Лабораторна робота 11. Створення користувачів та права доступу. Резервне копіювання та відновлення.

Мета: Навчити студентів створювати користувачів і ролі, а також надавати їм права доступу до різних об’єктів бази даних у PostgreSQL. Ознайомити студентів з основними підходами до резервного копіювання та відновлення даних у PostgreSQL, їхньою практичною реалізацією та особливостями в різних сценаріях.

Завдання:

1. Ознайомитися з мовою Data Control Language;
2. Виконати SQL-оператори для створення користувачів та ролей;
3. Виконати SQL-оператори для надання прав доступу для відповідних користувачів та ролей;
4. Виконати SQL-оператори для зняття прав доступу для відповідних користувачів та ролей;
5. Виконати SQL-оператор(-и) для створення резервної копії бази даних;
6. Виконати SQL-оператор(-и) для відновлення копії бази даних.

Результат:

Студенти повинні подати SQL-скрипти, що відображають створення користувачів, ролей та надання їм різних прав доступу до різних об’єктів згідно завдання та предметної області, їх опис, а також звіт з результатами тестування. А також SQL-скрипти створення резервної копії бази даних та її відновлення.

Теоретичні відомості до виконання лабораторної роботи

У СУБД PostgreSQL доступ до об’єктів бази даних контролюється через механізм користувачів і ролей. Користувачі можуть бути об’єднані в ролі, які визначають їхні права на певні дії в базі.

Основні поняття:

1. Користувачі та ролі:

Користувач (User) — це тип ролі, яка може виконувати вхід у систему бази даних.

Роль (Role) — узагальнений об'єкт для управління правами. Ролі можуть мати або не мати право входу в систему.

Користувач може входити до кількох ролей, успадковуючи їхні права.

2. Права доступу:

PostgreSQL дозволяє визначати рівні доступу до таблиць, представлень, функцій, схем тощо.

Основні права доступу включають: SELECT, INSERT, UPDATE, DELETE, EXECUTE, USAGE, ALL.

3. GRANT та REVOKE:

GRANT — використовується для надання прав доступу до об'єктів.

REVOKE — для скасування прав доступу.

Ролі

Ролі бази даних глобальні для всієї установки кластера бази даних (не для окремої бази даних). Для створення ролі використовується команда SQL:

CREATE ROLE ***ім’я*** [[WITH] ***атрибут(и)*** [...]]

Тут ***ім’я*** відповідає правилам іменування ідентифікаторів SQL: або просте, без особливих знаків, або в подвійних лапках.

Атрибути ролей

Роль бази даних може мати атрибути, що визначають її повноваження та взаємодію із системою автентифікації клієнтів.

Право підключення: LOGIN | NOLOGIN

Ці пропозиції визначають, чи дозволяється новій ролі вхід на сервер; тобто, ця роль може стати початковим авторизованим ім'ям при підключенні клієнта. Можна вважати, що роль атрибуту LOGIN відповідає користувачу. Ролі без цього атрибуту бувають корисні для управління доступом у базі даних, але це не користувачі у звичайному розумінні. За замовчуванням мається на увазі варіант NOLOGIN, за винятком виклику CREATE ROLE у вигляді CREATE USER.

Для створення такої ролі можна використовувати будь-який з варіантів:

```
CREATE ROLE ім'я LOGIN;
CREATE USER ім'я;
```

(Команда CREATE USER еквівалентна CREATE ROLE за виключенням того, що CREATE USER за замовчуванням включає атрибут LOGIN, в той час як CREATE ROLE — ні.) Пропозиція USER вважається застарілим написанням пропозиції ROLE.

Статус суперкористувача: SUPERUSER | NOSUPERUSER

Суперкористувач бази даних обходить всі перевірки прав доступу, за виключенням права на вхід в систему. Це небезпечний привілей і він не повинен використовуватися недбало. Найкраще виконувати більшу частину роботи не як суперкористувач. Для створення нового суперкористувача використовується

```
CREATE ROLE ім'я SUPERUSER
```

(Потрібно виконати з під ролі, яка також є суперкористувачем). За замовчуванням мається на увазі NOSUPERUSER.

Створення бази даних: CREATEDB | NOCREATEDB

Роль повинна явно мати дозвіл на створення бази даних. За замовчуванням мається на увазі NOCREATEDB. Для створення такої ролі використовується

```
CREATE ROLE ім'я CREATEDB.
```

Створення ролі: CREATEROLE | NOCREATEROLE

Ці пропозиції визначають, чи зможе роль створювати нові ролі (тобто виконувати CREATE ROLE). Роль із правом CREATEROLE може також змінювати та видаляти інші ролі. За замовчуванням мається на увазі NOCREATEROLE. Для створення такої ролі використовується

```
CREATE ROLE ім'я CREATEROLE.
```

Пароль: PASSWORD '*пароль*' | PASSWORD NULL

Задає пароль ролі. Якщо автентифікація за паролем не буде використовуватися, цей параметр можна опустити. При вказівці порожнього значення буде встановлено пароль NULL, що не дозволить даному користувачеві пройти автентифікацію за паролем. За бажанням пароль NULL можна встановити явно, вказавши PASSWORD NULL. Пароль вказується під час створення ролі:

CREATE ROLE ***ім'я*** PASSWORD '***рядок***'.

