представления вроде pg_views, pg_tables упрощают доступ к метаинформации. Понимание структуры системного каталога критично для администрирования.

4. **TOAST эффективно решает проблему больших данных:** Автоматическое вынесение больших значений в отдельные таблицы происходит прозрачно. Разбиение на чанки по ~2000 байт оптимизирует хранение и извлечение. Различные стратегии хранения (plain, extended, external, main) дают гибкость в управлении компрессией.

5. **Методы сжатия влияют на производительность по-разному:** Алгоритм lz4 обеспечивает оптимальный баланс между скоростью и степенью сжатия. Для высокоэнтропийных данных (как base32) сжатие малоэффективно. Важно учитывать тип данных при выборе стратегии сжатия.

6. **Нежурналируемые таблицы требуют особого внимания:** UNLOGGED таблицы значительно быстрее, но данные теряются при сбое. Init-файлы обеспечивают быстрое восстановление пустой структуры. Использовать только для временных или легко восстанавливаемых данных.

Сравнительный анализ

Стратегии хранения TOAST:

Стратегия	Сжатие	TOAST	Применение
PLAIN	Нет	Нет	Малые неделимые типы
EXTENDED	Да	Да	Оптимально для большинства (по умолчанию)
EXTERNAL	Нет	Да	Большие несжимаемые данные
MAIN	Да	Редко	Предпочтение inline-хранению

Методы сжатия:

Метод	Скорость сжатия	Степень сжатия	Рекомендации
pglz	Медленная	Высокая	Архивные данные, редкая запись
lz4	Быстрая	Средняя	Активные данные, частая запись/чтение

Проблемы и решения

Проблема	Причина	Решение
Ошибка прав при создании tablespace	Каталог принадлежит root, а не postgres	Использовать sudo chown postgres:postgres
Невозможность удалить tablespace	Зависимые базы данных используют его	Переместить БД на pg_default через ALTER DATABASE

Проблема	Причина	Решение
Двойные кавычки вместо одинарных в INSERT	Синтаксическая ошибка PostgreSQL	Использовать одинарные кавычки для строковых литералов
Не найдены TOAST-данные	Использовано неправильное имя таблицы	Найти имя через pg_class.reltoastrelid
Низкая эффективность сжатия	Высокая энтропия base32-данных	Для теста использовать данные с повторениями

Ответы на контрольные вопросы

Вопросы из задания

1. **Почему размер базы данных увеличился после создания объектов?**

   Размер увеличился из-за нескольких факторов: создание схем добавило записи в системный каталог (pg_namespace), создание таблиц добавило записи в pg_class, pg_attribute и выделило файлы для хранения данных, создание последовательностей для SERIAL добавило объекты в pg_sequence, создание индексов для PRIMARY KEY выделило дополнительное пространство, вставка данных заняла место в файлах таблиц.

2. **Как работает параметр search_path?**

   Параметр search_path определяет порядок поиска объектов базы данных при обращении по неполному имени (без указания схемы). PostgreSQL последовательно просматривает схемы в указанном порядке и использует первый найденный объект. Схема pg_catalog всегда имеет наивысший приоритет, даже если не указана явно в search_path.

3. **В чем разница между таблицей и представлением?**

   Таблица физически хранит данные на диске в файлах, занимает место, поддерживает индексы. Представление — это виртуальный объект, сохраненный SQL-запрос, не хранит данные физически, выполняется при каждом обращении, используется для упрощения запросов и обеспечения безопасности.

4. **Зачем изменять табличное пространство template1?**

   База template1 используется как шаблон при создании новых баз данных. Изменение её свойств (включая табличное пространство по умолчанию) позволяет централизованно управлять конфигурацией всех новых баз данных без необходимости явно указывать параметры при каждом CREATE DATABASE.

5. **Почему PostgreSQL не поддерживает CASCADE для DROP TABLESPACE?**

   Это сделано из соображений безопасности данных. Табличные пространства могут содержать данные целых баз данных, и автоматическое каскадное удаление было бы слишком опасным. Администратор должен явно управлять зависимостями, что предотвращает случайную потерю данных.

6. **Что такое TOAST и когда он используется?**

   TOAST (The Oversized-Attribute Storage Technique) — механизм хранения больших значений вне основной таблицы. Используется когда значение не помещается на страницу (обычно >2000 байт). Большие данные автоматически выносятся в отдельную TOAST-таблицу и разбиваются на чанки по ~2000 байт для эффективного хранения и извлечения.

7. **Почему размер базы данных больше суммы размеров таблиц?**

   Функция pg_database_size возвращает размер всего каталога базы данных, включая служебные файлы: pg_filenode.map (отображение OID), pg_internal.init (кеш системного каталога), PG_VERSION (версия). Также включается неиспользуемое зарезервированное пространство в файлах.

8. **Какой метод сжатия TOAST лучше выбрать?**

   Для большинства случаев рекомендуется lz4: быстрое сжатие и распаковка, хорошая степень сжатия, низкие накладные расходы CPU. Метод pglz подходит для: архивных данных с редким доступом, когда приоритет — максимальная степень сжатия, данных с низкой энтропией (много повторений).

Заключение

В результате выполнения лабораторной работы были подробно изучены механизмы организации данных в PostgreSQL на логическом и физическом уровнях. Приобретены практические навыки работы с базами данных, схемами и табличными пространствами. Получено понимание структуры системного каталога и способов извлечения метаинформации. Также исследованы низкоуровневые аспекты хранения, включая использование TOAST, стратегии сжатия и нежурналируемые таблицы. Также важно понимание того, как PostgreSQL реализует физическую организацию данных и какие механизмы применяет для оптимизации хранения больших объемов информации. Знание этих принципов играет важную роль в эффективном администрировании СУБД и повышении производительности. Все задания лабораторной работы выполнены полностью, включая элементы повышенной сложности («Практика+»). В процессе работы закреплены практические навыки взаимодействия с системным каталогом, табличными пространствами и механизмами низкоуровневого хранения данных.