Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ДОПУСТИТИ ДО ЗАХИСТУ:
В.о. завідувача кафедри
кібербезпеки та захисту
інформації
_____ Іван ПАРХОМЕНКО
«13» червня 2025 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

кваліфікаційної роботи

галузь знань 12 Інформаційні технології
(шифр і назва галузі знань)

спеціальність 125 Кібербезпека
(код і назва спеціальності)

освітній ступень бакалавр

освітня програма Кібербезпека
(назва освітньо-професійної програми)

на тему: «Методика захисту конфіденційної інформації в базах даних»

Виконавець: студент IV курсу, групи КБ-41

_____ **Максим ВЕЗДЕЦЬКИЙ**
(підпис) (ім'я, прізвище)

	Підпис	**Ім'я ПРІЗВИЩЕ**
Керівник		Інна МИХАЛЬЧУК
Нормоконтроль		Юрій ЩЕБЛАНІН

Київ 2025

Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ЗАТВЕРДЖЕНО:
В.о. завідувача кафедри
кібербезпеки
та захисту інформації
_____ Іван ПАРХОМЕНКО
«29» листопада 2024 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи

спеціальності 125 Кібербезпека
(код і назва спеціальності)

освітньої програми Кібербезпека
(назва освітньо-професійної програми)

Студенту **КБ-41** (група) **Вездецькому Максиму Павловичу** (прізвище ім'я по батькові)

Тема кваліфікаційної роботи ***Методика захисту конфіденційної інформації в базах даних***

1. ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Тема кваліфікаційної роботи затверджена на засіданні кафедри кібербезпеки та захисту інформації протокол №6 від 28.11.2024 р.

2. ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Актуальні загрози та вразливості баз даних, сучасні засоби захисту.

3. ЗМІСТ РОЗРАХУНКОВО-ПОЯСНЮВАЛЬНОЇ ЗАПИСКИ

Аналіз типових загроз та вразливостей сучасних систем управління базами даних. Ознайомлення із сучасними засобами та методами забезпечення конфіденційності інформації в базах даних. Розробка методики захисту конфіденційної інформації в базах даних.

4. ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Практична цінність у впровадженні цілісної методики захисту, яка може бути адаптована в інформаційних системах різного масштабу та типу.

5. ДАТА ВИДАЧІ ЗАВДАННЯ

Дата видачі завдання: 29 листопада 2024 року

Завдання видала _____ (підпис) Інна МИХАЛЬЧУК (ім'я, прізвище)

Завдання прийняв до виконання _____ (підпис) Максим ВЕЗДЕЦЬКИЙ (ім'я, прізвище)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів робіт	Строки виконання робіт	Відмітка про виконання
1	Уточнення постановки задачі	29.11.2024 – 22.01.2025	*виконано*
2	Аналіз літератури	23.01.2025 – 11.02.2025	*виконано*
3	Обґрунтування вибору рішення	12.02.2025 – 17.02.2025	*виконано*
4	Аналіз загроз і вразливостей інформаційної безпеки в БД	18.02.2025 – 04.03.2025	*виконано*
5	Дослідження методів, засобів та принципів захисту	05.03.2025 – 24.03.2025	*виконано*
6	Вибір цільової СУБД, обґрунтування платформи	25.03.2025 – 08.04.2025	*виконано*
7	Проєктування та впровадження методики захисту	09.04.2025 – 09.05.2025	*виконано*
8	Оформлення пояснювальної записки	12.05.2025 – 27.05.2025	*виконано*
9	Підготовка до захисту кваліфікаційної роботи	28.05.2025 – 13.06.2025	*виконано*

Завдання видав _____ (підпис) Інна МИХАЛЬЧУК (ім'я, прізвище)

Завдання прийняв до виконання _____ (підпис) Максим ВЕЗДЕЦЬКИЙ (ім'я, прізвище)

Термін подання кваліфікаційної роботи до ЕК 13 червня 2025 року

РЕФЕРАТ

Дипломна робота складається зі вступу, трьох розділів, загальних висновків, списку використаних джерел, додатків, має 83 сторінки основного тексту та 15 рисунків. Список використаних джерел містить 30 найменувань і займає 4 сторінки.

Метою даної роботи є розробка та впровадження методики захисту конфіденційної інформації в системах управління базами даних, що забезпечує високий рівень безпеки при збереженні ефективності доступу та обробки даних.

Для досягнення зазначеної мети поставлено наступні задачі дослідження:

- Проаналізувати сучасні загрози безпеці баз даних, що містять конфіденційну інформацію.
- Вивчити існуючі методи захисту даних у СКБД, зокрема засоби шифрування, контролю доступу та моніторингу.
- Дослідити криптографічні підходи до забезпечення конфіденційності даних (шифрування, хешування тощо).
- Розглянути механізми автентифікації, авторизації та розмежування прав доступу в СКБД.
- Розробити методику моніторингу та журналювання дій користувачів для виявлення підозрілої активності.
- Сформувати узагальнену методику захисту конфіденційної інформації в базах даних на основі поєднання зазначених підходів.
- Реалізувати та протестувати модель захищеної бази даних із впровадженням розробленої методики.
- Оцінити ефективність методики за критеріями конфіденційності, цілісності, доступності та зручності адміністрування.

Об’єктом дослідження є системи керування базами даних (СКБД), що містять конфіденційну інформацію.

Предметом дослідження є методи та засоби захисту конфіденційної інформації в системах управління базами даних.

Практична цінність роботи полягає в розробці методики захисту конфіденційної інформації в базах даних, яка поєднує криптографію, контроль доступу та моніторинг користувачів. Запропоноване рішення підвищує безпеку зберігання чутливих даних і знижує ризики несанкціонованого доступу чи витоку інформації. Методика може бути впроваджена в державних та комерційних інформаційних системах для покращення їх захисту.

Ключові слова: база даних, конфіденційна інформація, PostgreSQL, авторизація, шифрування, SQL-ін’єкції, аудит, інформаційна безпека.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

БД	–	Бази даних
НСД	–	Несанкціонований доступ
ПЗ	–	Програмне забезпечення
СУБД	–	Система управління базами даних
API	–	Application Programming Interface
CIA	–	Confidentiality Integrity Availability
CVE	–	Common Vulnerabilities and Exposures
CVSS	–	Common Vulnerability Scoring System
DAM	–	Database Activity Monitoring
DBIR	–	Data Breach Investigations Report
DDoS	–	Distributed Denial of Service
DLP	–	Data Loss Prevention
DoS	–	Denial of Service
EDR	–	Endpoint Detection and Response
IDS	–	Intrusion Detection System
IPS	–	Intrusion Prevention System
LDAP	–	Lightweight Directory Access Protocol
ORM	–	Object Relational Mapping
PoLP	–	Principle of Least Privilege
SIEM	–	Security Information and Event Management
SQL	–	Structured Query Language

TDE – Transparent Data Encryption

TLS – Transport Layer Security

VPN – Virtual Private Network

ЗМІСТ

ВСТУП

У сучасному цифровому середовищі обсяги конфіденційної інформації, що зберігається у базах даних, невпинно зростають, що зумовлює нагальну потребу в удосконаленні засобів її захисту. Бази даних стали ключовими компонентами критичної інформаційної інфраструктури в державному, корпоративному та приватному секторах. З огляду на це, питання захисту даних, які мають обмежений доступ або становлять особливу цінність, набуває стратегічного значення як у технічному, так і в правовому контекстах.

Попри наявність великої кількості теоретичних підходів і практичних реалізацій, проблематика забезпечення інформаційної безпеки у СУБД залишається актуальною через постійне вдосконалення методів атак, зростання кількості внутрішніх інцидентів, розвиток хмарних рішень і поява нових регуляторних вимог. Більше того, велика частина інцидентів безпеки, як показує практика, трапляється через недосконаліть конфігурацій, нехтування основними принципами захисту або брак ефективного моніторингу.

Обрана тема кваліфікаційної роботи спрямована на комплексне вирішення цих проблем шляхом розробки методики, яка об’єднує сучасні засоби шифрування, контроль доступу, аудит, моніторинг та управління вразливостями у СУБД. В основі дослідження лежить застосування практично орієнтованого підходу із впровадженням рішень на прикладі платформи PostgreSQL.

Метою роботи є створення ефективної, багатоетапної методики захисту конфіденційної інформації, яка враховує поточні загрози, дозволяє адаптацію до різних типів систем та відповідає принципам побудови безпечних IT-архітектур. Досягнення цієї мети потребує глибокого аналізу типових загроз, оцінки вразливостей, систематизації сучасних методів захисту, а також впровадження реальних технічних засобів, здатних забезпечити конфіденційність, цілісність і доступність інформації у динамічному цифровому середовищі.

РОЗДІЛ 1
СУЧАСНИЙ СТАН ІНФОРМАЦІЙНОЇ БЕЗПЕКИ В БАЗАХ ДАНИХ

1.1 Основні загрози інформаційній безпеці в базах даних

Інформаційна безпека баз даних (БД) є критично важливою складовою загальної безпеки будь-якої інформаційної системи, оскільки саме БД зазвичай містять найбільш цінні активи – персональні дані користувачів, фінансову інформацію, комерційні таємниці, конфіденційні документи. *Загроза інформаційній безпеці* – це будь-який потенційний фактор, який може призвести до порушення конфіденційності, цілісності або доступності даних. В умовах зростання кіберзлочинності та ускладнення методів атак аналіз таких загроз стає необхідним етапом для побудови ефективної системи захисту.

1.1.1 Внутрішні та зовнішні загрози

Загрози, які виникають щодо БД, мають різноманітний характер і класифікуються за кількома критеріями. Найпоширенішим підходом є поділ загроз на внутрішні та зовнішні, оскільки ці категорії визначають джерело, спосіб реалізації та складність виявлення атак.

Внутрішні загрози виникають у межах самої організації та зазвичай пов’язані з діями співробітників або підрядників, які мають певний рівень авторизованого доступу до БД. Вони можуть бути як навмисними, коли особа свідомо здійснює дії, що шкодять безпеці, так і ненавмисними – коли помилки виникають внаслідок некомпетентності, недбалості або відсутності належної підготовки.

Типовими джерелами внутрішніх загроз є: системні адміністратори, розробники прикладного ПЗ, працівники служб підтримки, звичайні користувачі, тимчасові підрядники.

Навмисні дії можуть включати копіювання або експорт конфіденційної інформації, передачу облікових даних третім особам, зміну прав доступу без належного дозволу, модифікацію структури БД з прихованими цілями, або саботаж роботи системи шляхом умисного її пошкодження.

Ненавмисні дії зазвичай проявляються у вигляді неправильного введення даних, випадкового видалення таблиць або записів, конфігураційних помилок, які відкривають доступ до даних, або неправильного налаштування резервного копіювання, що унеможливлює відновлення після інциденту.

Окрему категорію становлять соціотехнічні інциденти, коли внутрішні користувачі, самі того не усвідомлюючи, стають посередниками у реалізації атак, наприклад, коли відкривають фішингові посилання або встановлюють шкідливе ПЗ, яке отримує доступ до БД через їхні облікові записи.

Загрозливість внутрішніх атак полягає у:

- високому рівні довіри до внутрішніх користувачів з боку систем безпеки;
- наявності прав доступу до критичних об'єктів;
- складності виявлення – дії співробітника можуть виглядати легітимними з точки зору системних логів;
- довготривалості впливу – внутрішній зловмисник може непомітно діяти місяцями.

На рис. 1.1 зі звіту M-Trends 2025 можемо спостерігати тенденцію глобального середнього часу для виявлення зовнішніх та внутрішніх вторгнень у систему з 2011 до 2024 рр. [9].

	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021	2022	2023	2024
All	416	243	229	205	146	99	101	78	56	24	21	16	10	11
External	—	—	—	—	320	107	186	184	141	73	28	19	13	11
Internal	—	—	—	—	56	80	57.5	50.5	30	12	18	13	9	10

Рисунок 1.1 – Середній час виявлення вторгнень у систему (вимірювання в днях)

Зовнішні загрози – це дії, що здійснюються особами або автоматизованими системами поза межами організації, які не мають авторизованого доступу до БД і намагаються отримати його шляхом використання вразливостей, помилок або недосконалих налаштувань. У контексті сучасної кібербезпеки зовнішні загрози є більш динамічними та технологічно розвинутими, оскільки зловмисники постійно вдосконалюють свої методи атаки.

До основних типів зовнішніх атак на БД належать:

- автоматизоване сканування уразливостей, що виконується ботнетами з метою виявлення відкритих інтерфейсів БД;
- експлуатація SQL-ін’єкцій, яка дозволяє обійти автентифікацію або отримати повний доступ до вмісту БД;
- атаки типу «людина посередині» у разі нешифрованих з'єднань між сервером БД і клієнтами;
- віддалене встановлення бекдорів та запуск шкідливого ПЗ на сервері, що дозволяє викрадати дані або порушувати роботу БД;
- атаки через вразливості сторонніх компонентів, зокрема вебсерверів, API, хмарних платформ.

Зовнішні зловмисники зазвичай мають чітко визначену мотивацію:

- фінансова вигода (викрадення персональних даних для продажу, вимагання викупу);

- шпигунство (збір конфіденційної інформації для конкурентів або держав);
- активізм або деструктивні наміри (публічне викриття організацій, компрометація даних, порушення діяльності установи).

Зовнішні загрози зазвичай реалізуються швидко і масово, часто — із використанням анонімних серверів, VPN або зламаних вузлів, що ускладнює ідентифікацію зловмисника.

Слід також враховувати, що зовнішні атаки можуть бути комбінованими, коли спочатку виконується атака на одного з працівників (наприклад, фішинг або інженерний вплив), а потім — проникнення в систему через скомпрометований обліковий запис. Такі комбіновані сценарії підвищують ефективність атак і є особливо небезпечними.

1.1.2 Типові загрози

Інформаційна безпека БД перебуває під постійним тиском нових загроз, які еволюціонують одночасно з розвитком інформаційних технологій. Незалежно від джерела внутрішнього чи зовнішнього низка загроз виявляється найбільш поширеною та критичною з погляду шкоди, яку вони можуть завдати.

Дивлячись на щорічний аналітичний звіт про розслідування витоків даних від компанії Verizon (DBIR 2025), можемо розглянути наступну тенденцію кіберзагроз в 2025 р. в порівнянні з 2024 р. на рис. 1.2 [1]. На діаграмі представлено процентне співвідношення основних типів кіберінцидентів:

1. Проникнення в систему (System Intrusion) – складні атаки, що включають використання шкідливого ПЗ, експлуатацію вразливостей систем та інші методи компрометації інфраструктури.
2. Різні помилки (Miscellaneous Errors) – інциденти, спричинені людським фактором або неправильними налаштуваннями систем (наприклад, публічно доступні БД через помилки конфігурації).

3. Соціальна інженерія (Social Engineering) – атаки, що ґрунтуються на маніпуляціях користувачами (фішинг, претекстинг тощо).

4. Базові атаки на веб-додатки (Basic Web Application Attacks) – стандартні веб-загрози, такі як SQL-ін'єкції, міжсайтовий скриптинг (XSS) та інші вразливості веб-додатків.

5. Зловживання привілеями (Privilege Misuse) – випадки зловживання наданими правами доступу, часто пов’язані з внутрішніми загрозами.

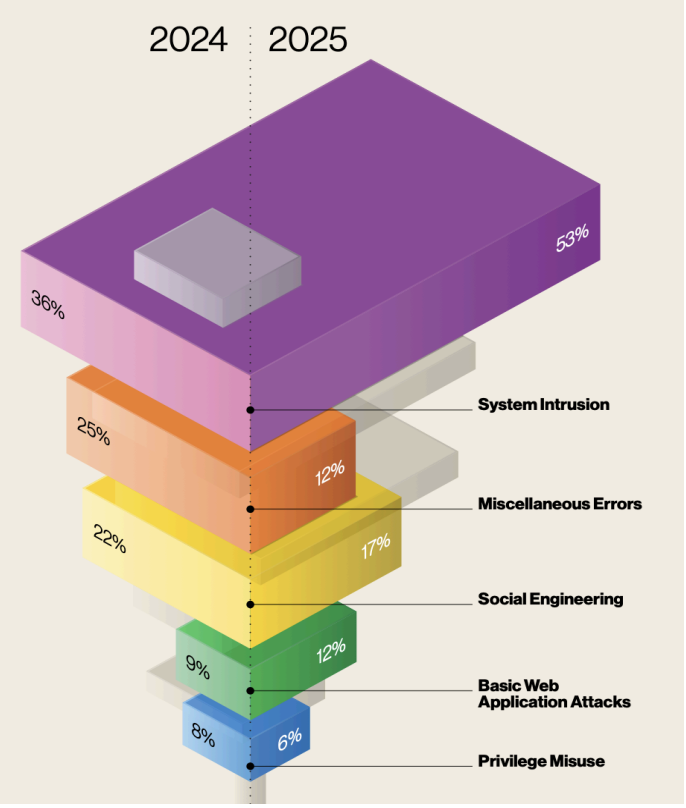

Рисунок 1.2 – Розподіл кіберінцидентів за типами загроз у 2024 – 2025 рр..