ADMIN ***ім'я_ролі***

Пропозиція ADMIN подібна до ROLE, але перелічені в ньому ролі включаються в нову роль з атрибутом WITH ADMIN OPTION, що дає їм право включати в цю роль інші ролі.

INHERIT | NOINHERIT

Ці пропозиції визначають, чи роль «успадковуватиме» права ролей, членом яких вона є. Без INHERIT членство в іншій ролі дозволяє лише виконати SET ROLE і перейти на цю роль; правами, призначеними іншій ролі, можна буде користуватися лише після цього. За умовчанням мається на увазі INHERIT.

VALID UNTIL 'дата_час'

Пропозиція VALID UNTIL встановлює дату і час, після якого пароль ролі перестає діяти. Якщо ця пропозиція відсутня, термін дії пароля буде необмеженим.

IN ROLE ім'я_ролі

У пропозиції IN ROLE перераховуються одна чи кілька існуючих ролей, в які буде негайно включена нова роль. (Зверніть увагу, що додати нову роль з правами адміністратора таким чином не можна; для цього потрібно окремо виконати команду GRANT.)

ROLE ім'я_ролі

У пропозиції ROLE перераховуються одна або декілька ролей, які автоматично стають членами створюваної ролі. (По суті таким чином нова роль стає "групою".)

Атрибути ролей можуть бути змінені після створення командою ALTER ROLE.

Для видалення ролі використовується команда:

DROP ROLE ***ім'я***;

Для отримання списку існуючих ролей:

SELECT rolname FROM pg_roles;

Членство в ролі

Часто буває зручно згрупувати користувачів для спрощення управління правами: права можна видавати для всієї групи та у всієї групи забирати. У PostgreSQL при цьому створюється роль, що представляє групу, а після членство у цій групі видається ролям індивідуальних користувачів.

Для налаштування групової ролі спочатку необхідно створити саму роль:

CREATE ROLE ***ім'я***;

Зазвичай групова роль не має атрибуту LOGIN, хоча за бажання його можна встановити.

Після того, як групова роль створена, до неї можна додавати або видаляти членів, використовуючи команди GRANT і REVOKE:

GRANT ***групова_роль*** TO ***роль1***, ... ;

REVOKE ***групова_роль*** FROM ***роль1***, ... ;

Членом ролі може бути й інша групова роль (бо насправді немає жодних відмінностей між груповими і негруповими ролями). При цьому база даних не допускає замикання членства у колі.

Видалення ролей

Оскільки ролі можуть володіти об'єктами баз даних і мати права доступу до інших об'єктів, видалення ролі не зводиться до негайної дії DROP ROLE. Спочатку мають бути видалені та передані іншим власниками всі об'єкти, що належать ролі; також мають бути відкликані всі права, надані ролі.

Володіння об'єктами можна передавати в індивідуальному порядку, застосовуючи команду ALTER, наприклад:

```
ALTER TABLE bobs_table OWNER TO alice;
```

Крім того, для перепризначення будь-якої іншої ролі володіння відразу всіма об'єктами, які належать ролі, яку потрібно видалити, можна застосувати команду REASSIGN OWNED. Оскільки REASSIGN OWNED не може звертатися до об'єктів в інших базах даних, її необхідно виконати в кожній базі, яка містить об'єкти, що належать цій ролі.

Після того, як всі цінні об'єкти будуть передані новим власникам, всі об'єкти, які залишаються, що належать ролі, яку потрібно видалити, можуть бути видалені за допомогою команди DROP OWNED. І ця команда не може звертатися до об'єктів в інших базах даних, тому її потрібно запускати в кожній базі, яка містить об'єкти, що належать ролі.

DROP OWNED також видаляє всі права, які дані цільовій ролі для об'єктів, що не належать їй. Так як REASSIGN OWNED такі об'єкти не торкається, зазвичай необхідно запустити і REASSIGN OWNED, і DROP OWNED (у цьому порядку!), щоб повністю ліквідувати залежність ролі, що видаляється.

З урахуванням цього, загальний синтаксис видалення ролі, яка володіла об'єктами, коротко такий:

```
REASSIGN OWNED BY doomed_role TO successor_role;
DROP OWNED BY doomed_role;
-- повторити попередні команди для кожної бази у кластері
DROP ROLE doomed_role;
```

Якщо не всі об'єкти потрібно передати одному новому власнику, краще спочатку вручну відпрацювати винятки, а на завершення виконати наведені вище дії.

При спробі виконати DROP ROLE для ролі, в якій зберігаються залежні об'єкти, буде видано повідомлення, яке говорить, які об'єкти потрібно передати іншому власнику чи видалити.

Встановити ідентифікатор поточного користувача в рамках сеансу можна за допомогою команди

SET ROLE *ім'я_ролі*.

Ім'я ролі може бути записане у вигляді ідентифікатора або строкової константи. Після SET ROLE права доступу для команд SQL перевіряються так, якби сеанс спочатку був встановлений з цим ім'ям ролі.

Вказуючи певне ім'я_ролі, поточний користувач повинен бути членом цієї ролі. (Якщо користувач сеансу є суперкористувачем, він може вибрати будь-яку роль.)

Форми NONE та RESET скидають ідентифікатор поточного користувача, тому активним стає ідентифікатор користувача сеансу. Ці форми можуть виконувати будь-які користувачі.

