ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра коип'ютерної інженерії

Пояснювальна записка

до кваліфікаційної роботи на ступінь вищої освіти магістр

на тему: «**УДОСКОНАЛЕННЯ СИСТЕМИ МОНІТОРИНГУ ТРАНЗАКЦІЙ ДЛЯ ЗМЕНШЕННЯ НАВАНТАЖЕННЯ НА СУБД ORACLE»**

Виконав: студент 6 курсу, групи КСДМ–61 спеціальності

123 _Комп'ютерна інженерія____
(шифр і назва спеціальності)

_____Гедірлі М.М._____
(прізвище та ініціали)

Керівник __Торошанко Я.І._____
(прізвище та ініціали)

Рецензент_____
(прізвище та ініціали)

Київ – 2022

ВСТУП

Для сучасного бізнесу в індустрії інформаційно-комунікаційних технологій загальноприйнятою практикою започаткування і ведення взаємовідносин із своїми клієнтами є укладання угоди про рівень наданого сервісу – так званий Service Level Agreement (SLA).

Оскільки клієнти делегують задачі провайдеру IT-послуг, виникає потреба у контролі над якістю обслуговування. Замовник звертає увагу на певні метрики та індикатори, зокрема на такі як:

– час роботи окремих сервісів;

– час реагування на інциденти;

– час вирішення інцидентів.

Такий підхід дає можливість оцінити об’єм та відповідність якості наданої послуги.

Довгий час ідентифікації проблеми може призвести до посилення наслідків інциденту [1, с. 40]. Це у свою чергу призводить до збільшення часу вирішення інциденту для сторони, що надає послугу та збільшення часу простою сервісу для сторони, що цією послугою користується. Отже, невідповідність перелічених показників призводить до так званої «невідповідності SLA». Це в свою чергу веде до фінансових втрат і зниження довіри з боку клієнтів до даного банківського продукту, тому важливо усвідомити актуальність заходів протидії і розробити комплексний підхід до вирішення проблеми для зменшення ризиків.

Для своєчасного реагування на інциденти та мінімізації часу простою сервісів існує роль Service Desk.

Service Desk – це служба, яка повністю відповідає перед клієнтами або користувачами за надання послуг, погоджених з ними, є центром прийому всіх скарг та пропозицій, контролює поточний стан сервісів і має повноваження призначати іншим спеціалістам наряди на усунення інцидентів.

Service Desk відіграє вирішальну роль у наданні послуг надаючи [1, с. 100]:

– більш досконале обслуговування та задоволеність клієнтів;

– розширену доступність через єдину точку для зв’язку, спілкування та інформування;

– кращу якість та швидшу обробку запитів клієнта або запитів користувачів;

– досконалішу та більш злагоджену роботу та взаємодію в команді;

– посилену спрямованість та активний підхід до надання послуг;

– зменшення негативного впливу на бізнес;

– покращене використання ресурсів відділу підтримки та більша продуктивність бізнес-персоналу.

Продуктивність інженера Service Desk залежить від його аналітичних здібностей, загальної злагодженості процесів та роботи команди та від інструментів, які інженер має у своєму розпорядженні.

Розглядаючи процесінговий центр (ПЦ) як, в основному, постачальника послуг пов’язаних з обробкою даних платіжних карток стає зрозумілим що головною системою що забезпечує сервісом клієнтів є авторизаційна система.

Авторизаційна система виконує роботу з обробки транзакцій, перевіряє чи можна провести транзакцію беручи до уваги поточний залишок на рахунку, карткові та інші ліміти, можливі заборони на деякий тип операцій тощо.

Тому у контексті забезпечення безвідмовності надання сервісу процесінговим центром слід розглядати систему моніторингу авторизаційної системи як один з ключових інструментів інженера Service Desk.

Актуальність теми полягає у необхідності розробки рішення для оптимізації існуючої системи моніторингу транзакцій.

Отже *об’єктом дослідження* є процес взаємодії системи моніторингу транзакцій з об’єктно-реляційною системою управління базами даних Oracle Database.

