Лабораторна робота 1. Розробка вимог до інформаційної системи за допомогою користувацьких історій. Знайомство з СУБД PostgreSQL

Мета:

Ознайомити студентів з процесом розробки вимог до інформаційної системи для заданої предметної області, а також зі встановленням та налаштуванням СУБД PostgreSQL.

Завдання:

1. Вивчення предметної області: описати предметну область, на основі якої буде створюватися база даних. Зазначити користувачів системи, задачі, які вона має вирішувати, вхідні та вихідні дані.
2. Розробити користувацькі історії (User Story). Мінімум 3 користувача та по 3 історії від кожного користувача (тобто мінімум 9 історій користувачів).
3. Встановити СУБД PostgreSQL. Перевірити його роботу, запустивши сервер.
4. Створити базу даних в PgAdmin, яка має назву латинськими літерами «ПрізвищеСтудента_pg».
5. Створити базу даних в psql, яка має назву латинськими літерами «ПрізвищеСтудента_psql».

Результат:

1. Студенти повинні подати опис заданої предметної області, тобто конкретизувати, хто буде працювати з майбутньою інформаційною системою, які задача вона буде вирішувати, які дані потрібно зберігати та які дані отримувати у результаті роботи.
2. Студенти повинні представити мінімум 9 користувацьких історій за своєю темою.
3. Студенти повинні представити скриншот створеної бази даних з відповідним ім’ям в PgAdmin.
4. Студенти повинні представити скриншот створеної бази даних з відповідним ім’ям в psql.

Теоретичні відомості до виконання лабораторної роботи

Основні терміни

Інформаційна система — це сукупність апаратного та програмного забезпечення, даних, процедур і людських ресурсів, що призначені для збору, зберігання, обробки, аналізу та передачі інформації з метою підтримки прийняття рішень, управління та організації процесів у певній предметній області.

База даних — це організована сукупність структурованих даних, яка зберігається в електронному вигляді та управляється системою управління базами даних (СУБД). Бази даних дозволяють ефективно зберігати, шукати, маніпулювати і аналізувати великі обсяги інформації, що робить їх ключовим компонентом багатьох сучасних програмних систем.

Предметна область — це частина реального світу, яка моделюється за допомогою бази даних. Вона включає всі об'єкти, процеси, події та їхні взаємозв'язки, які мають бути відображені у базі даних для виконання певних завдань. Визначення та розуміння предметної області є одним із перших кроків у проєктуванні бази даних. Воно допомагає зрозуміти, які дані необхідно зберігати, як ці дані взаємодіють між собою, та як їх найкраще організувати для ефективного управління і використання.

Система управління базами даних — це програмне забезпечення, яке дозволяє створювати, управляти та маніпулювати базами даних. СУБД забезпечує інтерфейс між користувачами або програмами і самою базою даних, дозволяючи ефективно зберігати, витягувати та змінювати дані.

Проєктування бази даних

Проєктування бази даних є критичним етапом у створенні інформаційної системи. Починати потрібно з ретельного планування і аналізу вимог, щоб забезпечити, що база даних буде ефективною, гнучкою та легкою в обслуговуванні. Основні етапи процесу проєктування бази даних включають:

1. Збір та аналіз вимог: зрозуміти потреби користувачів і бізнес-вимоги; визначити предметну область, яку буде охоплювати база даних; визначити, які дані потрібно зберігати, зв'язки між ними, і як вони будуть використовуватися.
2. Побудова концептуальної моделі.
3. Створення логічної моделі.
4. Проєктування фізичної моделі.
5. Реалізація та тестування.

Пункти 2-5 будуть вивчатися на наступних лабораторних заняттях.

Підходи до проєктування баз даних

При проєктуванні баз даних використовують різні підходи, кожен із яких підходить для певних типів проєктів та вимог. Нижче розглянемо найпопулярніші з них:

1. Проєктування на основі користувацьких історій (User story). Цей підхід часто використовується в Agile-процесах розробки програмного забезпечення, де акцент ставиться на потребах кінцевих користувачів. Більш детально цей підхід розглянемо нижче.
2. Проєктування на основі вимог (Requirements-driven design).

Цей підхід починається зі збору функціональних і нефункціональних вимог до системи. На основі вимог визначаються: які дані потрібно зберігати; як ці дані будуть взаємодіяти між собою; які обмеження мають бути встановлені.

Цей підхід корисний для проєктів, де важливо детально задокументувати всі вимоги та побудувати базу даних згідно з технічними вимогами.

3. Проєктування на основі даних (Data-driven design).

У цьому підході фокус робиться на структурі даних і їх оптимальній організації. Він підходить для ситуацій, коли дані мають складну структуру або коли вони вже визначені, і їх потрібно просто ефективно організувати в базі даних.

Цей підхід часто використовується в системах, де обробляються великі обсяги даних, або в аналітичних системах.

4. Проєктування на основі процесів (Process-driven design).

Проєктування бази даних орієнтується на бізнес-процеси організації. Спочатку визначаються процеси, які мають бути автоматизовані, а потім створюється база даних для підтримки цих процесів.

Цей підхід підходить для організацій, де база даних є частиною комплексної системи управління бізнес-процесами.

5. Проєктування на основі об'єктів (Object-oriented design).

Цей підхід використовується в поєднанні з об'єктно-орієнтованими методами програмування. Він передбачає, що структура бази даних відображає об'єктно-орієнтовану модель системи.

Такий підхід часто використовується в системах, де база даних є частиною великого об'єктно-орієнтованого програмного середовища.

6. Проєктування на основі подій (Event-driven design).

Цей підхід орієнтований на обробку подій у системі. База даних проєктується таким чином, щоб забезпечити підтримку дій та реакцій на події.

Цей підхід використовується в системах реального часу або в системах, де необхідно оперативно реагувати на певні події.

Кожен з цих підходів має свої переваги та застосовується в залежності від специфіки проєкту, але незалежно від методу, важливо забезпечити послідовність і логічність на

кожному етапі проєктування. Важливо розуміти цілі проєкту та вимоги до даних, щоб вибрати найбільш відповідний підхід для проєктування бази даних.

Проєктування бази даних на основі користувацьких історій

Як писати User Story

User Story – це коротке, просте описання функціональності системи з точки зору кінцевого користувача.

User Story – це не вимоги, вони відрізняються рядом тонких, але критичних нюансів:

– вони не повинні бути детальним описом вимог (тобто того, що система мала б робити), а є скоріше обговорюваним поданням наміру (потрібно зробити щось на зразок цього);

– вони повинні бути короткими та легко читатися, зрозумілими розробникам, стейкхолдерам та користувачам;

– вони являють собою невеликі збільшення цінної функціональності, яка може бути реалізована в межах декількох днів або тижнів;

– вони відносно легко піддаються оцінюванню, таким чином, зусилля, необхідні для реалізації, можуть бути швидко визначені;

– вони не займають величезних, громіздких документів, а скоріше організовані у списки, які легше впорядкувати та переупорядкувати під час надходження нової інформації;

– вони недеталізовані на самому початку проєкту, а вже детальніше розробляються «точно в термін», уникаючи таким чином занадто ранньої визначеності, затримок у розробці, нагромадження вимог та надмірно обмеженого формулювання рішення;

– вони вимагають мінімум або взагалі не вимагають супроводу і можуть бути безпечно скасовані після імплементації.

Структура User Story

Текст *User Story* повинен пояснювати роль/дії користувача в системі, його потребу та вигоду, яку користувач отримає після того, як історія здійсниться. **Наприклад: Як, <роль/персонаж користувача>, я <щось хочу отримати>, <з такою метою>. Тобто,**

1) є один actor;

2) є одна дія;

3) є одна цінність.