Приклади:

```
SELECT SESSION_USER, CURRENT_USER;
```

```
 session_user | current_user
--------------+--------------
 peter        | peter
```

```
SET ROLE 'paul';
SELECT SESSION_USER, CURRENT_USER;
```

```
 session_user | current_user
--------------+--------------
 peter        | paul
```

Привілеї

Привілеї – це те, що визначає, чи може вказаний користувач виконати дану команду. Є кілька типів привілеїв, що відповідають декільком типам операцій. Привілеї даються і скасовуються двома командами SQL: GRANT (ДОПУСК) і REVOKE (СКАСУВАННЯ).

Кожен користувач в базі даних має набір привілеїв. Ці привілеї можуть змінюватися з часом – нові додаватися, старі видалятися.

Привілеї надаються певному користувачеві у зазначеній таблиці, або базовій таблиці, або представленні. Користувач, який створив таблицю (будь-якого виду), є власником цієї таблиці, тобто має всі привілеї в цій таблиці і може передавати привілеї іншим користувачам в цій таблиці.

Привілеї, які можна назначити користувачу представлені в таблиці.

Таблиця – Привілеї, які можна назначити користувачу

Привілеї	Надані можливості
SELECT	Користувач з цим привілеєм може виконувати запити в таблиці.
INSERT	Користувач з цим привілеєм може виконувати команду INSERT в таблиці.
UPDATE	Користувач з цим привілеєм може виконувати команду UPDATE в таблиці. Можна обмежити цей привілей для певних стовпців таблиці.
DELETE	Користувач з цим привілеєм може виконувати команду DELETE в таблиці.
REFERENCES	Користувач з цим привілеєм може визначити зовнішній ключ, який використовує один або більше стовпців цієї таблиці, як батьківський ключ. Можна обмежити цей привілей для певних стовпців.

Крім того, можливі такі нестандартні привілеї об'єкта, як INDEX, який дає право створювати індекс в таблиці, SYNONYM – дає право створювати синонім для об'єкта, і ALTER – дає право виконувати команду ALTER TABLE в таблиці.

Команда GRANT ***надає привілеї***:

GRANT привілея(ї) ON таблиця TO користувач(і);

Списки привілеїв або користувачів, відокремлюваних комами, є абсолютно прийнятними, необов’язково застосовувати одиночні привілеї окремому користувачеві командою GRANT.

Всі привілеї об'єкта використовують один і той же синтаксис, крім команд UPDATE і REFERENCES в яких можна вказувати ***імена стовпців***:

GRANT UPDATE (атрибут1, атрибут2) ON таблиця TO користувач(і);

Ця команда дозволить зазначеному користувачу змінювати значення тільки в атрибут1, атрибут2 зазначеної таблиці.

REFERENCES використовується аналогічно. Надання привілею REFERENCES іншому користувачеві, дозволяє йому створювати зовнішні ключі, що посилаються на стовпці таблиці як на батьківські ключі.

SQL підтримує два аргументи для команди GRANT, які мають спеціальне значення: ***ALL PRIVILEGES*** (всі привілеї) або просто ALL і ***PUBLIC*** (загальні).

ALL використовується замість імен привілеїв в команді GRANT, щоб віддати всі привілеї в таблиці.

GRANT ALL ON таблиця TO користувач(і);

PUBLIC – більше схожий на тип аргументу «захопити всі (catch-all)», ніж на призначену для користувача привілею. У такому випадку всі користувачі автоматично отримують задану привілею. Найбільш часто, це застосовується для привілеї SELECT в певних базових таблицях або представленнях, які необхідно зробити доступними для будь-якого користувача.

GRANT привілея(ї) ON таблиця TO PUBLIC;

PUBLIC не обмежений в його передачі тільки поточним користувачам. Будь-який новий користувач, який додається до системи, автоматично отримає всі привілеї, призначені раніше всім, так що, якщо необхідно буде обмежити доступ до таблиці всім, зараз або в майбутньому, найкраще надати інші привілеї, крім SELECT, для індивідуальних користувачів.

Іноді, власнику таблиці потрібно, щоб інші користувачі могли мати такі ж привілеї в його таблиці. Зазвичай це робиться в системах, де один або більше людей створюють кілька (або всі) базові таблиці в базі даних, а потім ***передають відповідальність*** за них тим, хто буде фактично з ними працювати. SQL дозволяє робити це за допомогою WITH GRANT OPTION

GRANT привілея(ї) ON таблиця TO користувач(і) WITH GRANT OPTION;

Користувач за допомогою GRANT OPTION в особливій привілеї для даної таблиці, може, в свою чергу, надати цей привілей до тієї ж таблиці, з або без GRANT OPTION, будь-якому іншому користувачу. Це не змінює приналежності самої таблиці, як і раніше таблиця належить її власнику. Тому користувачі, які отримали права, повинні встановлювати префікс у вигляді імені власника, коли посилаються на ці таблиці

GRANT привілея(ї) ON власник.таблиця TO користувач(і);

Можна зробити дії привілеїв більш точними, використовуючи представлення. Кожного разу, коли передаються привілеї в базовій таблиці користувачеві, вони автоматично поширюються на всі рядки, а при використанні можливих винятків UPDATE і REFERENCES, на всі стовпці таблиці. Створюючи представлення, яке посилається на основну таблицю, і потім переносить привілеї на представлення, а не на таблицю, можна обмежувати ці привілеї будь-якими виразами в запиті, що містяться в представленні. Це значно покращує базисні можливості команди GRANT.