Несанкціонований доступ (НСД) до бази даних є однією з найбільш поширених та небезпечних загроз. Йдеться про ситуації, коли зловмисник отримує можливість читати, змінювати або знищувати дані, обійшовши або зламавши механізми автентифікації та авторизації. На практиці це може бути реалізовано через викрадення облікових даних, підбір паролів, використання конфігураційних помилок або брак багатофакторної автентифікації.

Згідно з графіком на рис. 1.3, понад 22% інцидентів у сфері витоку даних були пов’язані саме з використанням вкрадених облікових записів. При цьому в значній кількості випадків зловмисники отримували доступ до систем, зокрема БД, завдяки відкритим сховищам із конфігураційними файлами або через фішингові атаки(15%) на співробітників. Особливо критичним є те, що в 46% таких інцидентів використовувалися пристрої, не підпорядковані жодній централізованій IT-політиці. Це означає, що корпоративні БД були доступні з приватних пристроїв без належного контролю [1].

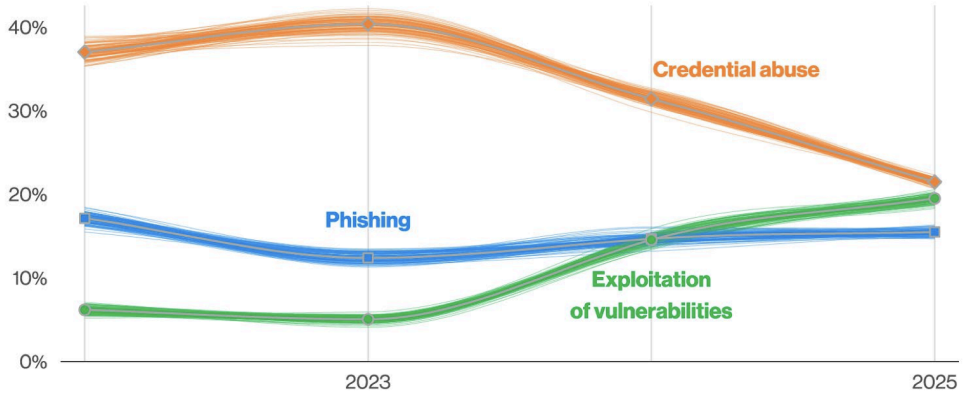

Рисунок 1.3 – Вектори початкового доступу під час порушень інформаційної безпеки

НСД часто є першим етапом складнішої атаки, яка надалі може включати встановлення бекдорів, модифікацію структур таблиць або розгортання шкідливого ПЗ в інфраструктурі організації.

SQL-ін’єкції (SQL Injection) залишаються одними з найнебезпечніших методів атак на БД, особливо у випадках, коли СУБД взаємодіє з вебдодатками. Суть такої атаки полягає у впровадженні зловмисного SQL-коду у введені користувачем дані (наприклад, у форму авторизації), які потім без перевірки передаються до БД. У разі відсутності коректної фільтрації введених значень або використання непараметризованих запитів система може виконати

шкідливу команду, що дозволяє зловмиснику обійти автентифікацію або отримати повний доступ до інформації.

За даними на рис. 1.2, SQL-ін’єкції належать до групи «Basic Web Application Attacks» і становлять приблизно 12% усіх підтверджених інцидентів. Із щорічного звіту компанії Verizon відомо, що значну частину атак вдалося запобігти завдяки впровадженню WAF (Web Application Firewall) та захисту на рівні ORM, однак багато організацій все ще не фільтрують введення з користувацьких форм [1]. Попри тривалу історію цього типу атак, велика кількість сучасних застосунків залишається вразливою, особливо ті, що побудовані на застарілих фреймворках або створені без урахування принципів безпечного програмування. Поширення відкритих інтерфейсів API та мікросервісів без достатнього рівня захисту також сприяє зростанню кількості SQL-ін’єкцій.

Витік інформації – це ще одна критично важлива загроза, яка призводить до порушення конфіденційності даних. Така ситуація виникає, коли конфіденційна інформація організації стає доступною стороннім особам без відповідного дозволу. Причинами можуть бути як технічні вразливості, так і організаційні прорахунки: неналежна конфігурація доступу, зберігання резервних копій без шифрування, недостатній захист журналів або логів, а також людський фактор — випадкове поширення файлів із чутливою інформацією.

Згідно зі звітом DBIR за 2025 рік, у 54% підтверджених інцидентів організації постраждали внаслідок витоку облікових даних у відкритих джерелах. У багатьох випадках ця інформація надалі використовувалася у вторинних атаках: для входу в системи, шифрування даних або прихованої ескалації привілеїв. Середній час між витоком облікових даних і реакцією компанії становив 94 дні, що свідчить про серйозну проблему в ранньому виявленні таких загроз [1].

Шкідливе програмне забезпечення (ПЗ) є ефективним інструментом для викрадення облікових даних, конфігурацій файлів або журналів підключення до бази даних. Таке ПЗ може потрапити на комп'ютери користувачів через фішинг, заражені вебсайти або шкідливі вкладення, після чого воно приховано працює у фоновому режимі, збираючи інформацію про з'єднання, збережені паролі та інші чутливі об'єкти.

За даними DBIR у 2025 році відзначається, що близько 30% заражених шпигунськими програмами пристроїв мали прямий або опосередкований доступ до корпоративної інфраструктури, зокрема баз даних [1]. Отримана таким чином інформація використовувалася зловмисниками для обходу автентифікації, створення нових облікових записів, копіювання бази даних або для розгортання шкідливих скриптів у межах СУБД.

Ситуація ускладнюється ще й тим, що шпигунське ПЗ часто не фіксуються антивірусним ПЗ, особливо якщо використовуються кастомізовані версії або варіанти, створені для цільових атак.

Атаки на відмову в обслуговуванні (DoS/DDoS) спрямовані не на викрадення даних, а на виведення з ладу систем, зокрема БД. Їхня суть полягає у перевантаженні ресурсу численними або складними запитами, внаслідок чого сервер перестає обробляти легітимні запити або взагалі аварійно завершує роботу.У контексті БД такі атаки можуть здійснюватися шляхом надсилання об'ємних або неефективних SQL-запитів, що заблокують великі таблиці або викличуть колізії у механізмах реплікації. Також можливе сповільнення чи блокування резервного копіювання або внутрішніх служб моніторингу.

DoS-атаки дедалі частіше використовуються не як самостійна мета, а як засіб відволікання уваги адміністраторів під час проведення більш небезпечної атаки, наприклад, впровадження шкідливого ПЗ або крадіжки облікових даних. Це робить їх особливо ефективними в умовах низької оперативної готовності організацій до реагування на інциденти.

1.2 Вразливості сучасних СУБД

Порушення захищеності СУБД становить серйозну загрозу для безперервності бізнес-процесів, відповідності нормативним вимогам, а також репутаційної стійкості організацій. Одним із першочергових завдань у сфері захисту інформації є виявлення, класифікація та розуміння характеру потенційних вразливостей, що притаманні СУБД.

Класифікація вразливостей БД є необхідною умовою для системного аналізу ризиків, проектування засобів захисту та побудови ефективної архітектури інформаційної безпеки. Вразливості можуть бути класифіковані за різними критеріями: джерелом походження, рівнем реалізації в системі, способом експлуатації та наслідками для організації. Кожен із цих підходів дозволяє висвітлити певний аспект проблеми та краще зрозуміти механізми виникнення й використання вразливостей на практиці [10].

Одним із базових критеріїв є походження вразливості, яке дозволяє поділити їх на *архітектурно-програмні*, *конфігураційні* та *інфраструктурні*:

- *Архітектурні* вразливості виникають ще на етапі проєктування або розробки СУБД і можуть бути зумовлені логічними помилками, недостатньою валідацією вхідних даних, некоректною реалізацією механізмів доступу або відсутністю захисту від типових атак, таких як SQL-ін’єкції.
- *Конфігураційні* вразливості, своєю чергою, є наслідком неправильно налаштованих параметрів безпеки: використання облікових записів за замовчуванням, надання надлишкових прав доступу, відсутність шифрування під час передавання даних, відкриті порти або служби.
- *Інфраструктурні* вразливості пов’язані з недоліками у зовнішньому середовищі функціонування СУБД — операційній системі, мережевій інфраструктурі, сервісах резервного копіювання чи моніторингу, а також хмарних середовищах розгортання.

Наступним важливим підходом є класифікація вразливостей за рівнем реалізації в системі, що дозволяє виявити, на якому саме етапі взаємодії з СУБД виникає небезпека. Зокрема, виділяють *вразливості ядра* СУБД, які стосуються механізмів обробки транзакцій, індексації, буферизації та оптимізації запитів. У разі компрометації цих механізмів атакувальник може або вплинути на поведінку всієї бази, або використати вразливість для подальшого проникнення в систему. *Вразливості механізмів автентифікації та авторизації* — ще одна поширена категорія – виникають через слабку політику паролів, відсутність багатофакторної автентифікації, помилки у логіці перевірки доступу або використання застарілих бібліотек. Не менш важливими є *вразливості інтерфейсів взаємодії з базою даних* – API, вебконсолей, адміністраторських панелей, драйверів. Останнім часом особливої уваги потребують *вразливості додаткових розширень та плагінів*, які підключаються до СУБД сторонніми розробниками й нерідко мають нижчий рівень безпеки, ніж ядро системи.

Нарешті, доцільно класифікувати вразливості за рівнем потенційної шкоди, яку вони можуть завдати. *Критичні* вразливості можуть призвести до повного витоку даних, знищення таблиць, компрометації облікових записів адміністратора або використання СУБД як стартової точки для проникнення в інші компоненти інформаційної системи. *Середні* за рівнем ризику вразливості зазвичай не дають прямого доступу до всієї системи, але можуть бути використані як частина складніших атак . Вразливості з *низьким рівнем* впливу часто не становлять серйозної небезпеки без додаткових умов, однак у поєднанні з іншими факторами вони можуть бути використані для реалізації атаки ланцюжком. На рис. 1.4 можна переглянути діаграму, яка пропонує повне уявлення про розподіл вразливостей за рівнем потенційної шкоди за 2023-2024 рр. (діаграма включає вразливості, для яких немає конкретної інформації про серйозність, позначені як “(blank)”.

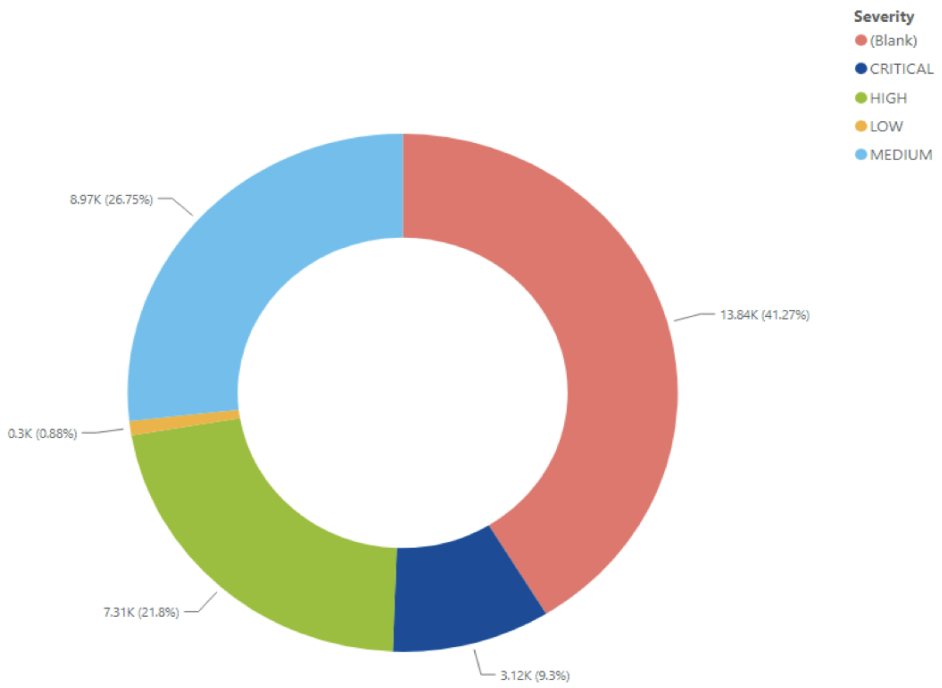

Рисунок 1.4 – Відсоток CVE за рівнем потенційної шкоди

1.2.1 Вразливості програмного забезпечення СУБД

Програмна складова СУБД – її ядро, системні функції, бібліотеки доступу, модулі реплікації та розширення являються центральним об'єктом для атак зловмисників. Помилки в реалізації внутрішніх алгоритмів або недоліки у коді компонентів можуть призводити до виникнення критичних вразливостей, що дозволяють здійснити несанкціоноване виконання коду, підвищити привілеї, обійти автентифікацію або викликати відмову в обслуговуванні.

Згідно з ENISA Threat Landscape 2024, упродовж останніх років спостерігається стабільно високий рівень виявлення критичних вразливостей у популярних СУБД, особливо у проєктах з відкритим кодом. Аналіз CVE-бази свідчить, що більшість таких вразливостей класифікуються за рівнем загрози як "високий" або "критичний" відповідно до CVSS v3, а частина з них отримує експлойти практично одразу після оприлюднення [2].

До найбільш небезпечних вразливостей належать ті, які безпосередньо стосуються механізмів збереження, обробки та індексації даних. Наприклад, у

2024 році було виявлено критичну вразливість у MySQL (CVE-2024-20917), що дозволяла виконати довільний SQL-запит з привілеями адміністратора через некоректну обробку запитів у режимі prepared statements [3].

У PostgreSQL у вересні 2024 року виявлено вразливість (CVE-2024-23480), яка дозволяла користувачеві з низькими привілеями модифікувати внутрішні функції системного каталогу через помилки в логіці перевірки ролей. Це дало змогу атакувальникам ескалувати привілеї до рівня суперкористувача в межах одного вузла кластера [4].

У випадку MongoDB зафіксовано критичні вразливості, що стосуються open-access конфігурацій, які, поєднані з помилками в механізмі перевірки автентифікації в деяких версіях драйверів, дозволяли обійти захист і зчитувати колекції без жодної авторизації.

Недоліки в реалізації механізмів автентифікації регулярно призводять до серйозних загроз. Часто джерелом проблем є те, що багато СУБД підтримують зворотну сумісність зі застарілими методами авторизації. Наприклад, в Oracle Database деякий час залишався активним механізм автентифікації з використанням DES-хешів, які легко піддаються атакам грубої сили (brute-force), особливо якщо пароль не був змінений після інсталяції [5].

У MSSQL були виявлені вразливості, пов’язані з механізмами інтеграції з Active Directory, які дозволяли зловмиснику зі зламаними корпоративними обліковими даними обійти політику дозволів на рівні БД (CVE-2024-36891) [6].

СУБД усе частіше використовуються в модульному або мікросервісному форматі, що передбачає інтеграцію численних сторонніх бібліотек, плагінів і драйверів. Ці компоненти, на відміну від ядра СУБД, рідше проходять повноцінний аудит безпеки.

Наприклад, у 2024 році фреймворк Sequelize для роботи з PostgreSQL та MySQL мав критичну вразливість у механізмі escaping SQL-запитів, що дозволяло здійснювати SQL-ін’єкції через типові ORM-запити в Node.js-проєктах [7].

У MySQL-плагінах для аудиту було виявлено вразливість, яка дозволяла користувачу з привілеями AUDIT_ADMIN переглядати записи з інших сесій, включно з параметрами автентифікації [8].

1.2.2 Конфігураційні вразливості та людський фактор

Поряд із програмними помилками вразливості, що виникають унаслідок неправильного або необачного конфігурування СУБД, залишаються однією з найпоширеніших причин порушення безпеки. До цього додається ще один суттєвий аспект – людський фактор, який охоплює не лише помилки технічного персоналу, а й організаційні прорахунки, нестачу підготовки або недбале ставлення до політик захисту інформації.

Одна з найтиповіших помилок конфігурації – надання загального доступу до СУБД через Інтернет без обмежень за IP-адресами або VPN-захисту. Найбільш часто це стосується MongoDB, Redis, Elasticsearch і PostgreSQL, які за замовчуванням прослуховують усі інтерфейси (0.0.0.0), що дозволяє підключення будь-кому ззовні.