Предметом дослідження є метод отримання транзакцій у системі моніторингу транзакцій з журнальної таблиці.

Метою дослідження є підвищення відмовостійкості бази даних Українського процесінгового центру, розробити комплекс рішень по оптимізації системи моніторингу транзакцій.

1 ОГЛЯД АРХІТЕКТУРИ СИСТЕМИ МОНІТОРИНГУ ТРАНЗАКЦІЙ

1.1 Авторизаційна система «Authentic»

Authentic - це інтелектуальна платформа для обробки транзакцій, постачальником якої є американська компанія «NCR». Ця платформа пристосована для сьогоденних реалій швидко мінливого платіжного бізнесу. Незалежно від того, чи є ви еквайєром, емітентом, центральним комутатором, банком або банком, що обслуговує транзакції з картами, альтернативні платежі або платежі в режимі реального часу, Authentic пропонує гнучкість, необхідну для задоволення сучасних потреб і майбутніх нововведень.

Authentic пройшов сертифікацію PA-DSS (Payment Application Data Security Standard) і має продуктивність до 10 000 транзакцій в секунду. Він призначений для того, щоб тримати все під контролем у своєму платіжному середовищі і забезпечує функціонально багатство, безпечність, відмовостійкість і масштабовану продуктивність. Authentic може приймати транзакції будь-якого типу з будь-якого пристрою, джерела або системи, авторизовувати і аутентифікувати їх і направляти в будь-яке місце призначення. Authentic може бути розгорнута в хмарних середовищах, а також в традиційному середовищі центру обробки даних. [1]

Особливості:

- мультиінституційна, багатовалютна, багатомовна та багатоканальна підтримка
- підтримка банкоматів та POS для всіх основних пристроїв
- підтримка схем карток PCI PA-DSS
- відповідність стандартам EMV, включаючи безконтактні та NFC прозорі та повторні
- налаштування бізнес-логіки
- правила управління ризиками в реальному часі
- легка конфігурація для нових мережевих і хост-інтерфейсів

- масштабованість - від малих систем шлюзів до глобальних мереж
- 99,9999% стійкість
- понад 10 000 операцій в секунду

1.1.1 Компоненти сервісу Authentic

Сервіс Authentic складається з наступних основних компонентів:

а) кластерні ноди Authentic:
   1) основні:
      - ax1-prod
      - bx1-prod
   2) резервні
      - ax2-prod
      - bx2-prod

б) сервісні ноди Authentic:
   1) основна нода ax1serv-prod
   2) резервна нода bx1serv-prod

в) Authentic DB

г) HSM(s)

д) Authentic Desktop

Кластерні ноди (пункт а) виконують обробку (процесінг) транзакцій. Ноди дублюють одна одну для забезпечення високої відмовостійкості. Наочна схема системи зображена на рисунку 1.1.

Сервісні ноди (пункт б) надають як зовнішнім по відношенню до сервісу Authentic системам, так і процесам Authentic-серверів різні сервіси (наприклад, сервіс шифрування / дешифрування карткових даних з використанням Encrypt/Decrypt API).

Authentic-сервера є набором Java процесів, які використовують відповідну конфігурацію для виконання всіх внутрішніх операцій сервісу Authentic. Основним компонентом сервера Authentic є ПЗ Authentic.