Наприклад, складемо користувацьку історію „бажання користувача Amazon.com“. Пробний варіант виглядає так: „Мені як споживачеві потрібен найбільший у світі магазин книг, де я можу купити будь-яку книгу в будь-який час“. Цей опис цілком відповідає характеру Amazon, але історія вийшла надто розпливчастою, щоб із нею можна було щось зробити. Потрібно фрагментувати нашу історію. Зробити її дійсно дуже конкретною та функціональною. Зразки історій користувача, які можна написати, маючи на увазі книжковий інтернет-магазин:

Як споживачеві мені зручно шукати книги за жанрами, щоб швидко знайти ті, які я люблю читати.

Як споживач я, відбираючи книги для покупки, хочу класти одразу кожну у кошик.

Як менеджер з випуску нової продукції, я хочу мати можливість відстежувати покупки наших клієнтів, щоб знати, які книги їм можна пропонувати.

Ось професійно зроблені побажання користувача, характер яких група має взяти до уваги».

Actor

Необхідно виділити персони чи ролі і вписувати їх у початок історії. Є одна проблема. Якщо історія нічого не втратила, коли прибрали частину історії про *actora*, – значить ця частина марна.

Якщо ролей в системі визначено не дуже багато – Користувач, Оператор та Адмін, створено по 100 історій, які починаються як “Як Користувач Я …”, а також закриваються кілька задач, історії яких починаються однаково, це марно. Джеф Паттон пропонує наступне:

1) розділити усіх *actorів* на групи. Цільова група, важлива група, менш важлива група тощо;

2) дати унікальні назви *actoram* у цих групах. Навіть якщо у системі вони будуть мати однакові ролі “Користувача системи”;

3) писати історії з точки зору цих *actorів*, вказуючи їх унікальні назви;

4) у результаті можна візуально побачити, які історії необхідні для *actorів* цільової групи, які – для кожної групи тощо. В цьому випадку є можливість більш правильно вибудувати пріоритет, оскільки історії *actorів* цільової групи є більш важливими.

Дія

Дія – це суть історії, “що потрібно зробити”. Що можна покращити. Дія має бути одна – основна. Немає сенсу описувати “авторизується та виконується пошук” або “вказує параметри пошуку та виконує пошук”. Вкажіть ту дію, яка вам дійсно потрібна.

Важливо описувати історію лише на рівні “ЩО” робить, а не “ЯК” Це головне в історії. Опишіть проблему, а не її вирішення.

Цінність

Головна проблема з *User Story*. Завжди легко вказати першу частину історії, але завжди складно вказати для чого це робиться.

Не слід відмовлятися від формулювання “щоб”. Для якихось історій можна вказати цінність історії у такому форматі, але не для більшості.

Можна перейти з поняття цінності (*value*) на вплив (*impact*). Історія не обов’язково повинна мати цінність, але обов’язково має впливати на того *actora*, що вказаний в історії. А вже цей вплив веде зрештою до мети, яка має цінність.

Уявімо, створили історію “Як інвестиційний аналітик я отримую звіт № 17 про інвестиції, щоб ШВИДШЕ ухвалити рішення”.

Як поміряти таку цінність? Як зрозуміти, що аналітик ухвалив рішення швидше? Як зрозуміти, що історія виконана?

Переробимо історію на вплив – “Як інвестиційний аналітик я отримую звіт №17 про інвестиції ШВИДШЕ”. Тобто, зараз цей звіт формується за 60 с. Ви вказуєте у вимогах, що звіт має формуватися за 15 с. Наприкінці зрозуміло чи виконано вимогу, зрозуміло, який вплив здійснено на роботу аналітика.

Найбільш поширені помилки при написанні історій

Історія для користувача

Приклад: «Як користувач я хочу управляти рекламними оголошеннями, щоб видаляти застарілі або помилкові оголошення».

На перший погляд, у цій історії не має жодних вад – усі елементи на місці. Але для буде розроблятися ця функція і що цей користувач знає про управління оголошеннями? Він адміністратор порталу оголошень, якому потрібно час від часу чистити базу та премодерувати оголошення? Чи, може, він рекламодавець, якому потрібно переглядати список створених ним оголошень і мати можливість видаляти непотрібні оголошення прямо з цього списку?

Ви могли помітити, що у цих двох користувачів зовсім різні ролі, з різними очікуваннями, з різними вимогами до системи. Основна помилка цієї історії – ігнорування ролі та персони користувача.

Історія для девелопера

Приклад: «Як девелопер, я хочу замінити віджет папок, щоб у мене був найкращий віджет папок».

Часто такі історії стають частиною технічного завдання, яке складається з переходу на оновлені версії фреймворків та бібліотек, рефакторингу (контрольований процес покращення

коду, без написання нової функціональності) тощо. У них є повне право на те, щоб бути зробленими, але на словах вони не представляють жодної цінності для користувача і досить важко буде змусити *Prodact Owner* (власник продукту) купити їх. До того ж будь-яка історія повинна створювати користувацьку цінність.

Можна переписати цю історію з точки зору користувача: «Як рекламодавець, я хочу, щоб система дозволяла створювати мені папки, щоб я міг швидше працювати з великими списками оголошень».

Технічні завдання цієї історії можуть виглядати так:

– «Відрефакторити механізм додавання папок, щоб дозволяти створювати вкладені папки до 3 рівня вкладеності»;

– «Проапдейтувати версію *SDK* для використання нового механізму локального зберігання даних на пристрої».

При цьому до такої історії набагато простіше написати критерії приймання:

– «Користувач може створювати папки із трьома рівнями вкладеності».

– «Користувач не може створити більше 100 папок».

Жодної бізнес цінності для користувача

Приклад: «Як рекламодавець, я хочу, щоб я мав можливість фільтрувати оголошення».

У нас є роль, є потреба, але причина чи бізнес цінність кудись зникли. Навіщо рекламодавцеві фільтрувати оголошення? Чого він хоче досягти? Історія справді втрачає сенс без потрібних елементів.

Жодних критеріїв приймання

У кожному з прикладів, наведених вище, крім історії для девелопера, немає критеріїв приймання. Історії можуть провалювати тести, або тест кейси можуть перевіряти не ті критерії через відсутність розуміння того, як має виглядати кінцевий результат та яким вимогам він має відповідати. Критерії приймання потрібні саме для того, щоб розсіяти помилкові припущення, а іноді навіть перепланувати історію або розбити її на менші.

Практичні поради щодо написання історій користувача:

– краще написати багато менших історій, ніж кілька громіздких;

– кожна історія в ідеалі має бути написана уникаючи технічного жаргону – щоб клієнт міг пріоритезувати історії та включати їх в ітерації;

– історії мають бути написані таким чином, щоб їх можна було протестувати;

– тести мають бути написані до коду;

– якомога довше варто уникати прив’язки до інтерфейсу. Історія має виконуватися без прив’язки до конкретних елементів;

– кожна історія має містити оцінку;

– історія має мати кінцівку – тобто призводити до конкретного результату;

– історія має вміщатися в ітерацію.