Щоб створювати представлення, необхідно мати привілей SELECT у всіх таблицях, на які є посилання в представленні. Якщо представлення модифікуються, будь-які привілеї,

INSERT, UPDATE або DELETE, які є для базової таблиці, будуть автоматично передаватися представленням. Якщо відсутній привілей на модифікацію в базових таблицях, він буде відсутній і в представленнях, які створили, навіть якщо самі ці представлення модифікуються. Так як зовнішні ключі не використовуються в представленнях, привілей REFERENCES ніколи не використовується при створенні представлень.

Видалення привілеїв виконується командою REVOKE. Синтаксис команди REVOKE схожий на GRANT, але має протилежне значення

REVOKE привілея(ї) ON таблиця FROM користувач(і);

Списки привілеїв або користувачів, відокремлюваних комами, є абсолютно прийнятними, аналогічно команді GRANT.

Привілеї скасовуються користувачем, який їх надав, і скасування буде каскадуватися, тобто воно буде автоматично поширюватися на всіх користувачів, які отримали від нього цей привілей.

Для ***створення користувача*** адміністратор бази даних надає користувачу привілею CONNECT:

GRANT CONNECT TO ім’я_користувача;

Для ***видалення користувача*** необхідно використовувати для REVOKE привілей CONNECT. Якщо зробити спробу видалити привілей CONNECT користувача, який має створені ним таблиці, команда буде відхилена, тому що її дія залишить таблицю без власника, а це не дозволяється. Перш ніж видалити його привілей CONNECT, необхідно видалити всі таблиці, створені цим користувачем. Якщо ці таблиці не порожні, то можна передати їх дані в інші таблиці за допомогою команди INSERT, яка використовує запит.

Перевірка встановлених привілей

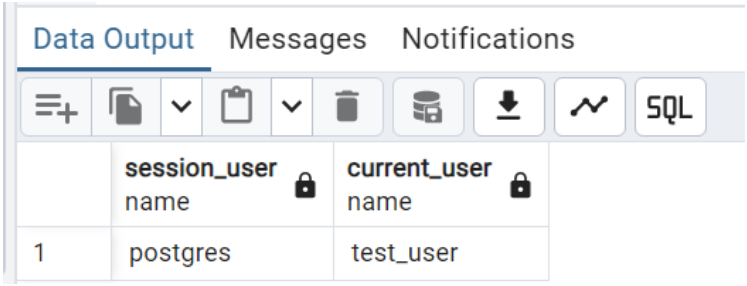
Для перевірки наданих привілеїв можна встановити ідентифікатор поточного користувача в рамках сеансу за допомогою команди

SET ROLE *ім'я_ролі*

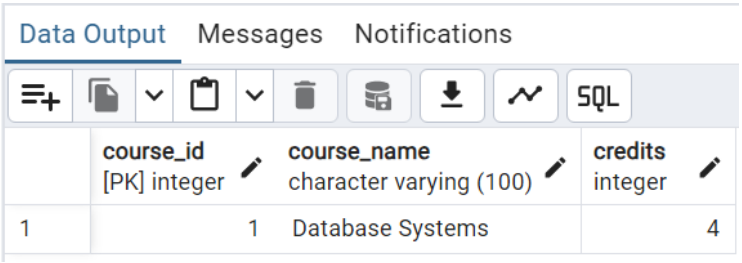
та виконати дії, що дозволені та недозволені певному користувачу.

Наприклад, користувачу test_user надано привілеї на перегляд та зміну даних тільки в таблиці «courses». Всі інші привілеї не надано. Перевіряємо:

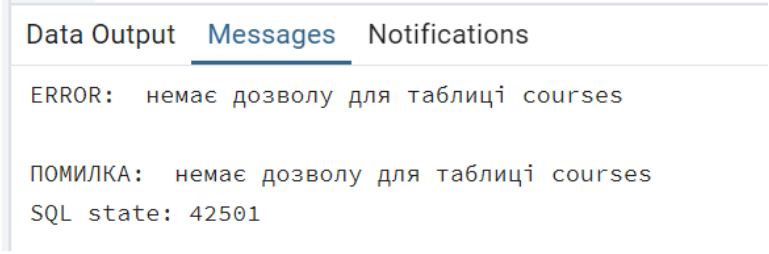
```
SET ROLE 'test_user';
SELECT SESSION_USER, CURRENT_USER;
```



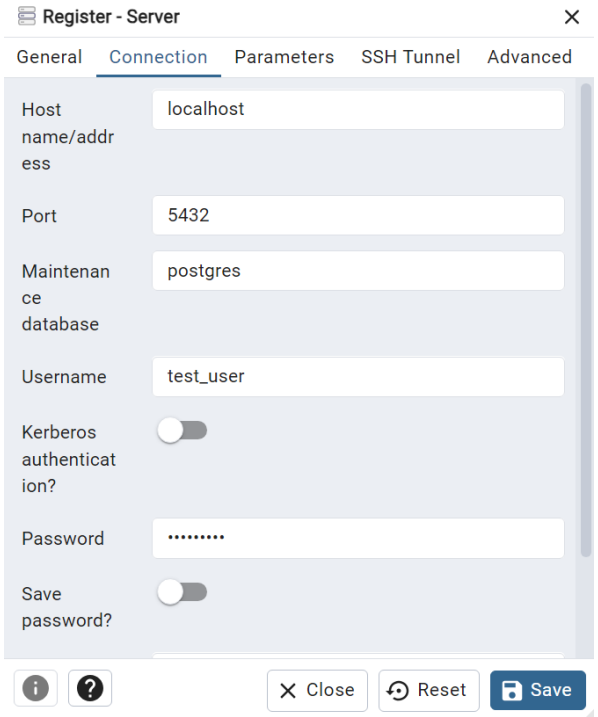

```
SELECT * FROM courses WHERE course_id=1;
```