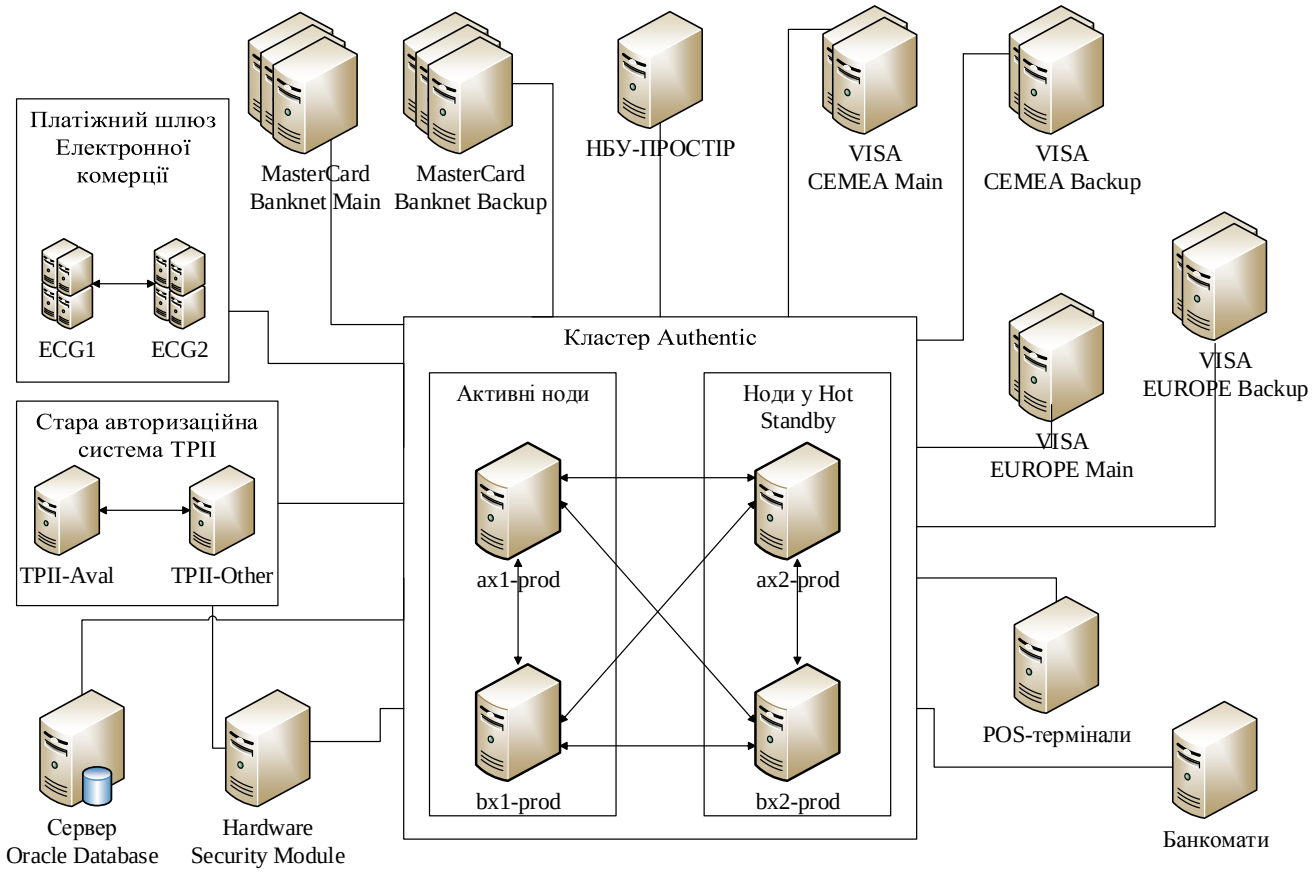

Рисунок 1.1 – Схема роботи нод Authentic

Authentic DB являє собою БД (схему) яка містить в собі: конфігураційну інформацію, дані програми Authentic, транзакційні дані, аудитні і іншу інформацію сервісу Authentic.

Authentic Desktop представляє собою GUI-утіліту яка використовується для:

– конфігурування інтерфейсів, процесів, точок доступу и т.д.;
– управління користувачами Authentic та їх рабочим оточенням;
– моніторингу роботи системи Authentic, подій, команд, процесів та ін.
– управління роботою процесів системи Authentic;
– перегляду статистичної інформації;

Система Authentic є продуктом компанії NCR Limited (в минулому Alaric Systems Ltd). Програмні засоби системи функціонують на серверах, що працюють

під управлінням операційної системи Red Hat Enterprise Linux (RHEL) Server. До додаткових елементів і засобів, необхідних для функціонування системи Authentic, відносяться СУБД Oracle Database, віртуальна машина Java, брокер повідомлень Apache ActiveMQ, а також криптографічний модуль HSM.