Сумнівні практики:

– занадто формальні/занадто деталізовані завдання. Іноді власники продукту з найкращими намірами прагнуть писати надто детальні історії. Якщо на зустрічі з планування ітерації команда бачить набір історій, що виглядає як багатотомна специфікація, то спокуса припустити, що всі деталі відмінно освітлені та пропустити обговорення, дуже велика;

– не пропускайте обговорення. Історії для цього і обговорюються на плануванні ітерації, щоб розкрити незрозумілі моменти, уточнити всі деталі та отримати повне уявлення про завдання\;

– не видавайте технічні завдання за історії. Якщо необхідно втиснути у формат історії технічні завдання, то наприкінці ітерації точно не з’явиться готовий шматочок продукту. Якщо

технічні завдання дійсно потрібні і мають цінність для будь-яких користувачів (у тому числі й внутрішніх, у команді), то сміливо оформлюйте їх у технічні історії, але не складайте всю ітерацію з подібних завдань, адже у вас, найімовірніше, є бізнес-клієнти.

Приклад розробки User Story

Отже, історії:

1. Як користувач, я можу зберігати свої фотографії в системі, щоб мати можливість показати або продати їх іншим користувачам.

2. Як рекламодавець, я можу розміщувати свою рекламу в системі, орієнтовану на користувачів.

3. Як адміністратор, я можу керувати фотографіями користувачів, щоб контент сайту був легальним.

Під час обговорення першої історії, замовник та команда приходять до того, що користувачі системи мають бути авторизовані системою перед виконанням будь-яких дій із фотографіями. Це призводить до появи нової користувацької ролі «гостя» – групі людей, які неавторизовані системою або взагалі поки що не мають облікового запису користувача.

4. Як гість, я можу зареєструватися в системі для отримання облікового запису користувача та подальшої роботи.

5. Як гість, я можу увійти до системи під раніше створеним обліковим записом для подальшої роботи.

Користуючись принципом симетричності вимог, команда та замовник приймають рішення, що користувач повинен мати можливість видалити свій обліковий запис у разі потреби:

6. Як користувач, я можу видалити свій обліковий запис і перестати бути користувачем системи.

Обговорюючи концепцію облікових записів, народжуються також такі історії:

7. Як користувач, я можу змінити дані свого облікового запису.

8. Як користувач, я можу зробити деякі поля облікового запису видимими для інших користувачів.

Система управління базами даних PostgreSQL

PostgreSQL — це одна з найпотужніших і найбільш популярних СУБД у світі. Вона відзначається надійністю, гнучкістю та розширюваністю, що робить її вибором багатьох компаній і розробників для створення складних інформаційних систем.

PostgreSQL бере свій початок у 1986 році як частина проєкту POSTGRES в Каліфорнійському університеті в Берклі, під керівництвом Майкла Стоунбрейкера. Спочатку POSTGRES був продовженням системи Ingres і став основою для нових рішень у галузі управління базами даних. У 1996 році проєкт був перейменований у PostgreSQL, коли до нього було додано підтримку SQL, стандартної мови для управління реляційними базами даних.

Основні характеристики:

Об'єктно-реляційна СУБД: PostgreSQL підтримує як реляційну модель, так і об'єктно-орієнтовані можливості, що дозволяє працювати зі складними типами даних та ієрархічними структурами.

Відкрите програмне забезпечення: PostgreSQL є відкритим програмним забезпеченням з ліцензією PostgreSQL License, що дозволяє вільно використовувати, змінювати та розповсюджувати її безкоштовно.

Масштабованість і продуктивність: PostgreSQL підтримує масштабування як у вертикальному (в межах одного сервера), так і горизонтальному (шардінг) напрямках, що дозволяє працювати з великими обсягами даних і високими навантаженнями.

Широка підтримка стандартів SQL: PostgreSQL підтримує більшість стандартів SQL, що забезпечує сумісність з багатьма іншими СУБД та інструментами.

Розширюваність: Завдяки модульній архітектурі, PostgreSQL дозволяє додавати нові типи даних, функції, індекси та інші компоненти, що дозволяє адаптувати її до специфічних потреб проєкту.

Архітектура PostgreSQL:

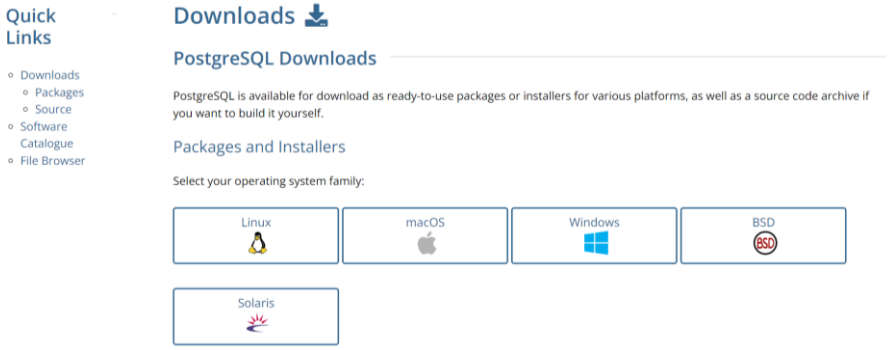
- Серверна частина: Ядро СУБД, яке відповідає за виконання запитів, управління транзакціями, зберігання даних та їхнє відновлення після збоїв.

- Клієнтська частина: PostgreSQL надає кілька інтерфейсів для роботи з базою даних, таких як командна оболонка `psql`, драйвери для мов програмування (наприклад, для Python, Java), та графічні інтерфейси (pgAdmin).

PostgreSQL використовується в багатьох галузях, включаючи фінанси, охорону здоров'я, електронну комерцію, геоінформаційні системи тощо. Завдяки своїй розширюваності, стабільності та відповідності стандартам, вона ідеально підходить для створення як невеликих, так і великих, високонавантажених систем.

Встановлення СУБД PostgreSQL

Завантажити СУБД PostgreSQL можна з офіційного сайту за адресою https://www.postgresql.org/download/. Оберіть відповідну операційну систему



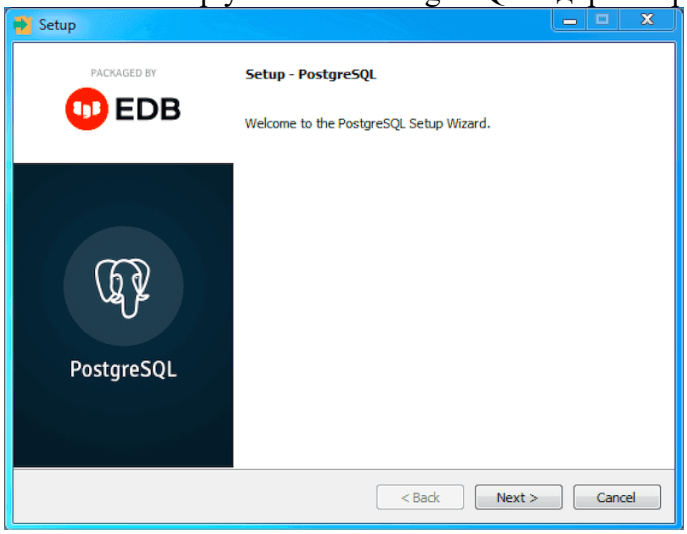
Встановлення СУБД PostgreSQL на операційну систему Windows

Для виконання установки за допомогою графічного майстра установки потрібні права суперкористувача або адміністратора.

Якщо використовується графічний майстер установки для оновлення системи, інсталятор зберігає параметри конфігурації, зазначені під час попереднього встановлення.