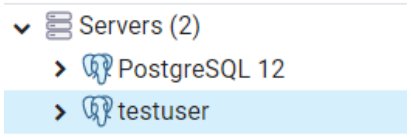
DELETE FROM courses;



Або створити в PgAdmin сервер з ім’ям та паролем створеного користувача:
Правою кнопкою на PostgreSQL XX -> Створити -> Сервер
На вкладці Загальні задаємо будь-яке ім’я серверу.
На вкладці З’єднання задаємо:
Ім’я/адреса сервера – localhost (якщо працюєте з локальною версією)
Ім’я користувача – ім’я створеного вами користувача
Пароль – пароль створеного вами користувача
Зберегти



З’явиться сервер, на якому ви можете спробувати виконати дії, що дозволені та недозволені певному користувачу.



У будь-якій базі даних дані можуть піддаватися ризикам — наприклад, через апаратні помилки, збої програмного забезпечення або людський фактор. Резервне копіювання дозволяє зберігати копії даних, щоб за потреби відновити їх. PostgreSQL підтримує кілька видів резервного копіювання, кожен з яких має свої особливості та оптимальний для різних ситуацій.

Резервне копіювання та відновлення

Типи резервного копіювання в PostgreSQL

1. **Логічне резервне копіювання**

Виконується на рівні SQL-команд за допомогою утиліт pg_dump та pg_dumpall.

Команда pg_dump в PostgreSQL — це утиліта для створення резервних копій бази даних, яка зберігає структуру та/або дані в текстовий або архівний файл. На відміну від фізичного копіювання файлів, pg_dump генерує файл резервної копії у вигляді SQL-скриптів або спеціальних архівів, що дозволяє легко відновити базу даних навіть на іншому сервері чи версії PostgreSQL.

Основні параметри pg_dump та їх значення:

Формати виведення:

-F c або --format=c: Спеціальний формат (custom) — дозволяє створити архів, який можна відновити за допомогою pg_restore. Цей формат підтримує паралельне відновлення і є гнучким для відновлення окремих об’єктів бази даних.

-F t або --format=t: Tar-архів (tar) — створює резервну копію у форматі tar, також сумісний з pg_restore. У цьому форматі можна вибірково відновлювати окремі таблиці.

-F p або --format=p: Плейн (plain) — звичайний текстовий файл SQL. Цей формат можна відновити, виконуючи команду psql, але він не підтримує паралельне відновлення.

Вибір бази даних та користувача:

-d або --dbname=DBNAME: Вказує базу даних, для якої потрібно створити резервну копію.

-U або --username=USERNAME: Ім’я користувача для підключення до бази даних.

Фільтрація даних для резервного копіювання:

-t або --table=TABLE: Створює резервну копію тільки для конкретної таблиці. Можна вказати декілька таблиць, повторюючи параметр.

-n або --schema=SCHEMA: Вибирає певну схему для копіювання. Якщо потрібно резервувати лише об’єкти в межах конкретної схеми, цей параметр буде корисним.

-T або --exclude-table=TABLE: Виключає вказану таблицю з резервної копії.

-N або --exclude-schema=SCHEMA: Виключає вказану схему.

Резервування даних та/або лише структури:

-a або --data-only: Резервне копіювання тільки даних без створення структури таблиць.

-s або --schema-only: Створює резервну копію тільки для схеми бази даних (структури) без даних.

Інші параметри та налаштування:

-j або --jobs=NUM: Виконує паралельне резервне копіювання за вказаною кількістю потоків. Цей параметр корисний для великих баз даних і працює лише зі спеціальним та tar-форматами.

--inserts: Генерує інструкції INSERT замість COPY. Це може бути корисно для сумісності з іншими базами даних, але працює повільніше, ніж COPY.

--clean: Додає команди для видалення існуючих об'єктів перед відновленням резервної копії (корисно при оновленні структури бази).

--if-exists: Використовується разом з --clean, щоб додати IF EXISTS до команд видалення.

--no-owner: Не включає команди зміни власника об'єктів. Рекомендується при відновленні на іншій базі даних або іншому сервері.

Приклади використання:

Резервне копіювання бази даних у форматі SQL

pg_dump -U user_name -d database_name -F p > backup.sql

Резервне копіювання з паралельними потоками в спеціальному форматі

pg_dump -U user_name -d database_name -F c -j 4 -f backup_file

Резервне копіювання зокрема для таблиці "students"

pg_dump -U user_name -d database_name -t students -F p > students_backup.sql

Особливості та рекомендації:

Рекомендується використовувати спеціальний (custom) або tar формат для великих баз даних, щоб спростити відновлення та отримати можливість відновлювати об’єкти окремо.