За даними ENISA Threat Landscape 2024, близько 18% інцидентів витоку даних з БД у 2023 році були пов’язані саме з відкритими інтерфейсами без автентифікації [2]. Такі випадки особливо небезпечні в хмарному середовищі, де адміністратори можуть помилково виставити публічний доступ до сервісів через панель керування або невірно налаштовані правила брандмауеру.

Ще однією поширеною проблемою є використання облікових записів за замовчуванням, як-от root у MySQL, admin у MongoDB, або sa в MSSQL. Часто такі облікові записи залишаються без зміни пароля або з банальними комбінаціями на зразок admin123, password, що робить їх легкими мішенями для атак грубої сили.

У звіті M-Trends 2025 року вказується, що у 17% з усіх підтверджених випадків компрометації БД зловмисники отримали доступ до систем через

облікові записи з дефолтними паролями або привілеями, які не обмежувались принципом найменших прав [9].

Особливо критичною ця вразливість стає у випадках, коли адміністратори використовують одні й ті самі паролі на різних середовищах (розробка, тестування, продуктив), а також коли облікові дані зберігаються у відкритих конфігураційних файлах репозиторіїв.

За замовчуванням багато СУБД не використовують захищене з’єднання (TLS/SSL) для клієнт-серверної комунікації. Це відкриває можливість перехоплення автентифікаційної інформації або даних, що передаються, через атаки типу «людина посередині».

Хоча сучасні версії PostgreSQL, MySQL і MSSQL підтримують шифрування з'єднання, його активація часто залишається на розсуд адміністратора. У багатьох середовищах без шифрування працюють не лише вторинні вузли кластера (реплікація), а й навіть інтерфейси, які обробляють фінансові запити або авторизацію.

Політика доступу до БД часто будується за принципом «спрощення роботи», що призводить до надання користувачам надлишкових прав – наприклад SELECT, INSERT, UPDATE, DELETE і навіть EXECUTE або DROP – тоді як їм потрібен лише обмежений доступ до конкретних таблиць.

Цей тип помилки – типовий прояв людського фактора, коли через відсутність детального аналізу ролей система поступово набуває численних “слабких місць”. Порушення принципу найменших привілеїв прямо суперечить найкращим практикам безпеки, закріпленим у стандартах ISO/IEC 27001 та NIST SP 800-53.

У звіті DBIR за 2025 рік йдеться, що в 14% випадків зловмисникам вдалося проникнути глибше в систему саме через надмірні привілеї, що були надані низькорівневим обліковим записам [1].

Навіть за умови правильної початкової конфігурації відсутність системи логування доступу та дій користувачів (аудиту) ускладнює виявлення та

розслідування інцидентів. Часто організації дізнаються про витік або злам лише після того, як інформація з’являється у відкритому доступі або їх повідомляє сторонній аналітик.

Механізми реєстрацій подій в СУБД зазвичай не ввімкнені за замовчуванням або мають обмежені можливості (наприклад, запис лише критичних подій без фіксації всіх запитів). Це створює сліпі зони, які активно використовуються зловмисниками.

Висновки за розділом 1

У першому розділі було досліджено сучасний стан інформаційної безпеки в базах даних, зокрема основні загрози та вразливості. Аналіз показав, що загрози поділяються на внутрішні (пов’язані з діями співробітників) та зовнішні (атаки з боку зловмисників), причому обидві категорії мають високий потенціал для завдання значної шкоди. Серед найпоширеніших загроз виділяються несанкціонований доступ, SQL-ін’єкції, витік інформації, шкідливе ПЗ та атаки на відмову в обслуговуванні.

Також було розглянуто вразливості сучасних СУБД, які класифікуються за походженням (архітектурні, конфігураційні, інфраструктурні) та рівнем потенційної шкоди. Особливу увагу приділено програмним вразливостям ядра СУБД, механізмів автентифікації та інтерфейсів взаємодії, а також конфігураційним помилкам, які часто виникають через людський фактор.

Отримані результати підтверджують необхідність розробки ефективної методики захисту конфіденційної інформації в базах даних, яка врахувала б усі виявлені загрози та вразливості.

РОЗДІЛ 2
ПРИНЦИПИ, МЕТОДИ ТА ЗАСОБИ ЗАХИСТУ ІНФОРМАЦІЇ В БАЗАХ ДАНИХ

2.1 Принципи побудови системи захисту баз даних

Захист баз даних починається з чітко визначених принципів, які задають загальний напрям побудови надійної інформаційної системи. Ці принципи формують фундамент для впровадження технічних і організаційних рішень, що мають на меті убезпечити дані від зовнішніх і внутрішніх загроз. У фокусі знаходиться не лише технічна реалізація засобів безпеки, а й логіка їх узгодженого застосування, що передбачає цілісне бачення архітектури СУБД з погляду конфіденційності, цілісності та доступності інформації.

2.1.1 Принцип конфіденційності, цілісності та доступності

Забезпечення безпеки інформації в базах даних є процесом, що передбачає гармонійне поєднання теоретичних засад і практичних механізмів захисту. Одним із базових підходів, на якому ґрунтується сучасна концепція захисту даних, є модель так званої тріади CIA – конфіденційність (Confidentiality), цілісність (Integrity) та доступність (Availability). Цей підхід застосовується для оцінки, проектування та впровадження заходів безпеки в усіх типах інформаційних систем, зокрема в системах управління базами даних.

Кожен із цих принципів відіграє окрему роль, проте ефективна реалізація захисту можлива лише за умови їх взаємодії. Ігнорування хоча б одного з елементів тріади може призвести до серйозних наслідків, пов’язаних як із втратою інформації, так і з порушенням роботи критичних сервісів.

Конфіденційність даних передбачає захист інформації від несанкціонованого доступу. Це означає, що лише уповноважені користувачі повинні мати змогу ознайомлюватися з вмістом бази даних або виконувати певні операції над інформацією, яка в ній зберігається. У контексті баз даних це стосується як безпосереднього доступу до таблиць і полів, так і непрямого – через запити, реплікації, журнали доступу тощо.

Забезпечення конфіденційності реалізується за допомогою низки технічних і організаційних заходів. Серед них: автентифікація користувачів, шифрування даних, сегментація доступу, впровадження ролей і політик управління доступом. Особливої актуальності набуває застосування алгоритмів шифрування даних як у стані спокою (наприклад, у сховищі), так і під час передавання – це дозволяє унеможливити перехоплення інформації в мережі.

У сучасному контексті конфіденційність набуває правового значення. Відповідно до Загального регламенту ЄС із захисту даних (GDPR), організації зобов’язані впроваджувати технічні та організаційні засоби, які б забезпечували належний рівень захисту персональної інформації, зокрема під час її зберігання у базах даних [11].

Цілісність означає збереження повноти та достовірності інформації. У базах даних це передбачає, що дані не можуть бути змінені або знищені без відповідних прав і протоколювання, а також те, що будь-які внесені зміни мають бути обгрунтованими та відстежуваними.

Порушення цілісності може виникнути як унаслідок зловмисних дій (наприклад, спроби саботажу або SQL-ін’єкцій), так і через технічні збої або помилки користувачів. Особливої уваги потребують механізми транзакцій, цілісність посилань між таблицями, перевірка вхідних даних, а також ведення журналів змін (логування).

Техніки, які сприяють збереженню цілісності, включають контрольні суми, цифрові підписи, механізми блокування записів, каскадні оновлення з валідацією, а також регулярне тестування резервних копій. Надійна система

контролю версій, що забезпечує аудит змін у базі, дозволяє не лише виявити інцидент порушення цілісності, але й швидко відновити дані до стану, що передував інциденту.

Принцип доступності гарантує, що інформація у базі даних буде доступною для авторизованих користувачів у потрібний момент часу. Це один із найважливіших критеріїв, особливо в умовах критично важливих або реального часу інформаційних систем (банківські сервіси, медичні системи, системи електронного урядування тощо).

Порушення доступності може бути зумовлене як технічними проблемами (наприклад, вихід із ладу апаратного забезпечення, відмова мережі, помилки в налаштуванні прав доступу), так і цілеспрямованими атаками, зокрема DDoS або логічними бомбами, що виводять систему з ладу. Для підтримки доступності впроваджуються механізми кластеризації, реплікації даних, балансування навантаження, а також регулярне резервне копіювання й сценарії відновлення після збоїв.

2.1.2 Принцип найменших привілеїв

Один із базових принципів побудови системи захисту інформації в базах даних – це принцип найменших привілеїв (PoLP). Його сутність полягає в тому, що кожному користувачу, процесу або компоненту системи надаються лише ті мінімально необхідні права доступу, які потрібні для виконання їх функцій. Цей принцип є ключовим у профілактиці як випадкових помилок, так і зловмисних дій, і відіграє особливо важливу роль у середовищах із високим рівнем ризику або з великою кількістю користувачів.

Ідея обмеження доступу лише необхідним обсягом базується на концепції «захисту через обмеження впливу». Якщо навіть облікові дані будуть скомпрометовані, мінімізація привілеїв дозволяє знизити ймовірність критичних наслідків. Наприклад, якщо користувач має доступ лише на читання

до певної таблиці, зловмисник не зможе змінити або видалити дані, навіть маючи повний контроль над його обліковим записом.

У контексті баз даних реалізація PoLP охоплює не лише розподіл прав між користувачами, а й обмеження функцій, які виконують автоматизовані процеси (служби реплікації, системи моніторингу, скрипти обробки), кожен з яких повинен функціонувати в ізольованому середовищі з чітко визначеними повноваженнями.

Згідно з аналітичним звітом IBM Security X-Force Threat Intelligence Index 2024, 16% зломів СУБД були зумовлені тим, що облікові записи низького рівня мали надмірні повноваження, зокрема можливість виконання запитів до адміністративних функцій [12]. Це свідчить про те, що навіть у сучасних середовищах принцип PoLP часто ігнорується або реалізується формально, без належного аудиту та тестування політик доступу.

На практиці принцип найменших привілеїв (PoLP), хоч і визнаний ефективним, часто застосовується недостатньо послідовно. Це пов’язано з низкою факторів, серед яких – недостатня обізнаність персоналу, особливо в невеликих компаніях, де адміністратори поєднують кілька ролей і через це мають повний доступ до системи. Ще однією проблемою є існування застарілих облікових записів із надмірними правами, які не переглядалися протягом багатьох років.

До порушень PoLP також призводить прагнення спростити доступ для зручності роботи: адміністратори часто призначають «тимчасові» привілеї, які згодом залишаються активними на постійній основі. Окремою перешкодою є відсутність автоматизованих інструментів аудиту, які дозволили б систематично відстежувати ефективність політик доступу та оперативно виявляти відхилення від принципу найменших привілеїв у змінному середовищі.

Таким чином, незважаючи на теоретичну обґрунтованість PoLP, його реалізація ускладнюється через організаційні, технічні та людські фактори, що потребує комплексного підходу до вдосконалення механізмів контролю доступу.

2.2 Методи захисту даних

Захист інформації в сучасних СУБД потребує впровадження ефективних методів, що охоплюють ідентифікацію користувачів, регулювання прав доступу, шифрування інформації, виявлення підозрілої активності та ведення журналів подій. Поєднання цих підходів дозволяє реалізувати як базові, так і адаптивні стратегії безпеки, що враховують як поточну поведінку користувача, так і загальні умови доступу до даних. Сучасна практика показує ефективність багатофакторної автентифікації, сегментованих політик авторизації та контекстно-залежного контролю, які значно підвищують стійкість бази даних до внутрішніх і зовнішніх загроз.

2.2.1 Аутентифікація, авторизація та контроль доступу

Один із основних напрямків захисту даних у системах управління базами даних – це обмеження доступу до інформації лише уповноваженим особам. Цей механізм реалізується через аутентифікацію (перевірку особи користувача) та авторизацію (надання прав доступу до конкретних об’єктів і операцій). У сучасних СУБД ці процеси є комплексними та реалізуються з використанням багатьох рівнів безпеки, що дозволяє ефективно протидіяти загрозам зовнішнього й внутрішнього характеру.

У більшості сучасних СУБД реалізовано підтримку як локальної автентифікації (через вбудовані механізми самої бази), так і інтеграції з зовнішніми службами ідентифікації, зокрема:

- Аутентифікація через LDAP (Lightweight Directory Access Protocol) / Active Directory – дозволяє централізовано керувати обліковими записами та політиками доступу, що особливо актуально для великих організацій. Наприклад, Microsoft SQL Server, Oracle Database та PostgreSQL підтримують інтеграцію з Windows-автентифікацією або Kerberos [13].

- OAuth 2.0 та OpenID Connect – використовуються в хмарних сервісах та API, що взаємодіють із базами даних через проміжні сервіси. Це дозволяє застосовувати багатофакторну аутентифікацію навіть при доступі до БД через RESTful-інтерфейси [14].
- Аутентифікація через сертифікати – застосовується у високозахищених середовищах, зокрема в фінансовій та державній сферах. Кожен користувач повинен надати не лише пароль, а й дійсний цифровий сертифікат для встановлення з'єднання.
- IAM-інтеграція у хмарних СУБД – наприклад, Amazon RDS для PostgreSQL дозволяє автентифікацію через AWS Identity and Access Management (IAM), що виключає потребу у зберіганні паролів у самій СУБД [15].

Усі ці механізми мають спільну мету – виключити можливість доступу до системи з боку сторонніх або зламаних облікових записів, що, за даними звіту IBM Security (2024), є причиною понад 30% успішних атак на бази даних [12].

Після підтвердження особи користувача необхідно обмежити його дії відповідно до рівня привілеїв. У цьому контексті авторизація відіграє роль не менш важливу, ніж автентифікація.

Найпоширенішими моделями контролю доступу в СУБД є:

- RBAC (Role-Based Access Control) — визначає доступ залежно від ролі користувача (наприклад, "аналітик", "адміністратор", "читач"). Це дозволяє централізовано призначати політики доступу й значно спрощує аудит. PostgreSQL, Oracle та MSSQL мають вбудовану підтримку RBAC.
- DAC (Discretionary Access Control) — власник об'єкта сам вирішує, кому надавати доступ. Такий підхід має гнучкість, але складніше в управлінні на великих системах.
- MAC (Mandatory Access Control) — заснований на мітках безпеки, що жорстко регламентують доступ незалежно від бажання користувача. Використовується переважно у військових або критичних системах.

- ABAC (Attribute-Based Access Control) — сучасна модель, у якій доступ визначається не лише роллю, а й атрибутами (часом доби, IP-адресою, пристроєм тощо). Наприклад, у Google Cloud SQL реалізована політика обмеження доступу за геолокацією та політиками контексту [16].

2.2.2 Шифрування даних на різних рівнях

У контексті СУБД шифрування відіграє ключову роль у запобіганні витоку інформації навіть у випадках компрометації облікових записів, мережевого доступу або фізичного викрадення носіїв. Сучасні підходи до шифрування передбачають його реалізацію на кількох рівнях – у процесі передавання даних, у стані спокою, на рівні окремих полів або таблиць, а також з використанням апаратних засобів.

Шифрування даних під час передавання (In-Transit Encryption). Передавання даних між клієнтом і СУБД, особливо в мережевому середовищі, потребує обов’язкового захисту від перехоплення. Для цього використовується шифрування з’єднання, найчастіше на базі протоколу TLS. Підтримка TLS реалізована у більшості сучасних СУБД, зокрема PostgreSQL, MySQL, Oracle, MongoDB, MSSQL та інших [13].

Після встановлення TLS-з’єднання передача автентифікаційних даних, запитів та відповідей відбувається у зашифрованому вигляді, що унеможливлює їх перехоплення через атаки типу «людина посередині». Також можуть використовуватись сертифікати клієнта та сервера, що додатково гарантує автентичність учасників взаємодії.

У хмарних середовищах шифрування з’єднання активується за замовчуванням. Наприклад, Google Cloud SQL, Amazon RDS та Azure SQL Database автоматично пропонують TLS-з’єднання, а також підтримують контроль за версією протоколу і вимогу обов’язкового шифрування [15].

Шифрування даних у стані спокою (At-Rest Encryption). Шифрування даних, що зберігаються у базі, є другою важливою лінією захисту. Його мета — зробити дані недоступними навіть у випадку фізичного доступу до дисків, резервних копій або знімків.