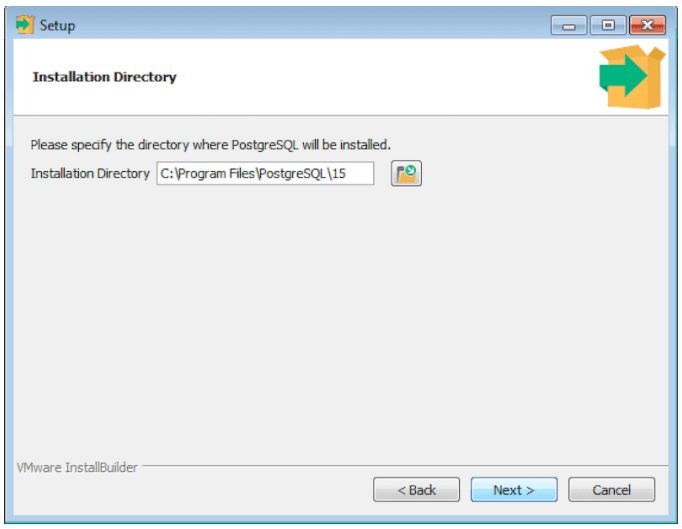
Щоб запустити майстер установки, надайте достатні права та двічі клацніть піктограму інсталятора. Якщо буде запропоновано, введіть пароль. (У деяких версіях Windows для виклику інсталятора з правами адміністратора необхідно вибрати Запуск від імені адміністратора в контекстному меню піктограми інсталятора.)

Майстер установки PostgreSQL відкриє екран вітання.



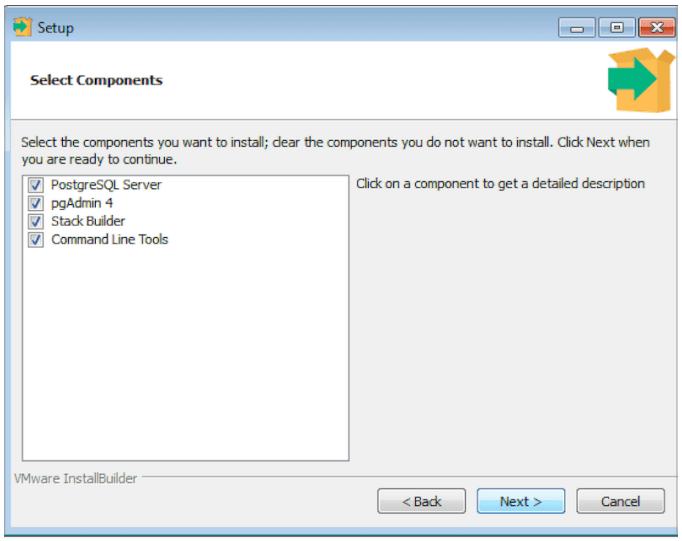
Виберіть Далі.

Відкриється вікно "Каталог установки".



Прийміть каталог установки за промовчанням або вкажіть місцезнаходження. Виберіть Далі.

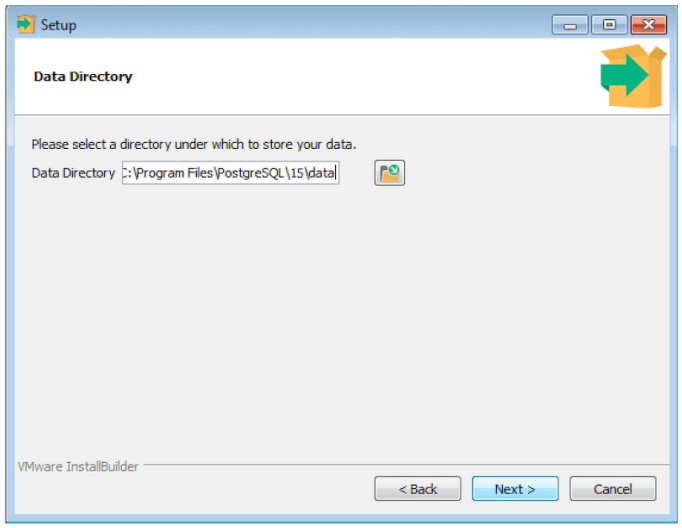
Відкриється вікно "Вибір компонентів".



Використовуйте параметри у вікні «Вибір компонентів», щоб вибрати компоненти програмного забезпечення для встановлення. Виберіть:
PostgreSQL Server для встановлення сервера бази даних PostgreSQL.
pgAdmin 4 для установки pgAdmin 4. Ця опція доступна для PostgreSQL версії 13.0 та пізніших версій.
Stack Builder для встановлення утиліти Stack Builder.

Виберіть Далі.

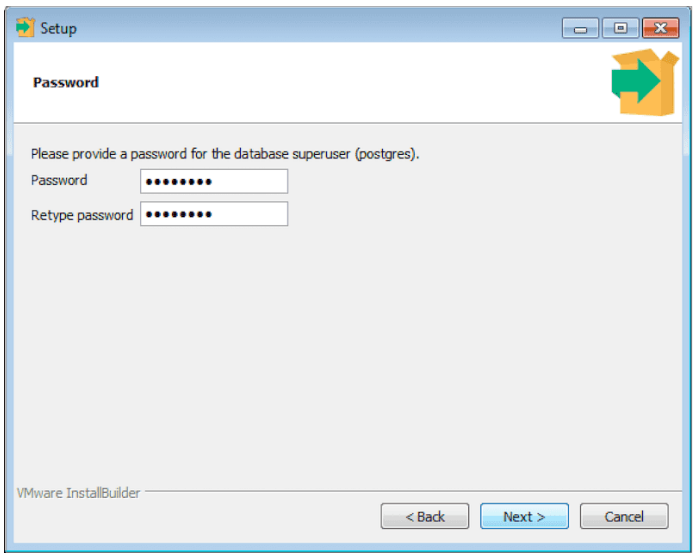
Відкриється вікно "Каталог даних".



Прийміть місцезнаходження за замовчуванням або вкажіть каталог, в якому потрібно зберігати файли даних. Виберіть Далі.

Відкриється вікно "Пароль".

PostgreSQL використовує пароль, вказаний у вікні "Пароль", як для суперкористувача, так і для облікового запису служби PostgreSQL.

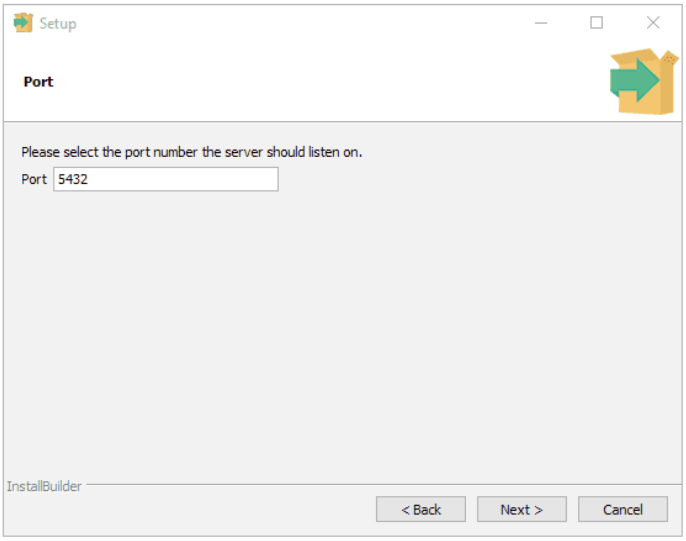


PostgreSQL працює як служба у фоновому режимі. Обліковий запис служби PostgreSQL називається postgres. Якщо вже створено обліковий запис служби під назвою postgres, необхідно вказати той же пароль, що й існуючий пароль для postgres облікового запису служби.

Зазначений пароль повинен відповідати всім політикам безпеки на хості PostgreSQL.