Для забезпечення повного резервного копіювання корисно поєднувати pg_dump із командою pg_dumpall, яка резервує глобальні об’єкти, такі як ролі та параметри конфігурації.

pg_dump — це потужний інструмент резервного копіювання в PostgreSQL, який дозволяє виконувати резервування з великою кількістю опцій для гнучкого налаштування.

Переваги: переносимість на інші версії PostgreSQL, можливість вибіркового копіювання (наприклад, окремих таблиць).

Недоліки: може займати багато часу для великих баз, не підтримує інкрементальне копіювання.

Команда pg_dumpall в PostgreSQL призначена для створення резервної копії всіх баз даних, що працюють на одному сервері, включаючи глобальні об’єкти, такі як ролі, параметри конфігурації та таблиці баз даних. Відмінність від pg_dump полягає в тому, що pg_dumpall захоплює не лише дані та схему, а й усі об’єкти сервера, забезпечуючи повне резервне копіювання всієї серверної конфігурації PostgreSQL.

Основні параметри pg_dumpall та їх значення

Формати виведення:

-f FILENAME або --file=FILENAME: Вказує файл для збереження резервної копії. Якщо цей параметр не вказаний, резервна копія буде створена у стандартний вихідний потік (stdout), що зручно для виконання pg_dumpall в межах інших команд, наприклад, з використанням gzip для стиснення.

Параметри автентифікації та підключення:

-h HOSTNAME або --host=HOSTNAME: Вказує хост (сервер), до якого потрібно підключитися. Це може бути IP-адреса або ім’я хосту сервера PostgreSQL.

-p PORT або --port=PORT: Вказує порт сервера PostgreSQL.

-U USERNAME або --username=USERNAME: Ім'я користувача, який підключається до сервера для виконання резервного копіювання. Рекомендується використовувати

суперкористувача, оскільки доступ до глобальних об'єктів обмежений для звичайних користувачів.

-w або --no-password: Не запитує пароль при підключенні. Це корисно для автоматизації скриптів резервного копіювання.

-W або --password: Змушує систему запитувати пароль при підключенні.

Фільтрація резервного копіювання:

--globals-only: Резервне копіювання лише глобальних об'єктів сервера, таких як ролі, права доступу та конфігурація. Всі інші дані баз даних не будуть включені.

--roles-only: Резервне копіювання лише ролей та пов'язаних з ними привілеїв, що дозволяє відновити лише користувачів без даних чи схеми баз даних.

--data-only: Резервує лише дані баз даних, без збереження структури об’єктів. Це обмеження буде застосовано до всіх баз на сервері.

Налаштування параметрів безпеки та привілеїв:

--clean: Додає команди для видалення існуючих об'єктів перед відновленням резервної копії (наприклад, DROP DATABASE). Це корисно при оновленні всього сервера баз даних.

--if-exists: Використовується разом із параметром --clean, щоб уникнути помилок при видаленні неіснуючих об'єктів.

Додаткові параметри:

-c або --create: Додає команду CREATE DATABASE для кожної бази даних у резервній копії, що дозволяє автоматично створювати бази даних під час відновлення.

-v або --verbose: Включає детальний режим виведення інформації про кожен крок резервного копіювання. Рекомендується для діагностики помилок чи моніторингу прогресу процесу.

--no-owner: Виключає команди для зміни власника об'єктів під час резервного копіювання, що може бути корисним при відновленні на іншому сервері або з іншим набором користувачів.

Приклади використання pg_dumpall:

Резервне копіювання всіх баз даних у файл SQL

```
pg_dumpall -U postgres -f full_backup.sql
```

Резервне копіювання всіх баз і глобальних об'єктів з деталізованим виведенням

```
pg_dumpall -U postgres -v -f full_backup_verbose.sql
```

Тільки ролі та привілеї

```
pg_dumpall -U postgres --roles-only -f roles_backup.sql
```

Тільки глобальні об'єкти (без баз даних)

```
pg_dumpall -U postgres --globals-only -f globals_backup.sql
```

Стиснене резервне копіювання

```
pg_dumpall -U postgres | gzip > full_backup.sql.gz
```

Рекомендації та зауваження:

Резервне копіювання глобальних об’єктів є важливим при переміщенні баз даних між серверами, оскільки воно дозволяє зберегти налаштування користувачів, ролей і прав доступу.

Використання pg_dumpall для створення повної резервної копії підходить для одноразових резервних копій або для перенесення бази на інший сервер. Для автоматизованого резервування кожної бази окремо доцільніше використовувати pg_dump.

pg_dumpall є ефективним інструментом для повного резервного копіювання сервера PostgreSQL, особливо коли потрібне збереження глобальних налаштувань, але варто враховувати його обмеження, зокрема відсутність підтримки спеціальних форматів резервного копіювання та паралельного резервування.

2. **Фізичне резервне копіювання**

Найчастіше використовується утиліта pg_basebackup, яка створює копію всіх даних бази.

Команда pg_basebackup в PostgreSQL використовується для створення фізичної резервної копії бази даних на рівні файлової системи. Вона призначена для створення повної копії всього кластера бази даних, включаючи конфігураційні файли та необхідні WAL-журнали, які забезпечують цілісність даних під час відновлення. Цей інструмент часто використовується для налаштування реплікації або створення відновлюваних резервних копій для великих обсягів даних.