Для цього використовуються технології:

- TDE — підтримується в Oracle, MSSQL, PostgreSQL (Enterprise рішення), MySQL (InnoDB). Дозволяє шифрувати файли таблиць, журналів транзакцій та бекапів без необхідності модифікації запитів користувачів.
- Шифрування на рівні файлової системи — наприклад, за допомогою BitLocker (Windows), LUKS (Linux) або cloud-native засобів, таких як AWS KMS та Azure Disk Encryption. Це забезпечує базовий рівень захисту для всіх даних на фізичних/віртуальних дисках.
- Апаратне шифрування — реалізується за допомогою спеціалізованих модулів безпеки, що відповідають стандарту FIPS 140-2 (Federal Information Processing Standard). Це рішення використовуються в банківських та державних системах, де потрібен високий рівень відповідності вимогам [17].

За даними звіту Thales Cloud Security Study 2023, 68% компаній використовують шифрування на рівні зберігання, причому TDE є найпоширенішим механізмом у середовищах з Oracle та MSSQL [18].

Шифрування даних на рівні поля або колонки (Field-Level Encryption). Інколи є потреба захистити окремі елементи даних (наприклад, номери карток, паспортні дані, адреси), зберігаючи при цьому можливість виконання запитів до інших полів без розшифрування всієї таблиці. У таких випадках застосовується шифрування на рівні полів або колонок.

Цей метод підтримується в MySQL (функції AES_ENCRYPT / AES_DECRYPT), PostgreSQL (через бібліотеки pgcrypto), MongoDB (native field-level encryption), а також через зовнішні інструменти типу Vault by HashiCorp.

Таке шифрування є більш гнучким, але потребує управління ключами, регламентів розшифрування та обмеження привілеїв. Наприклад, у MongoDB доступ до зашифрованого поля може бути наданий лише через роздільне управління ключами на сервері й клієнті.

Керування ключами шифрування. Ефективне шифрування неможливе без надійної системи керування криптографічними ключами. Більшість сучасних СУБД і хмарних платформ інтегруються з:

- KMS (Key Management Service) – Amazon KMS, Azure Key Vault, Google Cloud KMS;
- HSM (Hardware Security Module) – для високого ступеня захисту ключів;
- BYOK (Bring Your Own Key) – модель, у якій організація зберігає повний контроль над ключами, навіть в хмарі.

Контроль за життєвим циклом ключів, їхнє створення, обертання, зберігання, знищення, це все є критичним аспектом криптографічного захисту.

Виклики й обмеження. Хоча шифрування даних є потужним інструментом захисту інформації, його використання супроводжується певними обмеженнями. Одна з ключових проблем – значні обчислювальні витрати, особливо при роботі з великими обсягами даних або високонавантаженими транзакційними системами. Крім того, деякі операції, такі як сортування чи агрегація, неможливо виконати без попереднього розшифрування, що ускладнює обробку даних у зашифрованому вигляді.

Не менш важливою перешкодою є складність управління ключами шифрування, зокрема необхідність їхньої регулярної ротації відповідно до політик безпеки. Також варто враховувати, що реалізація шифрування програмними засобами без апаратного прискорення або оптимізації запитів може призвести до помітного зниження продуктивності системи.

Через ці обмеження для кожного конкретного проєкту необхідно ретельно аналізувати вимоги до безпеки, оцінювати потенційні ризики та наявні ресурси,

щоб визначити оптимальний рівень шифрування. Це дозволить знайти баланс між захистом даних і ефективністю роботи системи.

2.2.3 Маскування та дедуплікація даних

У складній екосистемі сучасних інформаційних систем бази даних використовуються не лише для оперативної обробки запитів, але й для тестування, аналітики, резервного копіювання та відновлення після збоїв. Це створює вимоги до методів, здатних обмежити видимість або повторне використання даних без загрози для цілісності чи продуктивності системи. Серед таких методів особливе місце займають маскування даних та дедуплікація.

Маскування – це метод обробки даних, при якому реальні значення замінюються на модифіковані (штучні або приховані), зберігаючи при цьому їх формат та структуру. Це дозволяє використовувати дані у тестових, аналітичних або навчальних середовищах без розкриття реальної чутливої інформації.

Маскування особливо ефективне при передаванні копій баз зовнішнім підрядникам або в разі створення тестових середовищ із реальними структурами таблиць. У Oracle Database реалізовано механізм Data Redaction, який дозволяє приховувати або замінювати значення полів залежно від запитувача. PostgreSQL реалізує динамічне маскування через механізм Row-Level Security і створення представлень (views).

Дедуплікація – це процес виявлення й усунення надмірного дублювання даних у базах. З одного боку, вона зменшує обсяг збережених даних, а з іншого – мінімізує потенційні точки витоку, адже дублікати чутливої інформації збільшують площу атаки.

Є два основні типи дедуплікації: на рівні записів – виявлення однакових записів у таблицях (наприклад, повторні записи клієнтів); на рівні

сховищ/бекапів – у системах зберігання, де ідентичні блоки не дублюються фізично.

У середовищах великих СУБД дедуплікація може бути реалізована за допомогою ETL-процесів (Extract, Transform, Load), регулярних процедур очистки або використання унікальних ключів. У хмарних сервісах дедуплікація є вбудованою функцією – наприклад, Azure Backup або AWS Backup застосовують інкрементальні знімки із виявленням повторюваних блоків [19].

2.2.4 Аудит і логування дій користувачів

Однією з найважливіших складових безпеки сучасних систем управління базами даних є аудит та логування дій користувачів. На відміну від реактивних заходів безпеки, які застосовуються після виявлення інциденту, аудит дає змогу виявити підозрілу активність, запобігти витоку даних та документально зафіксувати всі дії користувачів і адміністраторів, що особливо важливо у середовищах із підвищеними вимогами до відповідності законодавству.

Механізми аудиту дозволяють вирішувати відразу кілька завдань:

- Фіксація подій безпеки – спроби входу, відмови в авторизації, зміни в конфігурації, доступ до чутливих таблиць.
- Контроль привілейованих користувачів – зокрема, дії адміністраторів, які мають розширені права доступу.
- Розслідування інцидентів – повне відтворення хронології подій у разі витоку, спроби вторгнення або саботажу.
- Дотримання вимог нормативних актів, які прямо передбачають наявність механізмів аудиту і збереження журналів [20].

За даними звіту Verizon DBIR за 2025 рік, відсутність ефективного логування була чинником, який ускладнив розслідування понад 27% кіберінцидентів, пов'язаних із базами даних [1].

У великих інфраструктурах, де існує багато екземплярів баз даних, ефективним є централізований аудит за допомогою SIEM-рішень. До прикладу, Splunk, Elastic Stack (ELK), IBM QRadar, Azure Sentinel дозволяють збирати журнали з багатьох джерел, аналізувати їх у реальному часі та автоматично генерувати сповіщення у разі аномальної активності.

Такі системи можуть поєднувати події з БД із мережею, операційною системою, програмними сервісами, що значно розширює аналітичні можливості служби безпеки. Також дедалі більшого поширення набувають поведінкові системи виявлення (User and Entity Behavior Analytics), які вивчають звичні патерни дій користувачів та сигналізують про нетипову поведінку.

Сам факт ведення журналів подій ще не забезпечує повноцінного захисту інформації. Для ефективного функціонування системи аудиту необхідно дотримуватись низки критично важливих умов.

По-перше, журнали повинні зберігатися у спеціально захищеному середовищі, яке виключає можливість їх несанкціонованої зміни будь-якими користувачами, включаючи адміністраторів системи.

По-друге, обов'язковим є використання механізмів цифрового підписування логів, що робить неможливим їх непомітну модифікацію або фальсифікацію.

Не менш важливим аспектом є чітке визначення політик архівування та зберігання журналів, які мають відповідати як внутрішнім регламентам компанії, так і зовнішнім нормативним вимогам. Крім того, система повинна включати надійні механізми аварійного відновлення журналів, що дозволить уникнути втрати критично важливої інформації у разі технічних збоїв або апаратних відмов. У багатьох випадках рекомендовано вивантажувати журнали в окремий вузол (log server), який не є частиною продуктивного середовища.

2.3 Засоби забезпечення безпеки баз даних

Для реалізації політик інформаційної безпеки важливо залучати технічні засоби, які дозволяють контролювати роботу СУБД у режимі реального часу, реагувати на інциденти та забезпечувати збереження даних. Серед таких засобів – як традиційні інструменти моніторингу та виявлення загроз, так і сучасні хмарні сервіси, здатні масштабуватися та інтегруватися в розподілені системи. Використання цих засобів підсилює захисний потенціал СУБД і дозволяє реалізовувати комплексний підхід до захисту конфіденційної інформації.

2.3.1 Інтеграція з SIEM/IDS/IPS

Для формування повноцінної багаторівневої архітектури безпеки необхідна інтеграція з зовнішніми системами моніторингу та виявлення загроз, зокрема із SIEM-системами, а також системами виявлення та запобігання вторгненням (IDS/IPS). Таке поєднання дає змогу організаціям не лише збирати події безпеки, а й аналізувати їх у реальному часі, виявляти аномалії, реагувати на атаки та будувати контекстну картину загроз у межах усієї IT-інфраструктури.

Системи класу SIEM (Security Information and Event Management) здійснюють централізовану агрегацію логів, подій безпеки та телеметрії з різноманітних джерел – серверів, мережевого обладнання, застосунків, хостів і, зокрема, баз даних. Вони дозволяють: виявляти несанкціоновані спроби доступу до БД; контролювати активність привілейованих облікових записів; виявляти SQL-ін’єкції та аномальні запити; виконувати кореляційний аналіз поведінки користувачів; генерувати сповіщення про інциденти та автоматизувати реагування.

Серед популярних комерційних та open-source SIEM-платформ, які підтримують інтеграцію з базами даних, — Splunk, IBM QRadar, LogRhythm, Elastic SIEM, AlienVault OSSIM [21].

Наприклад, у Splunk є спеціальні додатки для PostgreSQL, Oracle та MSSQL, які дозволяють збирати журнали подій (log_collector, auditd, syslog) та аналізувати їх за допомогою створених правил кореляції [22]. У QRadar доступні конектори для MongoDB та MySQL, які автоматично розпізнають тип події та класифікують її за рівнем ризику.

На відміну від SIEM, які виконують глибокий аналіз практично у режимі реального часу, *системи IDS/IPS* фокусуються на виявленні та запобіганні вторгненням у режимі реального часу. IDS (Intrusion Detection System) лише виявляє підозрілу активність, тоді як IPS (Intrusion Prevention System) може також її блокувати.

У контексті СУБД IDS/IPS системи найчастіше використовуються для: моніторингу мережевого трафіку, виявлення підозрілих запитів або аномалій у з'єднаннях, блокування підозрілих спроб експлуатації вразливостей СУБД (наприклад, SQL-ін’єкцій, brute-force, scan-атак).

У великих організаціях застосовуються системи на кшталт Snort, Suricata, Cisco Secure IPS, які здатні працювати на межі мережі, контролюючи підключення до серверів баз даних. IDS може бути налаштовано на виявлення патернів запитів, що характерні для ін’єкцій, або раптового зростання обсягу операцій від одного користувача [21].

Інтеграція СУБД з системами SIEM, IDS та IPS, попри свої очевидні переваги, потребує ретельного підготовчого етапу. Першочерговим завданням є стандартизація форматів логування – дані з бази мають бути адаптовані під вимоги SIEM-системи для коректної обробки та аналізу. Критично важливим є налаштування фільтрації подій, щоб система фіксувала лише релевантні для безпеки події. Це дозволить уникнути інформаційного шуму та знизити навантаження на інфраструктуру.

Окремим викликом є продуктивність системи. Агрегація подій з великих баз даних може суттєво навантажити ресурси, тому необхідно заздалегідь протестувати систему та забезпечити можливість її масштабування. Це дозволить уникнути зниження продуктивності основних бізнес-процесів при роботі систем моніторингу.

2.3.2 Хмарні засоби захисту баз даних

Інтенсивне впровадження хмарних технологій у корпоративному та державному секторах призвело до переосмислення підходів до захисту інформаційних ресурсів, зокрема баз даних. Відмова від локального зберігання на користь хмарних сервісів вимагає не лише технічного, а й концептуального оновлення архітектури безпеки. Зважаючи на характер хмарного середовища – розподіленість, динамічність і постійну доступність через мережу – ключовим чинником захищеності є правильна конфігурація вбудованих сервісів безпеки, які надаються постачальниками хмарної інфраструктури.

Сучасні хмарні провайдери – такі як Amazon Web Services (AWS), Microsoft Azure і Google Cloud Platform (GCP) – пропонують широкий набір спеціалізованих засобів захисту, інтегрованих у власні платформи керування базами даних. Ці засоби охоплюють як класичні напрями захисту (аутентифікація, шифрування, контроль доступу), так і новітні технології: поведінковий аналіз, автоматизовану аналітику загроз, інтеграцію з політиками відповідності.

Захист баз даних у середовищі Amazon Web Services. У структурі AWS ключову роль у роботі з реляційними базами даних відіграє сервіс Amazon RDS (Relational Database Service), який підтримує популярні СУБД — PostgreSQL, MySQL, Oracle, SQL Server та інші. Для забезпечення захисту в Amazon реалізовано декілька рівнів контролю.

По-перше, впроваджено централізоване управління автентифікацією через IAM (Identity and Access Management). Це дозволяє визначати, які користувачі або служби можуть виконувати конкретні операції з базою, а також усунути необхідність зберігання облікових даних безпосередньо у СУБД. По-друге, усі з'єднання з базою підтримують шифрування за протоколом TLS, а дані у стані спокою можуть бути зашифровані автоматично через AWS KMS (Key Management Service), що дає змогу централізовано керувати ключами.

Додатково реалізовано ізоляцію інфраструктури баз даних за допомогою віртуальних приватних мереж (VPC), у межах яких можна обмежити доступ до інстансів за IP-адресою або роллю. Служба Amazon GuardDuty для RDS виконує аналітичний моніторинг запитів і спроб підключення, виявляючи аномальну активність, що може вказувати на компрометацію або спробу SQL-ін'єкції [15].

Механізми безпеки у Microsoft Azure. У хмарній екосистемі Azure для розміщення СУБД використовуються сервіси Azure SQL Database, Azure Database for PostgreSQL/MySQL та Cosmos DB. Захист цих сервісів реалізовано відповідно до принципів «захисту за замовчуванням» та принципу найменших привілеїв.

Для автентифікації користувачів застосовується інтеграція з Azure Active Directory, яка дозволяє централізовано управляти правами доступу та політиками багатофакторної аутентифікації. Дані в Azure SQL шифруються на рівні сховища за допомогою TDE, що охоплює як самі таблиці, так і журнали транзакцій.

Ключовим компонентом у сфері виявлення загроз є служба Microsoft Defender for SQL, яка автоматично аналізує поведінкові шаблони користувачів та запитів, попереджаючи про можливі інциденти безпеки. Також реалізовано функцію Always Encrypted, яка дозволяє зберігати чутливі дані в зашифрованому вигляді, причому сам сервер не має доступу до розшифрованих значень – ключі зберігаються виключно на стороні клієнта [23].

Azure також підтримує створення політик доступу на основі умов (наприклад, обмеження доступу за географічним положенням чи часом), а також аудит усіх операцій через службу Azure Monitor.

Google Cloud Platform: захист Cloud SQL та Spanner. У середовищі Google Cloud основними сервісами для роботи з базами даних є Cloud SQL (керовані СУБД) та Cloud Spanner (розподілена реляційна СУБД). Платформа орієнтована на автоматизацію безпеки, з акцентом на контроль доступу та вбудоване шифрування.

Усі бази Google Cloud шифруються за замовчуванням за допомогою AES-256. Для ключів шифрування доступні дві моделі: використання керованих ключів (Google-managed keys) або власних ключів користувача (CMEK — Customer-Managed Encryption Keys), що реалізує модель BYOK (Bring Your Own Key). Інтеграція з Cloud KMS та підтримка HSM-модулів (Hardware Security Module) забезпечує високий рівень криптографічного контролю [16].

Керування доступом реалізовано через IAM (Identity and Access Management), що дозволяє призначати права доступу не лише користувачам, а й окремим сервісам. Додатково забезпечено ізоляцію за допомогою VPC Service Controls, які обмежують експорт даних за межі визначеної зони довіри.

Google Cloud підтримує систему Cloud Audit Logs, яка фіксує всі спроби доступу, зміни конфігурацій та запити до баз. Поєднання з Chronicle або іншими SIEM-рішеннями дозволяє реалізувати повний цикл моніторингу безпеки.

2.3.3 Використання DLP, DAM, EDR для моніторингу та реагування

Враховуючи складність, багаторівневість і динамічність сучасних інфраструктур, необхідно застосовувати спеціалізовані рішення, що дозволяють не лише виявляти загрози, а й реагувати на них у режимі реального часу, запобігати витокам, відстежувати дії користувачів і контролювати безпеку даних на всіх етапах їх життєвого циклу. У цьому контексті особливої уваги

заслуговують три класи інструментів: DLP (Data Loss Prevention), DAM (Database Activity Monitoring) і EDR (Endpoint Detection and Response).