1.1.2 Апаратна частина сервісу Authentic

1.1.3 Огляд сутностей в Authentic

Перед втручанням у систему, необхідно ознайомитись з теоретичною інформацією. Тільки зрозумівши як влаштована система зсередини можна створити до неї якісний додаток.

Нода (Node) це один вузол кластеру процесінгової системи, на якому запущено екземпляр Authentic. На кожній активній ноді запущені необхідні процеси для обробки транзакцій, формування клірингових файлів, роботою з пулом підключень до баз даних, HSM тощо. Всі ноди повністю дублюють одна одну за функціональністю та пропускною спроможністю. Зазвичай одночасно працюють дві ноди що розташовані в одному з двох центрів обробки даних. Гнучкість системи дозволяє сконфігурувати будь-яку комбінацію нод – навіть одна нода здатна витримати пікові навантаження певний проміжок часу.

Процес (Process) безпосередньо виконує певну закладену роботу з обробки даних. Він може приймати та віддавати повідомлення по мережі через IAP, або комунікувати з іншими процесами. Один і той самий процес може бути запущений на декількох нодах. Один запущений процес, в залежності від його ролі, може або не мати жодного IAP, або мати їх довільну кількість

IAP (Interchange Access Point) – віртуальна точка обміну. Сама по собі існувати не може, бо має бути відкрита під якимось процесом. IAP може працювати як тільки з вхідним або вихідним трафіком, так із змішаним. Оскільки ця сутність віртуальна, вона інкапсулює в собі реальні сеанси з’єднання, ініційовані за різними протоколами.

Фізичне з’єднання (Physical connection) – реальний сеанс підключення до якогось віддаленого вузла на кшталт бази даних або віддаленого маршрутизатору МПС.

ISO 8583 - стандарт ISO, що описує процес передачі і формат фінансових повідомлень (транзакцій) систем, що обробляють дані банківських платіжних карт.

Транзакція даних платіжних карт (наприклад, для запиту авторизації держателя картки) починається з пристрою, що її ініціював, наприклад с POS-терміналу, проходить через ряд мереж і закінчується на системі, яка випустила карту [5] (рисунок 1.2).

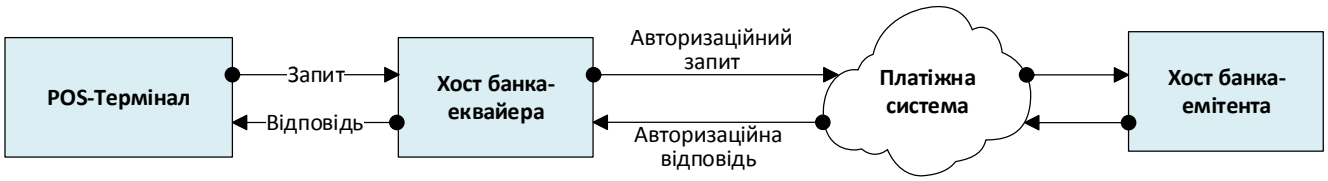

Рисунок 1.2 – Транзакція з використанням платіжної картки

Транзакційні дані включають в себе:

– інформацію про карту (наприклад, номер рахунку, англ. Account number)

– інформацію про термінал (наприклад, номер торговця, англ. Merchant)

– власне фінансова інформація (наприклад, ціна, кількість)

Система, яка випустила карту, авторизує транзакцію або відхиляє її і генерує відповідне повідомлення, яке повертається на термінал.

Повідомлення ISO 8583 складається з наступних частин:

– Message Type Indicator (MTI) - індикатор типу повідомлення;

– одна або кілька бітових карт, що вказують, які елементи даних присутні в повідомленні;

– елементи даних, поля повідомлення.

Зазвичай у повідомленні міститься близько 200 полів з різноманітною інформацією.

1.2 Система управління базами даних Oracle Database

Вибір системи управління базами даних є очевидним, адже Oracle Database найкраще зарекомендувала себе у фінансовій та банківській галузі. До того ж Authentic використовує саме базу даних Oracle, і лог транзакцій також зберігається у БД Oracle.