Основні параметри pg_basebackup та їх значення

Параметри збереження копії та формату вихідного файлу:

-D DIRECTORY або --pgdata=DIRECTORY: Вказує каталог, у який буде збережена резервна копія. Якщо він існує, команда поверне помилку.

-F FORMAT або --format=FORMAT: Визначає формат резервної копії. Параметри:

p (plain) — звичайний каталог із файлами бази даних;

t (tar) — формат архіву tar, який можна зберегти у вигляді одного файлу та легко перемістити.

Налаштування WAL-журналів для забезпечення цілісності:

-X METHOD або --wal-method=METHOD: Визначає спосіб збереження WAL-журналів для подальшого відновлення. Значення:

fetch — завантажує всі WAL-журнали під час резервного копіювання (рекомендовано для автономних резервних копій);

stream — зберігає WAL-журнали одночасно з резервним копіюванням.

--wal-dir=DIRECTORY: Вказує каталог для збереження WAL-журналів. Цей параметр працює тільки з --wal-method=stream.

Параметри з'єднання:

-h HOSTNAME або --host=HOSTNAME: Хост сервера PostgreSQL.

-p PORT або --port=PORT: Порт, який використовується для підключення.

-U USERNAME або --username=USERNAME: Ім'я користувача для підключення (часто потрібен користувач з правами REPLICATION).

Додаткові параметри для зручності та контролю процесу:

-R або --write-recovery-conf: Автоматично створює файл recovery.conf у каталозі резервної копії. Це важливо для налаштування реплікації на основі резервної копії.

--no-password: Забороняє запит пароля, що корисно для автоматизації резервного копіювання.

-P або --progress: Включає виведення інформації про прогрес резервного копіювання (відсоток виконання), що особливо корисно для великих баз даних.

--checkpoint=CHECKPOINT: Визначає метод контрольної точки під час резервного копіювання (значення: fast для прискореного завершення всіх транзакцій або spread для поступового зниження навантаження на систему).

-c або --checkpoint-timeout=TIMEOUT: Встановлює максимальний час очікування для створення контрольної точки.

Приклади використання pg_basebackup**:**

Створення резервної копії в стандартний каталог

pg_basebackup -U replicator -h localhost -D /backup/db_backup -Fp -P -X stream

Створення резервної копії у форматі tar з WAL-журналами в окремий каталог

pg_basebackup -U replicator -h localhost -D /backup/db_backup.tar -Ft -X stream --wal-dir=/backup/wal

Створення резервної копії для реплікації

pg_basebackup -U replicator -h localhost -D /var/lib/postgresql/backup -R -X stream

Рекомендації та особливості pg_basebackup

Параметри -X stream і -X fetch забезпечують різні способи захоплення WAL-журналів, але при налаштуванні реплікації рекомендується stream, оскільки він дає змогу захоплювати WAL у реальному часі.

Стиснення резервної копії можна досягти за допомогою зовнішніх інструментів, наприклад, gzip або lz4, оскільки pg_basebackup не має вбудованих параметрів стиснення.

Моніторинг прогресу з параметром -P дуже важливий для великих баз даних, оскільки резервне копіювання може тривати довго.

pg_basebackup — зручний інструмент для створення резервних копій, налаштування реплікації та швидкого відновлення сервера баз даних PostgreSQL.

Переваги: швидше від логічного резервного копіювання для великих баз, підтримка інкрементального резервного копіювання за допомогою WAL-архівів.

Недоліки: залежить від версії PostgreSQL та конфігурації системи.

3. **Інкрементальне резервне копіювання за допомогою WAL-архівів**

PostgreSQL записує всі зміни до журналів відновлення (WAL), які можна зберігати для інкрементального відновлення бази.

Переваги: дозволяє відновлення до точного моменту часу (point-in-time recovery).

Недоліки: вимагає регулярного копіювання WAL-файлів для мінімізації втрат даних.

Відновлення даних

Відновлення з логічного резервного копіювання: виконується за допомогою psql або іншої утиліти, що може обробити SQL-скрипт.

Команда pg_restore в PostgreSQL використовується для відновлення бази даних із архіву, створеного за допомогою команди pg_dump у форматах custom або directory. Цей інструмент дозволяє вибіркове відновлення окремих таблиць, схем або об’єктів бази даних, а також підтримує паралельне відновлення для підвищення швидкості на багатоядерних системах.

Основні параметри pg_restore та їх значення

Основні параметри для роботи з базою даних:

-d DBNAME або --dbname=DBNAME: Вказує базу даних, до якої здійснюється підключення для відновлення. Цей параметр обов’язковий, якщо відновлення проводиться безпосередньо в базу даних.

-h HOSTNAME або --host=HOSTNAME: Хост сервера PostgreSQL, до якого підключається pg_restore.

-p PORT або --port=PORT: Порт для підключення до сервера PostgreSQL.

-U USERNAME або --username=USERNAME: Ім'я користувача, який має права доступу для відновлення.

Формат і структура відновлення:

-F FORMAT або --format=FORMAT: Визначає формат резервного файлу. Значення:

c — custom (спеціальний формат, створений за допомогою pg_dump -Fc);

d — directory (каталог із файлами, створений за допомогою pg_dump -Fd);

t — tar (архів tar, створений за допомогою pg_dump -Ft).