DLP-системи орієнтовані на виявлення, моніторинг і блокування спроб несанкціонованого передавання або копіювання чутливих даних. Вони застосовуються як на рівні кінцевих точок, так і на рівні мережевої інфраструктури. У випадку з базами даних ці системи можуть аналізувати запити до БД, експорт даних або з'єднання з підозрілими ресурсами.

Класичний приклад – якщо користувач намагається скопіювати таблицю з номерами платіжних карток і відправити її електронною поштою, DLP автоматично виявляє цей шаблон (наприклад, регулярний вираз формату картки) та блокує операцію.

Сучасні рішення як-от Symantec DLP, McAfee Total Protection for DLP або Forcepoint інтегруються з системами баз даних, проводячи інспекцію SQL-запитів, моніторинг запитів на експорт, а також класифікацію даних за чутливістю [24].

Важливим компонентом є Data Discovery, що дозволяє виявляти конфіденційну інформацію в структурах бази та формувати політики її обробки відповідно до вимог регуляторів, таких як GDPR або PCI DSS.

DAM – це спеціалізовані рішення для глибокого аналізу всіх дій у СУБД, зокрема відслідковування запитів, змін схем, операцій із даними, а також адміністраторських втручань. Вони є важливим доповненням до внутрішнього аудиту СУБД, оскільки дозволяють: виявляти аномальні або позаграфікові дії; формувати звіти про дії користувачів у реальному часі; контролювати доступ до таблиць із конфіденційною інформацією; зберігати історію дій навіть у випадку, якщо внутрішній аудит вимкнено або обійдений.

Прикладами рішень DAM-класу є IBM Guardium, Imperva SecureSphere for Databases, Oracle Audit Vault. Вони дозволяють формувати звіти про те, хто, коли і до яких саме даних звертався, з якої IP-адреси та з якими результатами [26].

DAM-системи часто є обов’язковим компонентом у банківських, медичних та державних установах, де вимагається повний аудит доступу до чутливих даних.

Хоча *EDR-системи* традиційно асоціюються із захистом робочих станцій і серверів, їх роль у захисті баз даних останніми роками зросла. Це пояснюється тим, що інциденти безпеки часто починаються саме з компрометації кінцевої точки, з якої згодом здійснюється доступ до СУБД.

EDR-рішення, такі як CrowdStrike Falcon, Microsoft Defender for Endpoint, SentinelOne дозволяють: виявляти запуск шкідливих процесів, які можуть використовувати облікові дані для доступу до БД; фіксувати підозрілі з'єднання до портів СУБД; ізолювати комп’ютери, з яких іде несанкціонована взаємодія з БД; ідентифікувати lateral movement – тобто переміщення зловмисника між вузлами в межах мережі [25].

Таким чином, EDR дозволяє відстежувати початковий вектор атаки та захищати бази даних через превентивне блокування дій на рівні операційної системи.

Висновки за розділом 2

У другому розділі було розглянуто основні принципи, методи та засоби захисту конфіденційної інформації в базах даних. Аналіз показав, що ефективний захист даних ґрунтується на комплексному підході, який поєднує теоретичні засади з практичними механізмами реалізації. Ключовим аспектом є забезпечення тріади CIA (Confidentiality, Integrity, Availability), яка визначає базові вимоги до безпеки: конфіденційність, цілісність і доступність даних.

Важливу роль у захисті інформації відіграють сучасні технології, такі як маскування даних для тестування та аналітики, дедуплікація для оптимізації сховищ, а також інструменти аудиту та логування, які забезпечують прозорість усіх операцій з даними. Особливу увагу приділено інтеграції баз даних із

зовнішніми системами безпеки, зокрема SIEM, IDS/IPS, DLP, DAM та EDR, що дозволяє формувати багаторівневу систему захисту та оперативно реагувати на загрози.

У хмарних середовищах захист баз даних реалізується через вбудовані сервіси провідних провайдерів (AWS, Azure, GCP), які автоматизують шифрування, управління доступом та моніторинг аномальної активності. Проте використання цих технологій супроводжується викликами, такими як складність управління ключами шифрування, обмеження продуктивності та необхідність адаптації до динамічного середовища.

Результати дослідження підтверджують, що ефективна методика захисту конфіденційної інформації в базах даних має ґрунтуватися на комплексному підході, який поєднує технічні, організаційні та нормативні заходи.

РОЗДІЛ 3
РОЗРОБКА ТА ВПРОВАДЖЕННЯ МЕТОДИКИ ЗАХИСТУ КОНФІДЕНЦІЙНОЇ ІНФОРМАЦІЇ В СУБД

3.1 Опис цільової платформи для реалізації методики

Системи управління базами даних є ключовим елементом інформаційних систем, що оперують конфіденційною інформацією. В цьому контексті актуальним є вибір між різними реалізаціями СУБД, зокрема між мінімалістичною SQLite та повнофункціональною серверною PostgreSQL. Обидві ці системи широко застосовуються, однак відрізняються функціональними можливостями, архітектурною побудовою та підходами до захисту даних.

PostgreSQL є серверною реляційною СУБД із клієнт-серверною архітектурою. Вона передбачає чітке розмежування між користувачами, має повноцінну систему керування доступом, механізми шифрування, аудит подій і підтримку розширень безпеки [13]. З іншого боку, SQLite – це вбудована СУБД, яка функціонує у вигляді бібліотеки, що вбудовується безпосередньо в застосунок. Вона не передбачає відокремленого процесу сервера, не має вбудованого механізму багатокористувацького доступу, а відповідно й класичної системи авторизації [27].

PostgreSQL реалізує ролеву модель безпеки, яка дозволяє створювати облікові записи користувачів, визначати для них різні привілеї на рівні бази, таблиць, функцій, а також підтримує механізми більш тонкого контролю, зокрема Row-Level Security (RLS). SQLite ж покладається на механізми доступу до файлів, надані операційною системою, тобто фактично вся безпека ґрунтується на налаштуваннях файлової системи.

У PostgreSQL підтримка шифрування реалізується на кількох рівнях:

- Передача даних може бути захищена через SSL/TLS;
- Зберігання даних може забезпечуватись сторонніми модулями, зокрема pgcrypto, pgsodium, або зовнішніми інструментами з повнодисковим шифруванням;
- Шифрування окремих колонок дозволяє гнучко захищати тільки чутливу інформацію, що мінімізує вплив на продуктивність.

У базовій конфігурації SQLite не забезпечує жодного шифрування. Для реалізації захищеного зберігання необхідно використовувати сторонні рішення, наприклад, SQLCipher, який базується на SQLite і додає шифрування файлу бази даних із використанням AES. Однак таке шифрування застосовується до всього файлу цілком і не підтримує вибіркове шифрування колонок чи гнучке управління ключами [28].

PostgreSQL має розвинуту систему журналювання подій. Вона дозволяє реєструвати запити, зміни структури, помилки автентифікації тощо. За допомогою розширень на кшталт pgaudit або pg_stat_statements можливе ведення детального аудиту дій користувачів. SQLite таких можливостей не має, і будь-який аудит необхідно реалізовувати на рівні застосунку [13].

PostgreSQL підтримує механізми ізоляції середовищ, а також дозволяє використовувати політики доступу, обмеження IP-адрес, автозавершення сесій тощо. Завдяки цьому значно простіше вбудовувати її в корпоративні рішення з централізованими засобами безпеки, зокрема SIEM-системами або системами виявлення атак (IDS/IPS) .

У SQLite захист від атак реалізовується лише опосередковано – через налаштування середовища, у якому працює застосунок. Вона вразлива до атак через ненадійний ввід (наприклад, SQL-ін’єкції), якщо не використано параметризовані запити.

Таким чином, PostgreSQL забезпечує суттєво вищий рівень безпеки порівняно з SQLite. Вона пропонує більш гнучкі й масштабовані механізми керування доступом, підтримку шифрування, аудит та засоби запобігання

атакам. У той же час SQLite залишається зручним рішенням для простих, автономних застосунків, однак потребує додаткових заходів безпеки з боку операційної системи або сторонніх модулів. Зважаючи на поставлену мету захисту конфіденційної інформації, основна увага в подальшому розділі буде приділена реалізації методики саме на базі PostgreSQL.

3.2 Реалізація методики автентифікації та авторизації

Впровадження засобів автентифікації та авторизації є одним із базових етапів забезпечення інформаційної безпеки в СУБД. Саме ці механізми визначають, хто має доступ до даних і в якому обсязі. Коректна реалізація облікових записів, ролей і дозволів значно знижує ризик несанкціонованих дій. Тому налаштування політик доступу на основі сучасних підходів є необхідною умовою для побудови надійної моделі взаємодії користувачів із системою.

3.2.1 Створення користувачів і ролей у PostgreSQL

Одним із базових механізмів забезпечення захисту інформації в системах управління базами даних є коректна організація автентифікації користувачів та розмежування їхніх повноважень. У PostgreSQL ця функціональність реалізована через систему ролей, які замінюють поняття користувачів у традиційних СУБД і забезпечують гнучке управління доступом.

У PostgreSQL роль – це об’єкт бази даних, який може виконувати роль як користувача (тобто суб’єкта, що входить у систему), так і групи (тобто сукупності прав або інших ролей). Такий підхід забезпечує гнучку модель успадкування прав та централізоване керування привілеями.

Роль може:

- бути здатною входити в систему (`LOGIN`);
- володіти правом створення баз (`CREATEDB`);

- створювати інші ролі (`CREATEROLE`);
- бути суперкористувачем (`SUPERUSER`);
- володіти будь-якими специфічними привілеями на об’єкти в базі.

Створення ролей у PostgreSQL виконується за допомогою інструкції `CREATE ROLE` або `CREATE USER` (остання є синонімом, який автоматично додає атрибут `LOGIN`) [13]. Наприклад:

```
CREATE ROLE analyst WITH LOGIN PASSWORD 'secure_password';
```

Також варто зазначити що створення ролей, і багато інших налаштувань бд, можна реалізувати через безкоштовний візуальний інструмент pgAdmin4, приклад зображено на рис. 3.1:

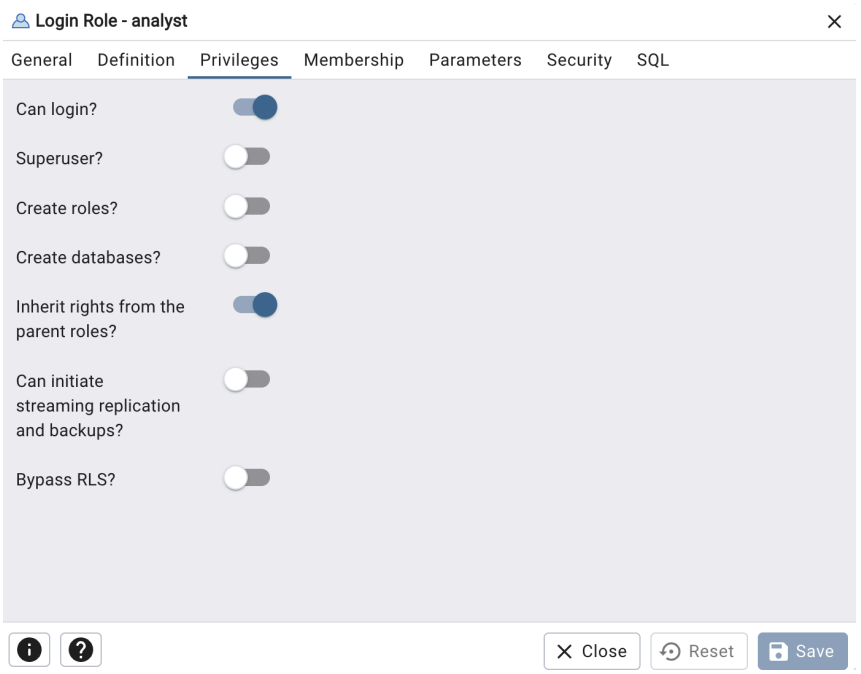

Рисунок 3.1 – надання привілеїв для ролі у середовищі pgAdmin4

У цьому прикладі створено роль із можливістю входу в систему та паролем. Вона не має жодних додаткових привілеїв, що відповідає принципу найменших привілеїв (PoLP).

Також можна створити роль без права входу, яка слугуватиме груповою роллю:

```
CREATE ROLE readonly;
```

Потім цю роль можна призначити конкретному користувачеві:

```
GRANT readonly TO analyst;
```

Права на об’єкти бази (таблиці, функції, представлення, схеми) призначаються через інструкції `GRANT` та `REVOKE`. Наприклад:

```
GRANT SELECT ON TABLE employees TO readonly;
```

Це дозволяє ролі readonly (а значить, і користувачу analyst) лише читати дані з таблиці employees.

Також варто обмежити доступ до інших об’єктів, якщо це потрібно:

```
REVOKE ALL ON TABLE employees FROM analyst;
```

Завдяки успадкуванню прав і ролей, у PostgreSQL можливо організувати багаторівневу ієрархію доступу. Наприклад:

- admin — суперкористувач;
- data_editor — має права на `INSERT`, `UPDATE`, `DELETE`;
- data_reader — має права лише на `SELECT`;
- auditor — доступ до журналів, без права змінювати дані.

Така модель дозволяє централізовано контролювати доступ до різних частин бази, не призначаючи права окремо кожному користувачеві.

У PostgreSQL реалізовано кілька можливостей для додаткового захисту облікових записів:

- SSL-з’єднання. Автентифікація може вимагати встановлення TLS-захищеного каналу (hostssl у файлі pg_hba.conf).
- Встановлення терміну дії пароля. Можна задати дату закінчення пароля:

```
ALTER ROLE analyst VALID UNTIL '2025-12-31';
```

- Блокування ролі. Тимчасово або назавжди заблокувати роль:

```
ALTER ROLE analyst NOLOGIN;
```

- IP-фільтрація. Файл pg_hba.conf дозволяє обмежувати підключення на основі IP-адреси [29].

Для досягнення високого рівня безпеки рекомендується: створювати окремі ролі для кожної категорії доступу; уникати використання `SUPERUSER` у щоденній роботі; не надавати ролям зайвих прав на об'єкти; використовувати `REASSIGN OWNED` та `DROP OWNED` при видаленні ролей; періодично перевіряти список ролей і їхні привілеї (\du в psql).

3.2.2 Впровадження принципу найменших привілеїв (PoLP)

Надання надмірних привілеїв збільшує ризик помилок, зловживань або компрометації даних, тому впровадження PoLP є обов'язковим для побудови надійної системи захисту баз даних.

Реалізація PoLP дозволяє:

- мінімізувати наслідки потенційного взлому окремого облікового запису;
- обмежити вплив помилок операторів або скриптів;
- запобігти горизонтальному та вертикальному ескалаційним атакам;
- сформувати чітку ієрархію доступу до об'єктів бази;
- полегшити аудит дій користувачів.

Етапи впровадження PoLP у PostgreSQL:

Ідентифікація категорій користувачів. На початковому етапі слід класифікувати користувачів за ролями, функціоналом та рівнем відповідальності.

Створення мінімальних привілейованих ролей. Згідно з виявленими категоріями створюються окремі ролі з максимально обмеженим набором прав. Кожна роль має обмеження лише на ті таблиці, функції чи схеми, з якими вона повинна працювати.

Уникнення "широких" прав доступу.

Забороняється:

- надання глобальних прав (`GRANT ALL`) без реальної потреби;
- присвоєння прав на всю базу, якщо користувач працює лише з окремою схемою;
- використання `SUPERUSER`, `CREATEROLE`, `CREATEDB` поза межами адміністративного контексту;
- автоматичне успадкування прав без перевірки потреб.

Розмежування прав читання, запису та керування. Кожна роль має мати лише окремий тип доступу. Наприклад, у зв'язці `data_reader`/`data_editor` права повинні бути чітко розмежовані.

3.2.3 Використання Row-Level Security (RLS) для обмеження доступу до рядків

Одним з надзвичайно корисних механізмів авторизації в PostgreSQL є Row-Level Security (RLS) — система, яка дозволяє накладати політики доступу не лише на об'єкти бази даних, а й на окремі рядки в таблицях [13]. Це дає змогу реалізовувати надзвичайно гнучкі сценарії доступу, зокрема в багатокористувацьких середовищах, де користувачі працюють з даними в одній таблиці, але повинні бачити лише власні записи.

У традиційних моделях авторизації контроль доступу відбувається на рівні таблиці або схеми. Це означає, що користувач або має доступ до всієї таблиці, або не має взагалі. У багатьох реальних системах така модель є надто грубою. Наприклад, у медичній, фінансовій чи освітній сфері користувачам дозволено бачити лише «свої» дані або дані свого підрозділу.