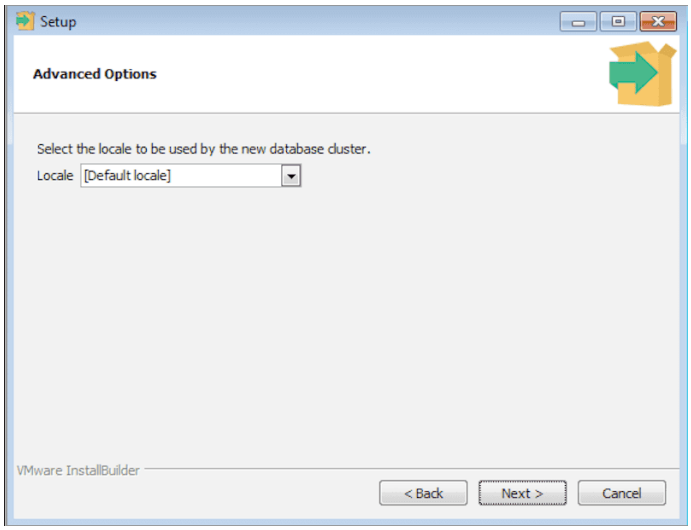
Введіть пароль у полі Пароль та підтвердьте пароль у полі Повторіть пароль. Виберіть Далі.

Відкриється вікно Порт.



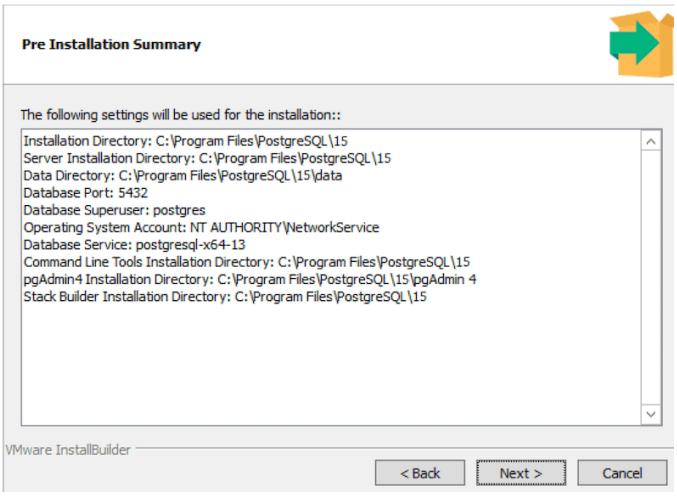
У полі Порт введіть номер порту, який прослуховує сервер. Стандартний порт прослуховувача — 5432. Виберіть Далі .

Відкриється вікно "Додаткові параметри".



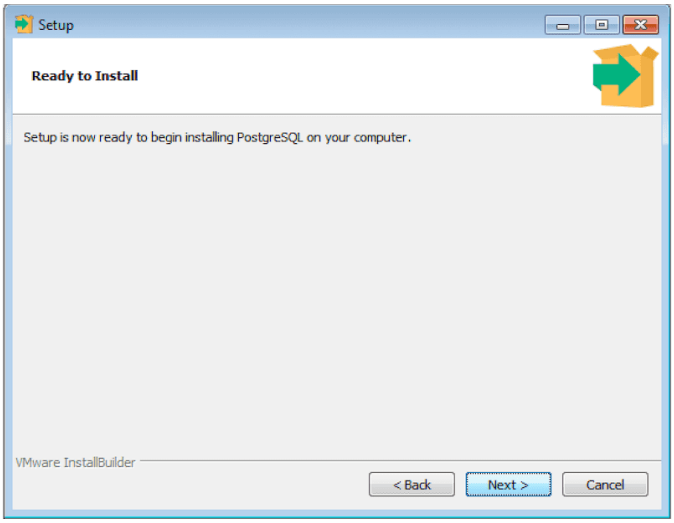
У полі Locale вкажіть локаль нового кластера бази даних. За умовчанням локаль — локаль операційної системи. Виберіть Далі.

Відкриється вікно Pre Installation Summary. Воно відображає установки, які вказано за допомогою майстра установки.



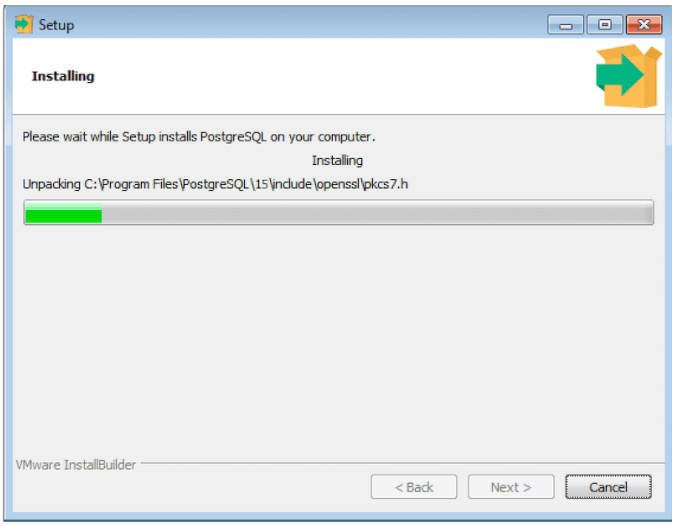
Перегляньте налаштування. Якщо потрібно, натисніть кнопку Назад , щоб повернутися до попереднього діалогового вікна, щоб змінити налаштування. Якщо зведенні налаштування правильні, виберіть Далі .

Майстер повідомить, що він має всю необхідну інформацію для встановлення PostgreSQL.

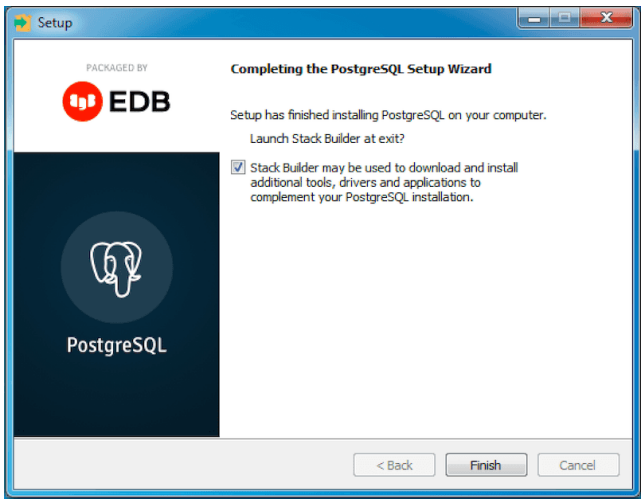


Виберіть Далі.

Під час інсталяції майстер установки підтверджує хід установки PostgreSQL за допомогою низки індикаторів виконання.



Перед тим, як майстер установки завершить установку PostgreSQL, він пропонує запустити Stack Builder при виході. Утиліта Stack Builder надає графічний інтерфейс, який завантажує та встановлює програми та драйвери, що працюють з PostgreSQL.



Залиште прапорець встановленим та виберіть Finish, щоб запустити Stack Builder. Або зніміть прапорець Stack Builder і виберіть Finish, щоб завершити встановлення PostgreSQL без запуску Stack Builder.

Посилання на відеозапис процесу встановлення СУБД PostgreSQL на операційну систему Windows: https://youtu.be/YFk8ESgX0IA.