-j NUM або --jobs=NUM: Задає кількість паралельних завдань для відновлення. Працює лише для форматів directory та custom, де дозволяє розподіляти навантаження на кілька процесорів для пришвидшення.

Фільтрація об'єктів для вибіркового відновлення:

-n NAME або --schema=NAME: Відновлює тільки зазначену схему.

-t TABLE або --table=TABLE: Відновлює тільки вказану таблицю (можна зазначати кілька разів для кількох таблиць).

-L FILE або --use-list=FILE: Використовує список об’єктів із файлу (зазвичай генерується за допомогою pg_dump з параметром -l), щоб відновити тільки зазначені в цьому файлі об’єкти.

Опції відновлення конфігурації:

--clean: Перед відновленням видаляє об’єкти, якщо вони вже існують у базі даних.

--create: Створює базу даних перед її відновленням (якщо вона не існує).

--if-exists: Додає умову IF EXISTS для команд DROP, щоб уникнути помилок, якщо об’єкти не існують.

-c або --no-comments: Ігнорує коментарі у відновленні.

-S ROLE або --superuser=ROLE: Визначає роль суперкористувача, яка використовується для створення об’єктів, що вимагають розширених прав доступу.

Логування та налагодження процесу:

-v або --verbose: Виводить детальні повідомлення про процес відновлення.

--no-data-for-failed-tables: У випадку помилки пропускає відновлення даних для таблиць, які не вдалося створити.

--disable-triggers: Вимикає тригери під час відновлення (тільки для суперкористувачів), що корисно для прискорення імпорту даних.

Приклади використання pg_restore**:**

Відновлення повної бази даних із файлу у форматі custom

pg_restore -U postgres -d mydb /backup/mydb.backup -F c -v

Відновлення вказаної таблиці з файлу у форматі directory

pg_restore -U postgres -d mydb -t students -Fd /backup/mydb_dir -v

Паралельне відновлення з використанням 4 потоків

pg_restore -U postgres -d mydb /backup/mydb.backup -j 4 -v

Використання списку об’єктів для вибіркового відновлення

pg_restore -U postgres -d mydb -L /backup/restore_list.txt –v

Додаткові рекомендації та особливості використання pg_restore

Використання параметра -j значно підвищує продуктивність на багатоядерних системах, але рекомендується використовувати його тільки для великих баз даних у форматах custom або directory.

Звітність процесу за допомогою -v забезпечує більше деталей під час відновлення, що корисно для діагностики можливих проблем.

Параметр --if-exists дозволяє уникнути проблем із уже існуючими об'єктами, що може допомогти при відновленні частин бази даних або при інтеграції з іншими базами.

pg_restore дозволяє гнучко та детально контролювати процес відновлення бази даних, забезпечуючи вибіркове відновлення окремих об’єктів або повної бази даних.

Відновлення з фізичного резервного копіювання: вимагає зупинки бази даних і заміни файлів даних з резервної копії. Після цього можна застосувати WAL-журнали для відновлення змін.

Point-in-Time Recovery (PITR): процес відновлення з фізичної копії разом з WAL-файлами для повернення бази до стану на точний момент часу.

1. Point-in-Time Recovery (PITR)

Скопіюйте файли WAL у визначений каталог, а потім налаштуйте параметр restore_command у конфігураційному файлі PostgreSQL (postgresql.conf) для застосування WAL-файлів.

2. Моніторинг стану резервного копіювання та відновлення

Для логічного резервного копіювання корисно стежити за журналами PostgreSQL.

У випадку PITR і фізичного резервного копіювання, потрібно переконатися, що всі WAL-журнали цілі.

Команди резервного копіювання та відновлення вводяться в звичайному командному рядку. Якщо в PostgreSQL не встановлено у змінних середовища PATH, то потрібно перейти до каталогу з утилітами, або задавати шлях до утиліт. Наприклад:

```
"C:\\Program Files\\PostgreSQL\\16\\bin\\pg_dump.exe" -U postgres -d test -f > bachup.sql;
```

PgAdmin дозволяє створити резервну копію та відновлення тільки з контекстного меню (клікнувши правою кнопкою миші).

Структура звіту до лабораторної роботи

Для кожного з запитів представити:

1) словесна постановка задачі, що вирішується;
2) SQL-код рішення;
3) скриншот отриманого результату.

1. Створіть трьох користувачів.
2. Надайте право першому користувачу на зміну декількох стовпців будь-якої таблиці.
3. Надайте право всім користувачам системи на перегляд будь-якої таблиці.
4. Надайте право другому користувачу вставляти або модифікувати значення таблиці з правом передавати іншим користувачам вказані права.
5. Надайте третьому користувачу права адміністратора (всі права на всі таблиці).
6. Зніміть привілей INSERT для третього користувача для будь-якої таблиці.
7. Надайте право першому користувачу на доступ тільки до певних рядків будь-якої таблиці.
8. Розподіліть інші привілеї на свій розсуд.
9. Виконайте логічне резервне копіювання бази даних, використовуючи відповідну утиліту.
10. Виконайте відновлення з логічного резервного копіювання бази даних.
11. Виконайте фізичне резервне копіювання бази даних.
12. Виконайте відновлення з фізичного резервного копіювання бази даних.

Звіт до лабораторної роботи 11 можна здати онлайн на сайті ДО edu.op.edu.ua до початку вашого заняття.