Механізм RLS дозволяє визначити політики доступу до окремих рядків залежно від вмісту рядка і властивостей користувача. Відтак, авторизація стає динамічною, і система може автоматично обмежити доступ без потреби реалізовувати фільтрацію в застосунку.

Для запобігання можливим обходам RLS, слід враховувати:

- Обмежити доступ до основної таблиці. Доступ повинен надаватись лише після застосування RLS-політик.
- Уникати функцій, які обробляють усі дані без урахування політик (наприклад, `COPY` або неавторизовані `JOIN`).
- Активувати `FORCE ROW LEVEL SECURITY` для примусового застосування політик навіть суперкористувачем:

```
ALTER TABLE orders FORCE ROW LEVEL SECURITY;
```

Це особливо важливо у випадках, коли адміністратори не повинні мати доступу до конфіденційної інформації без спеціального погодження.

Обмеження та особливості:

- Не підтримується у SQLite. Механізм RLS є унікальним для PostgreSQL серед відкритих СУБД.
- Може впливати на продуктивність, якщо таблиця велика і фільтрація складна.
- Не замінює політики на рівні колонок. RLS працює лише з рядками.

3.2.4 Захист та зберігання облікових даних

Облікові дані – це чутлива інформація, яка дає змогу користувачам і службам отримувати доступ до системи баз даних. Вони включають у себе логіни, паролі, ключі автентифікації, токени, а в деяких випадках – параметри доступу до зовнішніх сервісів. Компрометація таких даних може призвести до повного контролю над системою, витоку конфіденційної інформації, а також підміни або видалення критичних даних. Тому захист облікових даних є одним з ключових елементів методики безпеки.

Зберігання паролів у PostgreSQL. PostgreSQL не зберігає паролі користувачів у відкритому вигляді. Замість цього, під час створення облікового

запису або зміни пароля, система зберігає його хеш-значення (наприклад рис. 3.2), що створюється за допомогою алгоритмів SHA-256 або SCRAM-SHA-256.

Перевірка методу зберігання пароля:

```
SELECT rolname, rolpassword FROM pg_authid;
```

```
 readonly                |
 analyst                 | SCRAM-SHA-256$4096:gWZmXeR6Dwvg4Ztx/dDnSA==$FgeGnGrooo3gAfjjR55kXKuGlD0
wm8r3SJAfiuieukE=:3AHIOpPgYunkyYOKt9lSVL1wKJpoh3KkvdSgzhNI2Sc=
```

Рисунок 3.2 – демонстрація зберігання паролів у PostgreSQL

Захист доступу до файлу pg_hba.conf. Файл pg_hba.conf керує дозволами на підключення клієнтів до СУБД. У ньому зазначається, який тип автентифікації застосовується (наприклад, scram-sha-256, md5, peer, trust), для яких IP-адрес і баз даних. Наприклад можна визначити такі правила:

```
host            all             all             127.0.0.1/32            scram-sha-256
```

Де `host` вказує тип з'єднання (підключення через TCP/IP), перший `all` застосовується до всіх БД на сервері, другий `all` застосовується для всіх користувацьких ролей, `127.0.0.1/32` – дозволені IP-адреси, `scram-sha-256` – метод аутентифікації.

Для забезпечення безпеки:

- використовуйте лише шифровані методи автентифікації (наприклад, scram-sha-256);
- уникайте trust, який дозволяє доступ без перевірки пароля;
- обмежуйте доступ до цього файлу (chmod 600) і моніторьте його зміни.

Зберігання облікових даних застосунків. У багатьох випадках застосунки повинні підключатись до бази даних від імені службової ролі. У цьому випадку логін і пароль часто зберігаються в конфігураційних файлах. Рекомендації стосовно зберігання: ніколи не зберігати паролі у відкритому вигляді в

репозиторіях або середовищах CI/CD; використовувати шифрування або перемінні середовища; зберігати секрети в системному сховищі або через зовнішні менеджери (Vault, AWS Secrets Manager тощо).

Захист від атак при автентифікації. Рекомендується: обмежити кількість спроб входу (через утиліти логування або зовнішній firewall); використовувати SSL-з'єднання між клієнтом і сервером для запобігання перехопленню облікових даних; впровадити IP-фільтрацію, дозволяючи підключення лише з певних адрес; використовувати багатофакторну автентифікацію при інтеграції з зовнішніми сервісами.

Шифрування облікових даних у SQLite. У базовій версії SQLite не реалізовано механізму користувачів, автентифікації або зберігання паролів. Захист доступу до бази реалізується через обмеження доступу до файлу та використання SQLCipher – розширення SQLite із підтримкою AES-шифрування повного файлу бази. Приклад створення зашифрованої бази в SQLCipher:

```
PRAGMA key = 'super_secure_key';
```

У такій конфігурації фактичний обліковий запис – це файл бази. Тому безпека повністю залежить від зберігання ключа і контролю прав доступу до файлів [30].

Резервування паролів у PostgreSQL. Файл .pgpass може використовуватись для автоматизації підключення до PostgreSQL. Він містить:

```
hostname:port:database:username:password
```

Але все ж рекомендується уникати його використання, якщо можливо – перевага має надаватись шифрованому сховищу або інтеграції з системою автентифікації.

3.2.5 Аудит подій автентифікації

Аудит дій користувачів, зокрема подій автентифікації, дозволяє реєструвати факти входу, спроби підключення, помилки авторизації та інші

критичні події, пов’язані з доступом до бази даних. На основі таких даних можна виявляти зловмисну активність, проводити розслідування інцидентів безпеки та відстежувати відповідність політикам доступу.

Стандартне логування подій у PostgreSQL. Журналювання подій автентифікації налаштовується у конфігураційному файлі postgresql.conf. Для аудиту слід активувати такі параметри:

```
log_connections = on
log_disconnections = on
log_hostname = on
log_line_prefix = '%m [%p] %u@%d '
```

де `log_connections` фіксує кожне підключення до бази; `log_disconnections` фіксує момент завершення сесії; `log_hostname` дозволяє бачити ім’я хоста користувача; `log_line_prefix` налаштовує формат виводу для зручнішого аналізу логів [13].

Записи зберігаються у файлі postgresql.log, який розташований у директорії логів PostgreSQL. Наприклад:

```
2025-05-29 20:55:51.765 EEST [61131] @ LOG:  received smart shutdown request
2025-05-29 20:55:56.773 EEST [61145] maksimvezdeckij@postgres FATAL:  terminating connection due to unexpected postmaster exit
```

Реєстрація помилок автентифікації. Помилкові спроби входу (наприклад, з неправильним паролем) також фіксуються:

```
2025-05-29 20:28:37.650 EEST [60770] maksimvezdeckij@postgres FATAL:  password authentication failed for user "maksimvezdeckij"
```

Подібні події дозволяють виявити атаки типу brute force або несанкціоновані спроби доступу. Рекомендується налаштувати механізми сповіщення про велику кількість невдалих спроб.

Розширення для аудиту pgaudit. Для детальнішого моніторингу автентифікації та інших дій користувачів у PostgreSQL існує модуль pgaudit. Він дозволяє: записувати інформацію про виконані запити; ідентифікувати користувача, що виконує кожну операцію; бачити повні SQL-запити у логах; контролювати не лише автентифікацію, а й використання функцій, таблиць, транзакцій тощо [13].

Автоматичний аналіз логів. Для аналізу подій автентифікації логічно використовувати системні інструменти (grep, awk, log, cron), сторонні утиліти (pgBadger, pgAuditViewer), інтеграцію із системами моніторингу.

Приклад аналізу підключень за допомогою grep:

```
% grep "connection authorized" /opt/homebrew/var/log/postgresql@14.log | cut -d' ' -f1-6
2025-05-29 19:26:29.989 EEST [59246] db_analyst@mydb LOG:
```

SQLite обмеженість можливостей аудиту. У SQLite відсутні вбудовані механізми аудиту. У разі потреби моніторингу доступу до бази даних, є кілька можливих шляхів: використання тригерів на `INSERT`, `UPDATE`, `DELETE`, які фіксують зміни в лог-таблицях; журналювання на рівні застосунку (наприклад, ведення власного audit-log); моніторинг файлового доступу через засоби ОС [27].

3.3 Реалізація шифрування даних

Шифрування виконує роль останнього рубежу захисту, забезпечуючи недоступність інформації навіть у разі компрометації системи. Його застосування дає змогу зберігати чутливі дані у захищеному вигляді як у процесі передачі, так і під час зберігання. Реалізація різнорівневих схем шифрування дозволяє гнучко налаштувати захист у залежності від важливості

та контексту даних, а також забезпечити відповідність стандартам безпеки та нормативним вимогам.

3.3.1 Шифрування під час передачі

У PostgreSQL для захищеного зв’язку реалізована підтримка TLS – сучасного криптографічного протоколу, що забезпечує конфіденційність, цілісність та автентичність переданої інформації [13].

PostgreSQL реалізує TLS через бібліотеку OpenSSL. При встановленні захищеного підключення клієнт і сервер погоджують криптографічні параметри (алгоритми, ключі, сертифікати) й ініціалізують шифрування, перш ніж передати будь-які автентифікаційні або SQL-дані [29].

Альтернатива для SQLite. SQLite не підтримує мережеву передачу даних, оскільки не має серверної частини. Тому TLS тут не застосовується. Якщо SQLite використовується у мережевому середовищі (наприклад, файл бази передається через WebDAV, SMB або інші шари), захист трафіку повинен здійснюватися на зовнішньому рівні — наприклад, через VPN або файлові системи з TLS/SSL.

3.3.2 Шифрування в стані спокою

У PostgreSQL нативна підтримка повного шифрування бази (TDE) поки що відсутня, однак наявні ефективні альтернативи у вигляді розширень pgcrypto та pgsodium, а також можливість реалізації шифрування на рівні операційної системи.

TDE – це технологія, яка забезпечує автоматичне шифрування всіх даних на рівні СУБД, без потреби змінювати логіку роботи із таблицями. Розробка TDE для PostgreSQL триває в офіційному репозиторії – очікується інтеграція у майбутніх версіях ядра (версія 17+). Є сторонні модифіковані збірки PostgreSQL

з TDE, наприклад EDB Advanced Server або PGPro Enterprise від Postgres Professional, однак вони комерційні та не є відкритими [13].

pgcrypto – це офіційне розширення PostgreSQL, яке забезпечує симетричне шифрування (AES, Blowfish), хешування (SHA1, SHA256, MD5), асиметричне шифрування (RSA, PGP).

pgsodium – це новіше розширення, яке використовує бібліотеку libsodium, відому своєю крипографічною стійкістю та простим API. Воно підтримує симетричне шифрування (crypto_secretbox), асиметричне (crypto_box), захищене зберігання ключів, цифрові підписи (crypto_sign), захист даних у оперативній пам’яті. Розширення підтримує генерацію ключів і керування ними через внутрішній механізм Key Management System (KMS). Перевага pgsodium це автоматична обробка ключів і сучасна крипографія (наприклад, XChaCha20, Poly1305, Ed25519), рекомендована для нових систем.

У SQLite шифрування в стані спокою не підтримується нативно, однак доступне через стороннє рішення SQLCipher. SQLCipher шифрує весь файл бази даних за допомогою AES-256 і PBKDF2 [28].

3.3.3 Шифрування на рівні колонок

Шифрування на рівні колонок є одним із найгнучкіших способів захисту конфіденційної інформації в базах даних. На відміну від повного шифрування бази (TDE), яке застосовується до всіх даних без винятку, колонкове шифрування дозволяє захищати лише обрані поля, що містять чутливу або персональну інформацію – наприклад, номери документів, банківські реквізити, паролі, медичні діагнози тощо.

Цей підхід дозволяє поєднувати високий рівень захисту з економією ресурсів і збереженням продуктивності. У PostgreSQL шифрування на рівні колонок реалізується за допомогою розширень pgcrypto або pgsodium, які вже

розглядалися в попередньому розділі [13]. Колонкове шифрування є доцільним, коли:

- лише частина даних є конфіденційною, а решта — загальнодоступною;
- потрібно зберегти можливість пошуку або агрегації над незашифрованими полями;
- необхідна гнучка політика доступу до окремих елементів запису (наприклад, відкрити ім’я, але приховати номер паспорта);
- важливо знизити навантаження на СУБД порівняно з повним шифруванням.

У SQLite шифрування на рівні окремих колонок безпосередньо не підтримується. Якщо використовується SQLCipher, то вся база шифрується як єдине ціле. Утім, можливо реалізувати шифрування значень вручну у застосунку – наприклад, за допомогою бібліотек CryptoKit (у Swift/macOS) чи OpenSSL, які шифрують дані перед вставкою в базу.

3.4 Захист від SQL-ін’єкцій та шкідливих запитів

Однією з найнебезпечніших уразливостей для СУБД залишаються SQL-ін’єкції, які виникають у результаті неналежної обробки користувацьких даних. Навіть одинична помилка в логіці запиту може призвести до серйозних наслідків – витоку даних, їхньої модифікації або видалення. Захист від таких загроз потребує впровадження конкретних технік розробки та аналізу запитів, які дозволяють виявити та нейтралізувати спроби впливу на структуру SQL-інструкцій.

3.4.1 Використання параметризованих запитів у клієнтських застосунках

У системах, що використовують СУБД PostgreSQL або SQLite, захист від SQL-ін’єкцій є однією з базових вимог до інформаційної безпеки. Найбільш ефективним і загальноприйнятим методом захисту є використання параметризованих запитів. У цьому підході вхідні значення користувача не вставляються безпосередньо в текст SQL-запиту, а передаються у вигляді окремих параметрів, які попередньо обробляються СУБД як дані, а не як код.

Наприклад, у PostgreSQL за допомогою бібліотеки psycopg2 параметризований запит виглядає так:

```
cur.execute("SELECT * FROM users WHERE username = %s", (user_input,))
```

Аналогічно в SQLite (через Python sqlite3):

```
cur.execute("SELECT * FROM users WHERE email = ?", (email,))
```

У наведених прикладах СУБД автоматично виконує екранування вхідних значень, що унеможливлює виконання шкідливих конструкцій, таких як `OR 1=1`, `DROP TABLE users` тощо.

Окрім мов Python, аналогічні механізми доступні у Java (через PreparedStatement), PHP (через PDO), C# (через SqlCommand із параметрами), JavaScript (через ORM Prisma або бібліотеки pg, knex), а також у багатьох сучасних вебфреймворках.

Ще одним важливим засобом протидії ін’єкціям є використання збережених процедур або функцій. У PostgreSQL можна створити функцію з жорстко заданою логікою, яка не дозволяє динамічно формувати запити на основі введених даних. Це особливо корисно при реалізації критичних функцій – наприклад, видалення записів, обробки транзакцій, доступу до обмежених таблиць.

Перевагою такого підходу є не лише захист від SQL-ін’єкцій, а й покращення продуктивності за рахунок кешування запитів на рівні СУБД. У PostgreSQL параметризовані запити можуть бути автоматично переведені в підготовлені (prepared statements), що зменшує навантаження при повторному виконанні однакових запитів зі змінними параметрами [13].

У випадку з SQLite рівень захисту цілком залежить від застосунку, який формує запити, оскільки сама СУБД не має окремих користувачів чи внутрішньої системи прав. Тому застосування параметризованих запитів тут є обов’язковою вимогою. Якщо ж розробник будує SQL-запити шляхом прямої вставки вхідних даних у рядки коду – ймовірність SQL-ін’єкції в такому застосунку є надзвичайно високою [30].

3.4.2 Аудит логів на предмет аномалій і ін’єкцій

Аудит логів дозволяє виявляти аномальну активність, спроби SQL-ін’єкцій та інші ознаки несанкціонованого доступу або використання системи. Журнали, які генерує СУБД, містять безцінну інформацію про взаємодію користувачів із системою, виконані запити, час операцій, помилки автентифікації та інші події, що можуть сигналізувати про наявність загроз.

У PostgreSQL аудит логів проводиться переважно на основі даних, що фіксуються при активованому logging_collector, а також через спеціальні розширення, зокрема pgaudit та pg_stat_statements. Один із найпоширеніших підходів – це аналіз лог-файлів на наявність шаблонів, характерних для атак SQL-ін’єкцій. Наприклад, запити з фрагментами типу `' OR '1'='1'`, спроби вставки `--` або `DROP TABLE`, багатократні запити до однієї й тієї ж таблиці з мінімальними змінами – усе це може свідчити про атаку або сканування вразливостей.