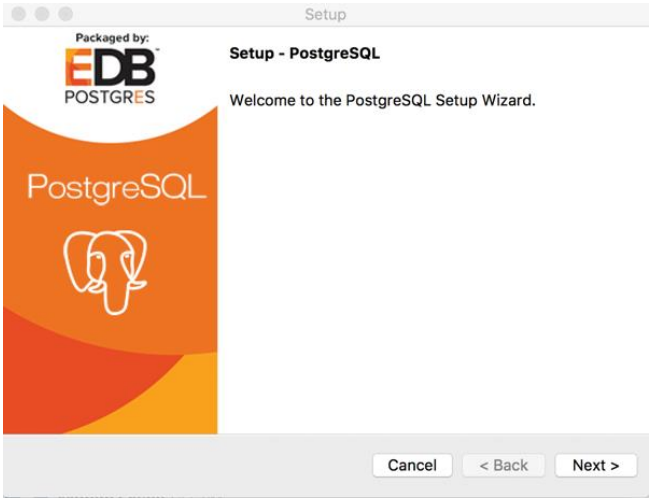
Встановлення СУБД PostgreSQL на операційну систему Mac OS

1. Завантаження установника

Відвідайте сторінку завантажень на сайті EnterpriseDB, https://www.enterprisedb.com/downloads/postgres-postgresql-downloads . Завантажте кращу версію зі списку доступних.

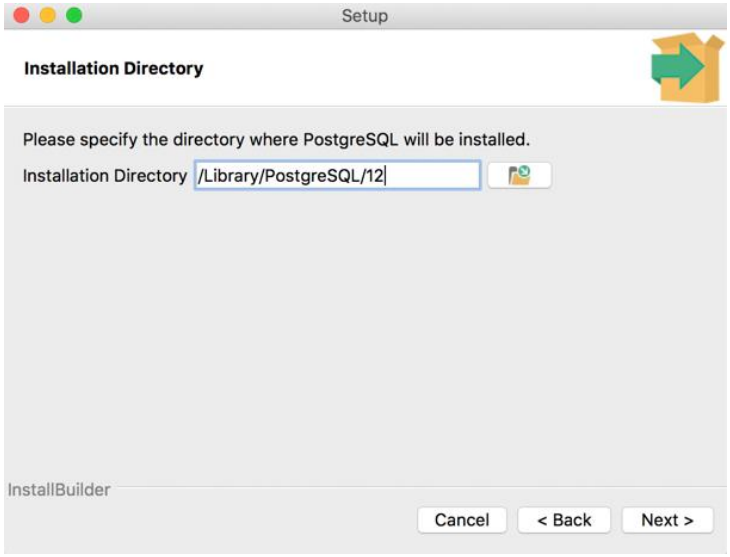
2. Запуск установки

Запустіть завантажений пакет dmg як адміністратор. Коли побачите екран нижче, натисніть кнопку «Далі»:



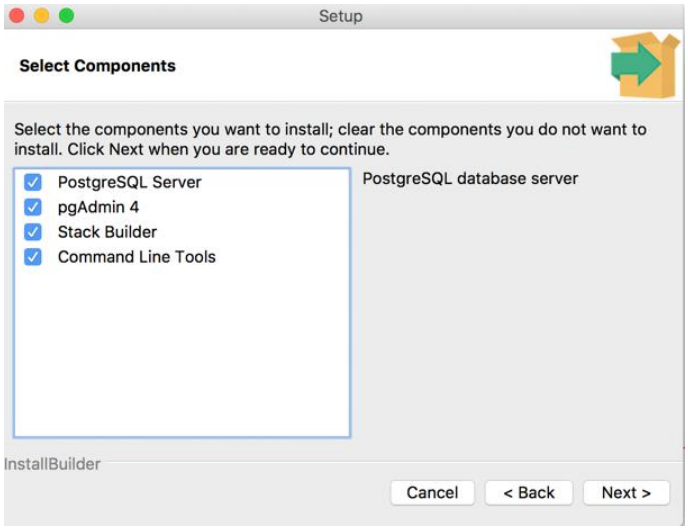
3. Вибір місця встановлення

Вам буде запропоновано вказати, який каталог буде використовуватися для встановлення Postgres. Виберіть бажане місце та натисніть «Далі»:



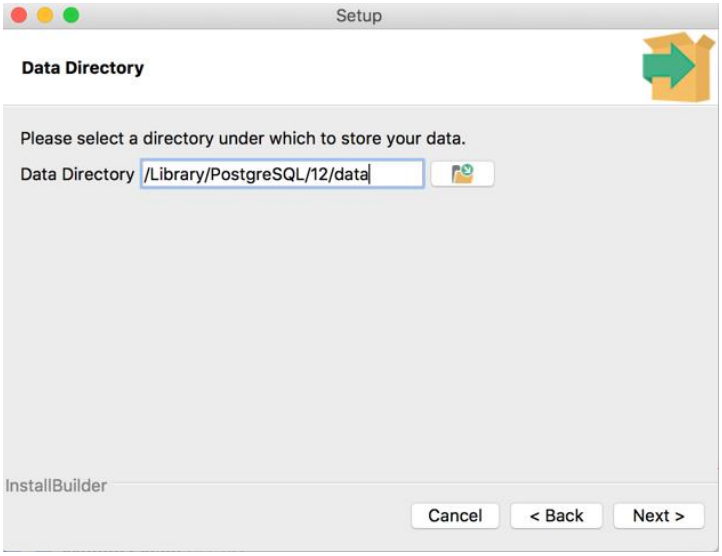
4. Вибір компонентів

Далі буде запропоновано вибрати інструменти, які необхідно встановити разом із установкою Postgres. Сервер PostgreSQL та інструменти командного рядка є обов'язковими. Stack Builder та pgAdmin 4 є необов'язковими. Виберіть зі списку та натисніть «Далі»:



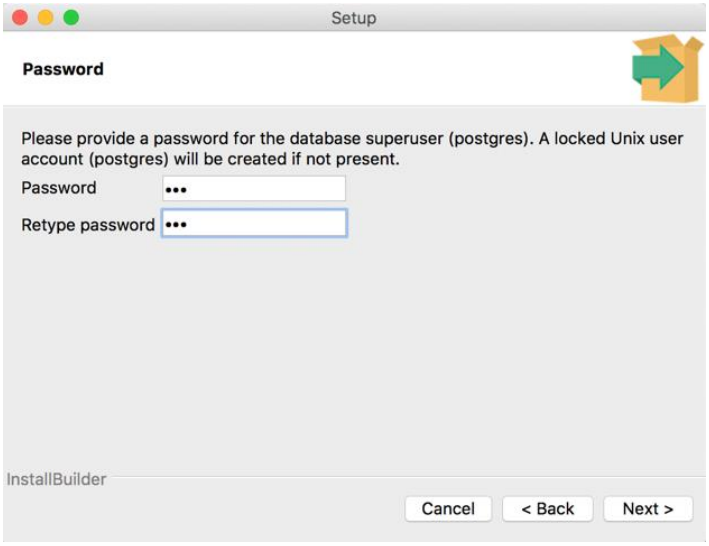
5. Вибір місця зберігання даних (розташування каталогу даних)

Буде запропоновано вибрати місцезнаходження для кластера Postgres Data Directory. Виберіть відповідне місце та натисніть «Далі»:



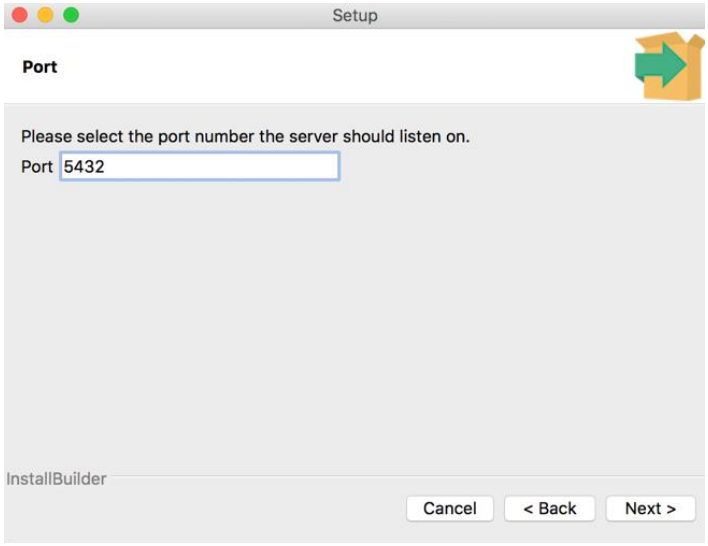
6. Встановлення пароля суперкористувача

Буде запропоновано вказати пароль суперкористувача Postgres Unix, який буде створено під час встановлення. Введіть відповідний пароль та натисніть «Далі»:



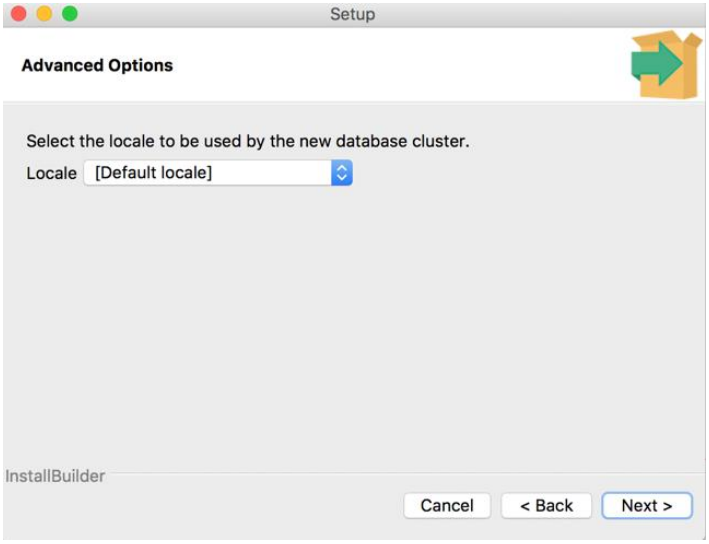
7. Вибір номера порту

Буде запропоновано вибрати номер порту, на якому сервер PostgreSQL прослуховуватиме вхідні з'єднання. Вкажіть відповідний номер порту. (Номер порту за замовчуванням – 5432.) Переконайтеся, що порт відкритий у вашому брандмауері та трафік на цьому порту доступний. Натисніть "Далі":



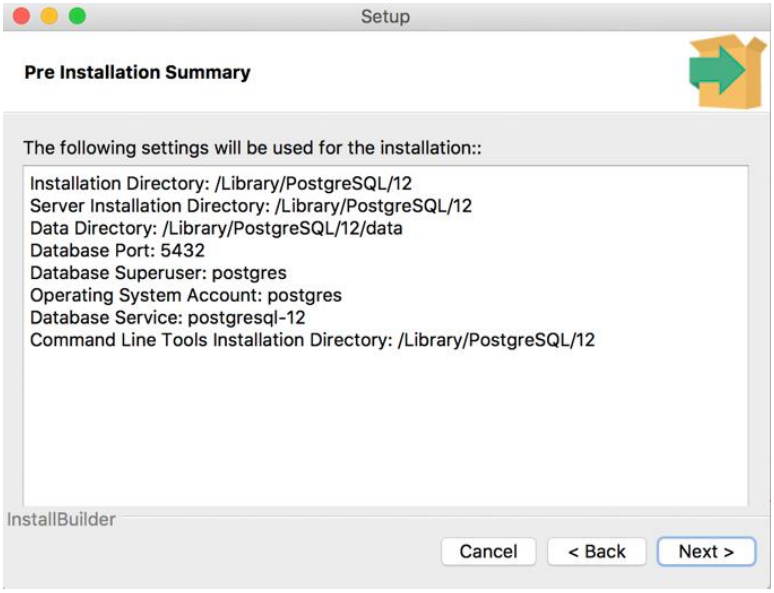
8. Встановлення локалі

Виберіть відповідну локаль (мовні налаштування) та натисніть «Далі»:



9. Перевірка та встановлення

Буде надано зведення виборів із попередніх екранів установки. Уважно перегляньте її та натисніть «Далі», щоб завершити встановлення:



10. Перевірка процесу

Установка успішно завершена, можете переконатися в цьому, підключившись до терміналу та виконавши наступну команду:

```
ps -ef | grep postgres
```

Це дозволить перевірити процеси, запущені для Postgres після встановлення.

Виправлення неполадок під час встановлення

Багатобайтові символи в імені користувача. Якщо ім'я користувача або ім'я машини у системі містить символи, що не входять до ASCII, установка може завершитися невдачею. Переконайтеся, що ім'я користувача або ім'я не містять багатобайтових символів.

Файли журналу установки. Якщо у вас виникли проблеми під час встановлення, зверніться до файлу журналу установки, install-postgresql.log, створеного в:

/tmpна Mac OS X

%TEMP%на Windows

Робота з СУБД PostgreSQL

Інтерфейс PgAdmin

PgAdmin — це графічний інтерфейс для адміністрування та управління базами даних PostgreSQL. Це зручний інструмент, який дозволяє взаємодіяти з базою даних без необхідності використання командного рядка, що робить його популярним серед розробників та адміністраторів баз даних.

Основні можливості PgAdmin:

- Створення і управління базами даних.
- Візуалізація структур бази даних (таблиці, схеми, індекси).
- Виконання SQL-запитів через вбудований редактор.
- Моніторинг активності серверів.
- Резервне копіювання та відновлення даних.

Створення бази даних за допомогою PgAdmin

1. Запуск PgAdmin

Після встановлення та запуску PgAdmin потрібно підключитися до серверу PostgreSQL. Зазвичай сервер встановлюється під час інсталяції СУБД. Якщо це перший запуск PgAdmin і сервер відсутній:

Створіть новий сервер, вибравши опцію **"Add New Server"**.

У вікні, що з'явиться, введіть ім'я серверу та необхідні дані для підключення (ім'я користувача та пароль).

2. Вибір сервера та створення бази даних

Після успішного підключення до сервера у лівій частині інтерфейсу PgAdmin побачите дерево серверів. Розгорніть сервер, до якого підключилися.

Натисніть правою кнопкою миші на розділ **Databases** і виберіть **"Create" → "Database"**.

3. Налаштування бази даних

У вікні **"Create Database"**, що відкриється, введіть ім'я вашої нової бази даних.

Виберіть власника бази даних (за замовчуванням це поточний користувач).

Інші параметри можна залишити за замовчуванням або налаштувати за потреби.

4. Завершення створення бази даних

Після введення імені та налаштування натисніть **"Save"**. Нова база даних буде створена і з'явиться у списку баз даних у вашому сервері.

Термінал psql

Psql – це інтерактивний термінал для роботи з PostgreS. Існує множина прапорів, доступних для використання під час роботи з psql, зосередимося на деяких найважливіших:

-h хост для підключення

-U користувач, до якого потрібно підключитися

-p порт для підключення (за замовчуванням 5432)

```
psql -h localhost -U username databasename
```

Інший варіант – використовувати повний рядок і дозволити psql проаналізувати його:

```
psql "dbname=dbhere host=hosthere user=userhere password=pwhere port=5432 sslmode=require"
```

Після підключення можна почати негайно запитувати. Крім базових запитів, можна використовувати певні команди. Запуск \? дасть список усіх доступних команд, кілька ключових команд наведено нижче.