Розглянемо приклад типової шкідливої ін’єкції у журналі:

```
2025-05-30 15:32:01 user1@securedb LOG: statement: SELECT * FROM users WHERE username = 'admin' --' AND password = 'pass'
```

Аналізуючи такі записи, можна автоматизовано або вручну виявити запити, які відрізняються від стандартної поведінки користувачів. У PostgreSQL для цього використовуються як скрипти на базі grep, awk, sed, так і більш просунуті утиліти типу pgBadger, GoAccess, Splunk, ELK Stack [13]. Наприклад, командою:

```
grep --ignore-case 'or 1=1' postgresql.log
```

можна швидко перевірити лог на наявність потенційно небезпечного шаблону.

Окрім явних ознак ін’єкцій, аудит логів може виявити аномальну активність, пов’язану з:

- незвичним обсягом даних, які витягуються за один запит;
- запитами до незвичних таблиць або функцій;
- високою частотою запитів за короткий проміжок часу;
- помилками автентифікації, що повторюються для різних імен користувачів.

Для систем, що обслуговують велику кількість користувачів або працюють у розподіленому середовищі, доцільно реалізовувати системи моніторингу на основі поведінкового аналізу. Такі системи дозволяють виявити відхилення від типової активності конкретного користувача або ролі. Наприклад, якщо користувач, який зазвичай читає лише одну таблицю, раптом починає робити запити до кількох інших таблиць, змінювати записи або запускати агрегатні функції – це може бути ознакою компрометації облікового запису.

У PostgreSQL журналювання можна поєднувати з механізмом тригерів, які фіксують нестандартну активність у додаткових таблицях. Наприклад,

тригер, що реагує на масові `UPDATE` або `DELETE`, може створювати запис у таблиці аудиту зі збереженням часу, користувача та SQL-команди.

У випадку з SQLite, аудит логів не підтримується на рівні самої СУБД, оскільки вона не має журналювального сервера. Однак контроль можливо реалізувати на рівні клієнтського застосунку – наприклад, логуванням усіх SQL-запитів перед їх виконанням або записом критичних операцій у окремий log-файл. Для спрощення цього процесу можна використовувати middleware-бібліотеки або спеціальні ORM-обгортки, які автоматично логують звернення до бази даних [27].

У системах із високими вимогами до безпеки доцільно реалізовувати централізований аудит, який дозволяє агрегувати логи з кількох джерел (наприклад, серверів баз даних, вебсервера, застосунку) та проводити комплексний аналіз. Це може бути реалізовано за допомогою таких систем, як Graylog, ELK Stack (Elasticsearch + Logstash + Kibana), Wazuh або Splunk.

3.5 Захист резервних копій та логів

В умовах сучасних кіберзагроз резервні архіви можуть стати легким об’єктом атаки, якщо вони зберігаються без належного захисту, а лог-файли – джерелом витоку чутливої інформації через збереження незашифрованих запитів, помилок авторизації, або часткових фрагментів SQL-операцій. Тому важливо реалізувати комплексну політику, яка охоплює створення, шифрування, зберігання та безпечне видалення таких даних.

У контексті розробленої методики захисту резервні копії бази даних створюються регулярно з використанням як логічного (pg_dump), так і фізичного (pg_basebackup) підходів. Логічне резервування дозволяє створити файл-дамп структури та вмісту бази, придатний для відновлення за допомогою pg_restore, у той час як фізичне резервування копіює бінарний стан бази, що корисно для швидкого повного відновлення.

Однак наявність резервної копії без її захищеного зберігання не лише не підвищує рівень безпеки, а й створює нові вразливості. Тому важливим кроком є впровадження політики, яка регламентує частоту копіювання (наприклад, щоденне повне копіювання та щогодинне інкрементальне), місце зберігання (ізольовані сервери або зашифровані хмарні сховища), а також обмеження терміну зберігання (наприклад, 7 днів для швидкого доступу, 30 днів — в архіві).

Для захисту від несанкціонованого доступу копії мають зберігатися у зашифрованому вигляді. Це може бути реалізовано за допомогою утиліти GPG, яка забезпечує як симетричне, так і асиметричне шифрування. Наприклад, логічна копія бази, створена через pg_dump, може бути зашифрована командою gpg -c, яка запитує пароль і шифрує вміст файлу. У разі використання відкритих ключів можлива безпечна передача архіву між сторонами без розголошення пароля. Аналогічні можливості надає OpenSSL – через алгоритми типу AES-256-CBC, які підтримуються більшістю систем. Крім того, деякі утиліти для PostgreSQL, зокрема pgBackRest, інтегрують шифрування як частину стандартного процесу копіювання, що забезпечує автоматизацію й зменшує ймовірність помилки [13].

Шифрування архівів має супроводжуватися належним захистом ключів. Паролі та приватні ключі необхідно зберігати окремо від самих архівів, із використанням менеджерів паролів або інфраструктури відкритих ключів. У разі втрати ключа відновлення даних буде неможливим, тому його зберігання є таким самим критичним, як і саме резервне копіювання.

Ще однією важливою складовою захисту є контроль над лог-файлами. PostgreSQL генерує логи, які фіксують запити, помилки, дії користувачів і багато іншого. Без належного обмеження часу зберігання ці логи можуть накопичувати велику кількість даних, включаючи фрагменти конфіденційної інформації. Для уникнення таких ситуацій у PostgreSQL налаштовується автоматична ротація логів через параметри log_rotation_age (наприклад, один

день) та log_rotation_size (наприклад, 100 МБ). Це дозволяє створювати нові файли логів на регулярній основі, мінімізуючи ризик неконтрольованого накопичення інформації в одному місці.

3.6 Впровадження методики захисту конфіденційної інформації в PostgreSQL

Побудова безпечного середовища для роботи з базами даних є не лише технічним завданням, а й процесом, що потребує стратегічного планування. Захист інформації в СУБД не досягається одноразовим впровадженням певного набору інструментів чи налаштувань. Ефективність забезпечення безпеки прямо залежить від чіткої послідовності дій, а також від врахування особливостей конкретного середовища розгортання.

Запропонована нижче послідовність є універсальною для впровадження безпечної інсталяції PostgreSQL, однак її основні етапи можуть бути адаптовані й до інших СУБД. Вона побудована на практичному досвіді з врахуванням типових загроз, описаних у попередніх розділах.

Проєктування структури БД з урахуванням принципів безпеки. Ще на етапі моделювання бази даних слід визначити, які саме поля міститимуть конфіденційну інформацію, які користувачі матимуть до них доступ, та яка бізнес-логіка передбачає їх обробку. Важливо впровадити принцип найменших привілеїв.

Наприклад, для системи, що зберігає персональні дані клієнтів, дані можна умовно розділити на:

- Відкриті дані (наприклад, ID-клієнта, місто);
- Обмежені дані (наприклад, email, телефон);
- Конфіденційні дані (ПІБ, паспортні дані, платіжна інформація).

З допомогою інструменту ERD (Entity-Relationship Diagram) можна відобразити графічну схему відносин даних у системі, приклад реалізовано на рис.3.3:

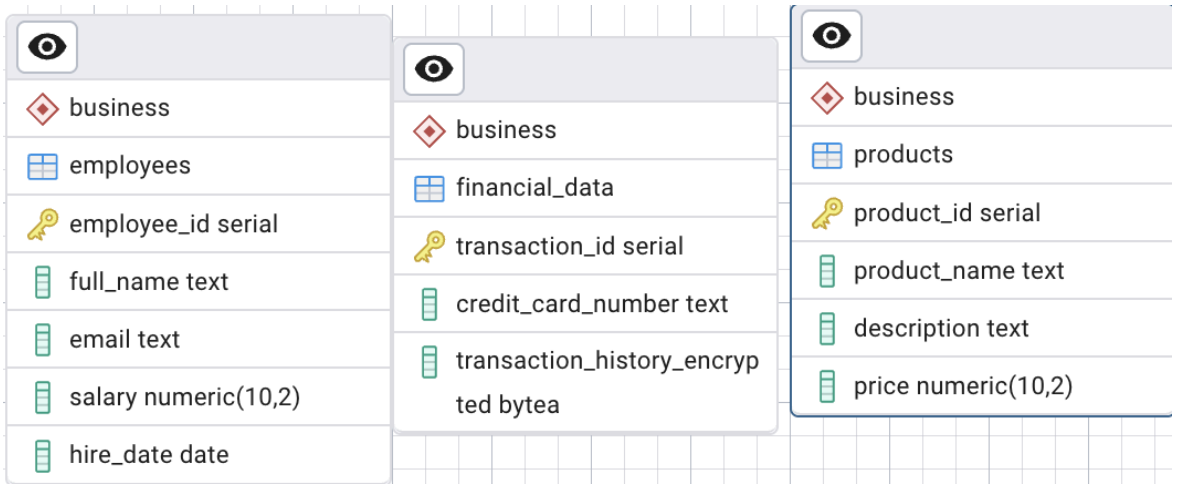

Рисунок 3.3 – приклад структури даних які можуть зберігатись в БД

Для кожної категорії визначаються ролі, які можуть здійснювати операції читання або модифікації. Наприклад:

Адміністратор бази даних. Повний доступ до всіх даних. Бізнес-логіка: адміністрування схеми БД, резервне копіювання, налаштування прав доступу, аудит логів.

HR-менеджер. Має доступ до персональних даних працівників. Бізнес-логіка: Перегляд персональних даних працівників, формування звітів по зарплаті та датах найму.

Фінансовий аналітик. Має обмежений доступ до фінансових даних. Бізнес-логіка: Аналіз фінансових потоків, формування статистичних і аналітичних звітів.

Працівники. Доступ: загальнодоступні дані та персональні дані (лише дані самого працівника). Бізнес-логіка: Додавання нових продуктів до каталогу, оновлення опису та цін, огляд персональних даних.

Розгортання СУБД із безпечними параметрами. Після встановлення СУБД необхідно одразу внести зміни до файлів конфігурації (postgresql.conf, pg_hba.conf):

- *обмежити доступ до сервера лише з дозволених IP-адрес та вимкнути автентифікацію за довільними методами на кшталт trust або ident.*

Спочатку знаходимо шлях до файлу pg_hba.conf, відкриваємо його в редакторі та вказуємо дозвіл лише з локальних IP-адрес, також реалізуєм механізм аутентифікації: в методі замінюєм значення TRUST на scram-sha-256 (з допомогою якого шифруємо облікові дані).

```
#TYPE DATABASE USER    ADDRESS        METHOD
host   all      all    127.0.0.1/32   scram-sha-256
host   all      all    ::1/128        scram-sha-256
```

- *активувати SSL-з'єднання (якщо застосовується).*

Активувати SSL-з'єднання можна під час створення серверу у pgAdmin4 рис. 3.4 або у файлі postgresql.conf встановлюємо значення – `ssl = on`.

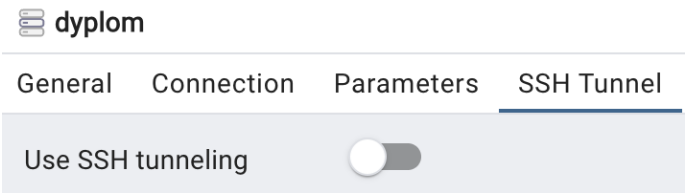

Рисунок 3.4 – налаштування SSL-з'єднання у pgAdmin4

- *встановити журналювання (logging_collector = on, log_connections = on, log_disconnections = on).*

У файлі postgresql.conf можемо встановити наступні значення:

```
logging_collector = on
log_directory = 'log'
log_filename = 'postgresql-%Y-%m-%d.log'
log_connections = on
log_disconnections = on
```

```
log_hostname = on
log_line_prefix = '%m [%p] %u@%d '
```

Управління обліковими записами і правами доступу. Після встановлення PostgreSQL за замовчуванням створюється суперкористувач postgres. Необхідно створити окремих користувачів для адміністративних завдань, читання та запису, та обмежити права postgres, відключивши можливість його використання через мережу. Кожному користувачу слід встановити надійний пароль і явно задати ролі з відповідними дозволами. Уникайте надання прав на рівні GRANT ALL.

Автоматично створений суперкористувач postgres може відігравати роль адміністратора БД, тому нам необхідно лише реалізувати обмеження в правах – відключивши можливість його використання через мережу (у pg_hba.conf):

```
host    all     postgres     0.0.0.0/0      reject
```

HR-менеджер:

```
CREATE ROLE hr_manager WITH LOGIN PASSWORD
'Strong_HR_Pass';
GRANT CONNECT ON DATABASE postgres TO hr_manager;
GRANT USAGE ON SCHEMA public TO hr_manager;
GRANT SELECT(full_name, email, salary, hire_date)
ON TABLE employees TO hr_manager;
```

Зробимо перевірку на коректність доступу ролі до даних (рис. 3.5):

```
maksimvezdeckij@MacBook-Pro-Maksim ~ % psql -h localhost -U hr_manager -d postgres

Password for user hr_manager:
psql (14.18 (Homebrew))
Type "help" for help.

postgres=> SELECT * FROM employees;
ERROR:  permission denied for table employees
postgres=> SELECT email FROM employees;
            email
-----------------------------
 ivan.petrenko@example.com
 maria.sydorenko@example.com
 olexii.kovalenko@example.com
(3 rows)

postgres=> SELECT credit_card_number FROM financial_data;
ERROR:  permission denied for table financial_data
```

Рисунок 3.5 – спроба доступу даних від імені hr_manager

Фінансовий аналітик, створюємо таким же чином роль, надаємо право доступу до історії транзакцій, а до інформації стосовно кредитних карток необхідно обмежити доступ (показувати лише останні 4 цифри) застосовуючи маскування даних наступним чином:

```
CREATE VIEW masked_credit_cards AS
SELECT transaction_id, RIGHT(credit_card_number, 4)
AS last_four_digits
FROM financial_data;
GRANT SELECT ON masked_credit_cards TO
finance_analyst;
```

Перевірку реалізації можна спостерігати на рис. 3.6:

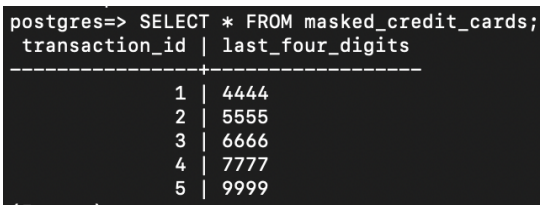

Рисунок 3.6 – перевірка доступу до замаскованих даних кредитних карток

Для працівників спочатку створено групу ролей employee_group, до якої їх віднесемо:

```
CREATE ROLE employee_group;
```

Створюємо користувачів для кожного працівника, та додаємо всіх до раніше створеної групи ролей:

```
GRANT employee_group TO "Ivan_Petrenko";
GRANT employee_group TO "Maria_Sydorenko";
GRANT employee_group TO "Olexii_Kovalenko";
```

Створюємо політику по якій кожен працівник зможе бачити лише особисті дані:

```
CREATE POLICY employee_self_select
```

```
ON employees
FOR SELECT
USING (current_user = full_name);
```

Вмикаємо RLS для таблиці employee, та спостерігаємо на рис. 3.7 спробу доступу до особистих даних користувача Ivan_Petrenko:

```
postgres=> SELECT * FROM employees;
 employee_id |   full_name   |          email          |  salary  | hire_date
-------------+---------------+-------------------------+----------+------------
           1 | Ivan_Petrenko | ivan.petrenko@example.com | 25000.00 | 2020-05-15
```

Рисунок 3.7 – спроба доступу до особистих даних працівника

Реалізація шифрування та маскування даних. Поля, які зберігають персональні дані, номери карток, IP-адреси тощо, повинні бути або зашифровані (наприклад, за допомогою pgcrypto), або доступні через динамічне маскування. Це дає змогу унеможливити витік навіть у разі компрометації облікового запису з обмеженими правами.

Шифруємо чутливі дані з допомогою pgcrypto:

```
UPDATE financial_data
SET transaction_history_encrypted =
pgp_sym_encrypt(transaction_history::text,
'secret_passphrase');
```

Та обов'язково видаляємо вхідні колонки з незашифрованими даними. На рис. 3.8 можна спостерігати результат шифрування:

transaction_history_encrypted bytea
[binary data]
[binary data]
[binary data]
[binary data]
[binary data]

Рисунок 3.8 – зашифровані фінансові дані в pgAdmin4

Налаштування системи моніторингу, логування та аудиту. Після встановлення базової конфігурації необхідно реалізувати логування запитів (log_statement = 'mod' або all) та встановити розширення pg_audit для глибшого аудиту. Система повинна записувати всі зміни до даних, спроби доступу до обмежених об’єктів та помилки авторизації.

Налаштовуємо системи моніторингу, логування та аудиту в postgresql.conf. pg_audit вже був встановлений в системі, наявність розширення можна перевірити запитом – `SELECT * FROM pg_extension WHERE extname = 'pgaudit';`.