Часто використовувані команди

За промовчанням час отримання результатів запиту недоступний, але можна увімкнути його за допомогою наступної команди:

```
# \timing
 Timing is on
```

Це покаже час виконання запиту у мілісекундах.

Список таблиць у базі даних

```
# \d
      List of relations
 Schema |   Name    | Type  | Owner
--------+-----------+-------+-------
 public | employees | table | craig
 (1 row)
```

Опишіть таблицю

```
# \d employees
           Table "public.employees"
  Column   |          Type          | Modifiers
-----------+------------------------+-----------
 id        | integer                |
 last_name | character varying(50)  |
 salary    | integer                |
Indexes:
    "idx_emps" btree (salary)
```

Список усіх таблиць у базі даних разом із деякою додатковою інформацією

```
# \d+
       List of relations
Schema | Name  | Type  |   Owner   |  Size  | Description
-------+-------+-------+-----------+--------+-------------
public | users | table | jarvis    | 401 MB |
 (1 row)
```

Опишіть таблицю з додатковою інформацією

```
#\d+ users
          Table "public.users"
  Column   |  Type   |   Modifiers   | Storage  | Stats target | Description
-----------+---------+---------------+----------+--------------+-------------
userid     | bigint  | not null      | plain    |              |
fullname   | text    | not null      | extended |              |
email      | text    | not null      | extended |              |
phone      | text    | not null      | extended |              |
credits    | money   | default 0.0   | plain    |              |
parked     | boolean | default false | plain    |              |
terminated | boolean | default false | plain    |              |

Indexes:
       "users_pkey" PRIMARY KEY, btree (userid)
```

Список усіх баз даних

```
# \l
           List of databases
Name     |   Owner   | Encoding | Collate | Ctype |      Access privileges
---------+-----------+----------+---------+-------+--------------------
 learning | jarvis    | UTF8     | C       | UTF-8 |
```

Список усіх баз даних із додатковою інформацією

```
# \l+
           List of databases
  Name    |   Owner   | Encoding | Collate | Ctype |      Access privileges
|  Size   | Tablespace |                Description
---------+-----------+----------+---------+-------+--------------------
--------+---------+------------+--------------------------------------------
 learning  | jarvis    | UTF8     | C       | UTF-8 |                        |
492 MB  | pg_default |
```

Список усіх схем

```
# \dn
List of schemas
 Name  | Owner
-------+--------
public | jarvis
 (1 row)
```

Перелічіть усі схеми з додатковою інформацією

```
# \dn+
List of schemas
Name   | Owner  | Access privileges |      Description
-------+--------+-------------------+------------------------
public | jarvis | jarvis=UC/jarvis +| standard public schema
       |        | =UC/jarvis        |
 (1 row)
```

Список усіх функцій

```
#\df
```

Перерахуйте всі функції з додатковою інформацією

`#\df+`

Підключитись до іншої бази даних

`#\c dbname`

Вийти з оболонки postgres

`#\q`

Текстовий редактор усередині psql

`#\e`

Це відкриває текстовий редактор за умовчанням усередині оболонки psql. Дуже зручно для модифікації запитів.

Звичайно, є багато інших команд, як було сказано вище.

Текстові редактори

При роботі з терміналом psql зручно використовувати будь-який текстовий редактор з підсвічуванням команд SQL. Зручно писати/правити команди та запити в текстовому редакторі, а потім копіювати в термінал.

Існує багато текстових редакторів і середовищ розробки, які підтримують підсвічування синтаксису SQL. Деякі з них є простими редакторами, подібними до Notepad++, тоді як інші є більш потужними IDE (інтегровані середовища розробки), призначені для роботи з базами даних. Ось кілька популярних варіантів:

1. Notepad++

Особливості: Легкий текстовий редактор з відкритим вихідним кодом. Підтримує підсвічування синтаксису для багатьох мов програмування, включаючи SQL. Можливість налаштування тем та плагінів.

Переваги: Легкий і швидкий. Широко підтримується спільнотою.

Недоліки: Обмежені можливості для роботи з базами даних (без вбудованих функцій виконання SQL-запитів).

Sublime Text

Особливості: Потужний редактор з підтримкою великої кількості мов програмування. - Підтримує підсвічування синтаксису SQL. Можливість встановлення плагінів для розширення функціоналу.

Переваги: Висока швидкість роботи. Інтуїтивно зрозумілий інтерфейс.

Недоліки: Платний (є безкоштовна пробна версія).

Visual Studio Code (VS Code)

Особливості: Безкоштовний редактор з відкритим вихідним кодом від Microsoft. Підтримує підсвічування синтаксису SQL та інших мов через розширення. Має розширення для роботи з базами даних, такі як `SQL Server`, `PostgreSQL`, `MySQL`.

Переваги: Можливість виконання SQL-запитів прямо в редакторі. Велика кількість доступних розширень та активна спільнота.

Недоліки: Може споживати більше ресурсів системи, ніж легші редактори.

Якщо потрібен легкий і швидкий редактор для швидкої роботи з SQL, Notepad++, Sublime Text або Visual Studio Code будуть хорошим вибором. Для більш глибокої роботи з базами даних, включаючи виконання SQL-запитів та управління базами, рекомендуються такі інструменти, як DBeaver або SQL Developer.

Структура звіту до лабораторної роботи

1. Завдання.
2. Опис предметної області згідно пояснень.
3. Розроблені User Story згідно пояснень. Мінімум 3 користувача та по 3 історії від кожного користувача (тобто мінімум 9 історій користувачів).

4. Скриншот створеної бази даних з відповідним ім’ям в PgAdmin.
5. Скриншот створеної бази даних з відповідним ім’ям в psql.

Звіт до 1 лабораторної роботи можна здати онлайн на сайті ДО edu.op.edu.ua до початку вашого заняття.

Перелік джерел

1. Що таке User Story і як її писати. URL: https://training.qatestlab.com/blog/technical-articles/user-story/ (дата звернення: 30.08.2022).
2. Andrii Pastushok. User Story та Acceptance Criteria: пишемо чіткі та зрозумілі вимоги | DOU. URL: https://dou.ua/lenta/articles/clear-user-stories/ (дата звернення: 30.08.2022).
3. User story - приклад, особливості, відгуки та застосування - Програмне забезпечення 2022. URL: https://ukr.kagutech.com/4087304-user-story-example-features-reviews-and-applications (дата звернення: 30.08.2022).
4. Гайд по User Stories для Junior BA - Про Бізнес-Аналіз Українською. URL: https://www.ba.in.ua/2021/09/21/gajd-po-user-stories-dlya-junior-ba/ (дата звернення: 30.08.2022).
5. User Stories and User Story Examples by Mike Cohn. URL: https://www.mountaingoatsoftware.com/agile/user-stories (дата звернення: 30.08.2022).
6. How to Write a Good User Story: with Examples & Templates. URL: https://stormotion.io/blog/how-to-write-a-good-user-story-with-examples-templates/ (дата звернення: 30.08.2022).
7. How (and Why) to Write Great User Stories | The Innovation Mode. URL: https://www.theinnovationmode.com/the-innovation-blog/user-stories-in-agile-the-whys-and-hows (дата звернення: 30.08.2022).
8. How to Write User Stories (in Agile): The Three C’s and Examples | Chisel. URL: https://chisellabs.com/blog/how-to-write-agile-user-stories-three-cs-examples/ (дата звернення: 30.08.2022).
9. 10 Tips for Writing Good User Stories. URL: https://www.romanpichler.com/blog/10-tips-writing-good-user-stories/ (дата звернення: 30.08.2022).