Тестування безпеки: penetration-тести і аудит ролей. Після налаштування бази необхідно виконати моделювання атак (наприклад, SQL-ін’єкцій через sqlmap, перебору паролів через hydra, підвищення привілеїв) для перевірки ефективності захисту. Також варто переглянути всі існуючі ролі в системі, переконатись у відсутності зайвих дозволів і логічних вразливостей.

Перед початком атак потрібно впевнитися, що структура ролей і дозволів відповідає принципу найменших привілеїв. Переглянувши список ролей у базі на рис. 3.9, можемо підтвердити, що права ролей відповідають PoLP, також перевіримо детальні права кожної ролі.

```
postgres=# \du
                                   List of roles
    Role name     |                         Attributes                         |    Member of
------------------+------------------------------------------------------------+-----------------
 Ivan_Petrenko    |                                                            | {employee_group}
 Maria_Sydorenko  |                                                            | {employee_group}
 Olexii_Kovalenko |                                                            | {employee_group}
 employee_group   | Cannot login                                               | {}
 finance_analyst  |                                                            | {}
 hr_manager       |                                                            | {}
 maksimvezdeckij  | Superuser, Create role, Create DB, Replication, Bypass RLS | {}
```

Рисунок 3.9 – перелік усіх ролей, їхні атрибути і в які групи вони входять

Переконаймося, що RLS-політики дійсно захищають дані на рис 3.10.:

```
 schemaname | tablename |      policyname      | permissive |  roles   |  cmd   |           qual            | with_check
------------+-----------+----------------------+------------+----------+--------+---------------------------+------------
 public     | employees | employee_self_select | PERMISSIVE | {public} | SELECT | (CURRENT_USER = full_name) |
```

Рисунок 3.10 – перегляд RLS-політик для таблиці employees

Виконавши моделювання шкідливих запитів вручну від лиця фінансового аналітика, було здійснено спробу отримання доступу до конфіденційних даних в таблиці employees (до якої, у цієї ролі, доступ не наданий), методом вставки небезпечних шаблонів, наприклад:

```
' OR '1'='1',
' OR EXISTS(SELECT 1) –,
'; DROP TABLE users; –,
' OR EXISTS(SELECT * FROM information_schema.tables)--,
SELECT 'salary' FROM information_schema.columns WHERE table_name='employees';.
```

Результатом спроб доступу є помилка доступу та виведення пустих стовпців (рис. 3.11), що свідчить про успішну реалізацію захисту від SQL-ін'єкцій.

```
[maksimvezdeckij@MacBook-Pro-Maksim ~ % python3 query_wrapper.py "' OR ''='"
 Error: permission denied for table financial_data

[maksimvezdeckij@MacBook-Pro-Maksim ~ % python3 query_wrapper.py "' OR 'x'='x"
 Error: permission denied for table financial_data

[maksimvezdeckij@MacBook-Pro-Maksim ~ % python3 query_wrapper.py "' OR EXISTS(SELECT 1) --"
 Error: permission denied for table financial_data

[maksimvezdeckij@MacBook-Pro-Maksim ~ % python3 query_wrapper.py "'; DROP TABLE users; --"
 Error: permission denied for table financial_data
```

```
 ?column?
----------
(0 rows)
```

Рисунок 3.11 – Результат моделювання SQL-ін'єкцій

Також було реалізовано спробу використання команд, які вимагають прав суперкористувача: `CREATE DATABASE test_db;`, у відповідь було отримано: `ERROR: permission denied to create database.`

Резервне копіювання і перевірка процедури відновлення. Після того, як система налаштована і захищена, необхідно налаштувати регулярне резервне копіювання з подальшим тестуванням можливості повноцінного відновлення. Цей етап включає створення скриптів резервного копіювання (pg_dump, pg_basebackup), їх шифрування та автоматизований контроль.

Для створення резервних копій був обраний логічний підхід pg_dump на основі якого було вручну здійснено перше резервне копіювання. Наступним була реалізація налаштування системи автоматичного запуску резервного копіювання через cron, яка буде оновлювати резервні копії щодня о 3 ночі.

Для забезпечення конфіденційності та захисту резервних копій від несанкціонованого доступу було впроваджено їх шифрування за допомогою інструменту openssl з використанням алгоритму AES-256-CBC. Для перевірки коректності резервного копіювання була здійснена процедура відновлення бази даних з зашифрованого архіву. Попередньо резервна копія розшифровувалась за допомогою openssl, після чого відновлювалась у тестове середовище PostgreSQL за допомогою утиліти pg_restore. Результати відновлення підтвердили цілісність і працездатність резервних копій.

3.7 Оцінка ефективності розробленої методики

Оцінювання базується на результатах експериментальних досліджень, навантажувальних тестів і реалізованих сценаріїв проникнення, а також на досвіді адміністративного впровадження в середовищі PostgreSQL.

Під час проведення експериментального тестування було окремо досліджено кожен з цих компонентів, а також їхню взаємодію. Зокрема, налаштування RLS-політик виконувалося на таблицях, що містять персональні дані користувачів та фінансову інформацію. Метою було переконатися, що за наявності SQL-запитів, які звертаються до кількох таблиць одночасно, жоден користувач без відповідного рівня доступу не зможе отримати вихідну

інформацію. У результаті тестування запитів, які намагалися обійти фільтрацію, визначено, що система коректно обмежує доступ до тих рядків, які не відповідають ролям і параметрам поточного користувача. Це було підтверджено тим, що в жодному з випробуваних сценаріїв імітація запитів «Ivan_Petrenko» не дозволяє переглядати дані, крім особистих, інших працівників, а роль «finance_analyst» могла бачити лише ті поля, на які дійсно мала право. Таким чином, реалізація RLS-модулю виявилася достатньо стійкою навіть до складних обхідних запитів, що свідчить про високу надійність обмежувальних політик.

Ключовий компонент методики – шифрування даних – оцінювався через серію експериментів, що включали вимірювання часу виконання операцій запису та читання. Зокрема, було підтверджено, що вставка нового запису з шифруванням декількох конфіденційних полів збільшує тривалість транзакції приблизно на 10 % у порівнянні зі вставкою без шифрування. Водночас вибірка із зашифрованих колонок із застосуванням функцій розшифрування додавала близько 15 % до часу виконання звичайного SELECT-запиту.

Наступним ступенем захисту стало маскування тих полів, які можуть відображатися в загальних звітах або запитах фінансового аналітика. Під час реалізації використано методи приховування частини даних у полях, які містять номери банківських карток. В результаті ролі аналітика при запиті було відображено лише останні чотири цифри карток. Експериментально було доведено, що запити, які звертаються до заздалегідь підготовлених маскованих представлень, повертають синтетичні дані, що не дозволяють відновити оригінал. При цьому інструментарій обробки маскування не перевантажував систему в момент виконання регулярного звітування.

З точки зору адміністрування, методика виявилася відносно нескладною в інтеграції: всі необхідні модулі описані в окремих SQL-скриптах, які можна легко застосувати до типової структури схеми даних. Управління ключами шифрування відбувається через змінні середовища та конфігураційні файли

PostgreSQL, що дозволяє централізовано змінювати шифрувальний пароль без необхідності масових перезаписів бази.

Висновок за розділом 3

У третьому розділі було здійснено практичну реалізацію запропонованої методики захисту конфіденційної інформації в СУБД на прикладі PostgreSQL. Запропоновані механізми забезпечення безпеки охопили ключові аспекти інформаційного захисту: автентифікацію, авторизацію, шифрування, моніторинг активності, захист від SQL-ін’єкцій, а також безпечне зберігання резервних копій і логів. Особливу увагу приділено практичним аспектам захисту облікових даних, включаючи їх шифрування та аудит спроб автентифікації.

Можна зробити висновок, що розроблена методика є ефективною у трьох основних аспектах. Поперше, вона надійно захищає чутливу інформацію від несанкціонованого доступу за рахунок багаторівневих бар’єрів, що включають RLS, шифрування та маскування, а також механізми пільгового доступу через псевдонімізацію. Подруге, реалізація не створює неприйнятних затримок у роботі СУБД: додаткове навантаження залишається в допустимих межах навіть при високих обсягах операцій, що для сучасних апаратних ресурсів практично не відчувається. І, потретє, впровадження методики може бути здійснено без значних додаткових інвестицій у навчання адміністраторів, оскільки всі компоненти базуються на стандартному функціоналі PostgreSQL і не вимагають сторонніх плагінів чи складних налаштувань. Разом це робить розроблену методику доцільною та практично прийнятною для використання в організаціях із різним рівнем ресурсів, де захист даних є пріоритетом.

ВИСНОВКИ

У межах даної роботи було досліджено одну з найбільш актуальних проблем сучасної кібербезпеки – захист конфіденційної інформації в базах даних. У результаті опрацювання теми сформовано цілісне уявлення про сучасний стан безпеки СУБД, ключові виклики та ефективні підходи до їх подолання. Дослідження має не лише теоретичну цінність, а й практичну спрямованість, пропонуючи конкретні рішення для підвищення безпеки баз даних у реальному середовищі.

Захист конфіденційних даних є критично важливим завданням для будь-якої інформаційної системи, яка працює з персональною, фінансовою чи службовою інформацією. У роботі показано, що загрози безпеці БД не лише різноманітні, але й постійно еволюціонують. Це зумовлює необхідність застосування багаторівневого захисту: від організаційних заходів до технічних рішень на рівні програмного забезпечення, мережевої інфраструктури та адміністрування.

Розроблена методика захисту конфіденційної інформації базується на поєднанні декількох ключових компонентів: шифрування даних, автентифікації та авторизації користувачів, обмеження доступу, логування, аудиту, резервування та моніторингу активності. Застосування цих механізмів у межах обраної СУБД дозволило створити стійку до поширених атак модель з високим рівнем контролю доступу і захисту інформації. Особливо важливим є те, що реалізація була адаптована під сучасні вимоги до безпеки й відповідає принципам конфіденційності, цілісності та доступності.

Отримані результати підтверджують доцільність впровадження комплексного підходу до захисту БД. В умовах зростаючої кількості інцидентів у сфері кібербезпеки, поєднання криптографічних засобів, політик доступу,

сучасних систем моніторингу та процедур резервування є одним із найефективніших способів зменшити ризики втрати або компрометації даних.

Таким чином, впровадження системної методики захисту дозволяє значно підвищити рівень інформаційної безпеки баз даних. Представлений підхід може бути масштабований, адаптований до різних середовищ і використаний як основа для подальшого вдосконалення інструментів захисту в рамках інформаційних систем.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Verizon. 2025 Data Breach Investigations Report (DBIR) [Електронний ресурс]. Режим доступу до ресурсу: https://www.verizon.com/business/resources/reports/dbir

2. ENISA (2024). ENISA Threat Landscape 2024. Європейське агентство з кібербезпеки [Електронний ресурс]. Режим доступу до ресурсу: https://www.enisa.europa.eu/publications/enisa-threat-landscape-2024

3. National Institute of Standards and Technology (NIST) (2024). CVE-2024-20917 Detail. Національна база даних вразливостей [Електронний ресурс]. Режим доступу до ресурсу: https://nvd.nist.gov/vuln/detail/CVE-2024-20917

4. MITRE (2024). CVE-2024-23480. CVE Details [Електронний ресурс]. Режим доступу до ресурсу: https://www.cvedetails.com/cve/CVE-2024-23480

5. Oracle (2024). Oracle Critical Patch Update Advisory – January 2024 [Електронний ресурс]. Oracle Corporation. Режим доступу до ресурсу: https://www.oracle.com/security-alerts/cpujan2024.html

6. National Institute of Standards and Technology (NIST) (2024). CVE-2024-36891 Detail: Windows Kernel Elevation of Privilege Vulnerability. National Vulnerability Database. Режим доступу до ресурсу: https://nvd.nist.gov/vuln/detail/CVE-2024-36891

7. Snyk Vulnerability Database (2024). Sequelize SQL Injection Vulnerability (CVE-2024-24752) [Електронний ресурс]. Режим доступу до ресурсу: https://snyk.io/vuln/npm:sequelize

8. Oracle. MySQL Audit Plugin Security Bulletin – 2024 [Електронний ресурс]. Oracle Corporation. Режим доступу до ресурсу: https://dev.mysql.com/doc/relnotes/mysql/8.0/en/news-8-0-36.html

9. Mandiant. M-Trends 2025 [Електронний ресурс]. Mandiant, Google Cloud. Режим доступу до ресурсу: https://cloud.google.com/security/resources/m-trends

10. CoreWin. 13 найкращих практик для безпеки баз даних [Електронний ресурс] CoreWin – 2024. Режим доступу до ресурсу: https://corewin.ua/blog/what-is-database-security/?utm_source.com

11. GDPR. Regulation (EU) 2016/679 of the European Parliament and of the Council [Електронний ресурс]. Режим доступу до ресурсу: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A32016R0679

12. IBM X-Force Threat Intelligence Index 2024. IBM Corporation, 2024 [Електронний ресурс]. Режим доступу до ресурсу: https://www.ibm.com/reports/threat-intelligence

13. PostgreSQL Documentation [Електронний ресурс]. Режим доступу до ресурсу: https://www.postgresql.org/docs/current/index.html

14. OWASP Foundation. Cheat Sheet Series. [Електронний ресурс]. Режим доступу до ресурсу: https://cheatsheetseries.owasp.org/index.html

15. AWS: Amazon RDS Documentation [Електронний ресурс]. Режим доступу до ресурсу: https://docs.aws.amazon.com/rds/

16. Google Cloud: Access Control in Cloud SQL [Електронний ресурс]. Режим доступу до ресурсу: https://cloud.google.com/sql/docs/mysql/iam-authentication

17. National Institute of Standards and Technology (NIST). FIPS 140-2 Standard for Cryptographic Modules. Режим доступу до ресурсу: https://csrc.nist.gov/publications/detail/fips/140/2/final

18. Thales 2023 Cloud Security Study. Thales Group, 2023 [Електронний ресурс]. Режим доступу до ресурсу: https://cpl.thalesgroup.com/resources/cloud-security/2023/cloud-security-research-report

19. AWS Backup: Features Overview. Amazon Web Services, 2025 [Електронний ресурс]. Режим доступу до ресурсу: https://aws.amazon.com/backup/features/

20. International Organization for Standardization (2022). Information security, cybersecurity and privacy protection – Information security management systems. (ISO/IEC 27001:2022). https://www.iso.org/standard/27001

21. Санченко В. І. Система виявлення та попередження вторгнень у комп'ютерні мережі: кваліфікаційна робота на здобуття освітнього ступеня магістр за спеціальністю „125 – кібербезпека" /Санченко В. І. – Київ: КАІ, 2024. Режим доступу: https://er.nau.edu.ua/handle/NAU/66065

22. Splunk Docs – Database Monitoring. Splunk, 2025 [Електронний ресурс]. Режим доступу до ресурсу: https://docs.splunk.com/Documentation/DBX

23. Microsoft Learn. Microsoft Defender for Azure SQL Databases [Електронний ресурс]. Режим доступу до ресурсу: https://learn.microsoft.com/en-us/azure/defender-for-cloud/defender-for-sql-introduction

24. Strac (2025). Top 7 Data Loss Prevention Solutions: Comparison and DLP Evaluation Checklist [Електронний ресурс]. Режим доступу до ресурсу: https://www.strac.io/blog/top-7-data-loss-prevention-solutions-comparison-and-dlp-evaluation-checklist

25. К. Шуліка, Д. Балагура, А. Смірнов, Д. Непокритов, А. Литвин (2024). Метод використання сучасних систем захисту кінцевих точок (EDR) для убезпечення від комплексних атак. Innovative technologies and scientific solutions for industries. Режим доступу: https://www.researchgate.net/publication/382478670_A_method_of_using_modern_endpoint_detection_and_response_EDR_systems_to_protect_against_complex_attacks

26. IBM Corporation (2025). Security Guardium [Електронний ресурс]. Режим доступу до ресурсу: https://www.ibm.com/products/guardium

27. SQLite (2025). SQLite Documentation [Електронний ресурс]. Режим доступу до ресурсу: https://sqlite.org/docs.html

28. GitHub (2025). SQLCipher Repository [Електронний ресурс]. Режим доступу до ресурсу: https://github.com/sqlcipher/sqlcipher

29. Баргилевич О.А., Найман Г.Г. (2022). Забезпечення безпеки системи управління базами даних PostgreSQL. Сучасний захист інформації. Режим доступу: https://journals.dut.edu.ua/index.php/dataprotect/article/view/2646/2544

30. MoldStud, 2025. Safeguarding Data Privacy in SQLite Applications [Електронний ресурс]. Режим доступу до ресурсу: https://moldstud.com/articles/p-safeguarding-data-privacy-in-sqlite-applications