Oracle Database (часто просто Oracle) — об'єктно-реляційна система керування базами даних від Oracle Corporation.

1.2.1 Мова SQL

SQL (англ. Structured query language — мова структурованих запитів) — декларативна мова програмування для взаємодії користувача з базами даних, що застосовується для формування запитів, оновлення і керування реляційними БД, створення схеми бази даних та її модифікації, системи контролю за доступом до бази даних. Сама по собі SQL не є ані системою управління базами даних, ані окремим програмним продуктом. На відміну від дійсних мов програмування (C або Pascal), SQL може формувати інтерактивні запити або, бувши вбудованою в прикладні програми, виступати як інструкції для керування даними. Окрім цього, стандарт SQL містить функції для визначення зміни, перевірки та захисту даних.

SQL — це діалогова мова програмування для здійснення запиту і внесення змін до бази даних, а також керування базами даних. Багато баз даних підтримує SQL з розширеннями до стандартної мови. Ядро SQL формує командна мова, яка дозволяє здійснювати пошук, вставку, оновлення і вилучення даних за допомогою використання системи керування і адміністративних функцій. SQL також включає CLI (Call Level Interface) для доступу і управління базами даних дистанційно.

Перша версія SQL була розроблена на початку 1970-х років у IBM. Ця версія мала назву SEQUEL і була призначена для обробки та пошуку даних, що містилися в реляційній базі даних IBM, System R. Мова SQL надалі була стандартизована Американськими Держстандартами (ANSI) в 1986. На початку SQL була

запланована як мова запитів і управління даними, а пізніші модифікації SQL створені продавцями системи управління базами даних, які додали процедурні конструкції, control-of-flow команд і темпоральні розширення мов. З випуском стандарту SQL:1999 такі розширення були формально запозичені як частина мови SQL через Persistent Stored Modules (SQL/PSM) [6].

Критика SQL включає відсутність крос-платформенності, невідповідну обробку відсутніх даних. Часто це неоднозначна граматика і семантика мови.

Мова SQL поділяється на кілька видів елементів:

– пункти (диз'юнкти) (Clauses), що є складовими частинами інструкцій та запитів. (Іноді вони не обов'язкові.);

– вирази (Expressions), які можуть генерувати скалярні значення, або таблиці з стовпчиками і рядками даних;

– предикати (Predicates), які описують умови, результатом яких є значення тризначної логіки SQL (true/false/unknown) або Булеві значення істинності і які використовуються для обмеження ефекту інструкцій та запитів, або для зміни потоку виконання програми;

– запити (Queries), які отримують дані на основі заданих критеріїв.

– інструкції (Statements), які чинять дію на схему даних чи самі дані, або контролюють транзакції, потік виконання програми, з'єднання, сесії, та виконують діагностику;

– інструкції SQL також включають крапку з комою (";") для позначення кінця інструкції. Хоча вона не є обов'язковою на кожній платформі, вона описується як стандартна частина граматики SQL;

– незначимі пропуски загалом ігноруються в інструкціях і запитах SQL, дозволяючи форматувати код SQL з метою покращення читабельності.

1.2.2 Опис функціоналу Oracle Database

Для інтернет-систем (public) і систем масштабу великої організації (enterprise) пропонується продукт Oracle9i Database Enterprise Edition (корпоративна редакція), для якого є цілий набір опцій, архітектурно і функціонально розширюють можливості сервера. Продукт Oracle9i Database Standard Edition (стандартна редакція) орієнтований на організації середнього масштабу або підрозділу в складі великої організації (workgroup). Для персонального використання пропонується "персональний Oracle" (Oracle9i Database Personal Edition), і для систем мобільного зв'язку і невеликих офісів - Oracle9i Database Lite. У стандартній, персональної і мобільного редакціях основний акцент зроблений на невисоку вартість, простоту установки і супроводу. При цьому всі варіанти сервера Oracle мають в своїй основі один і той же вихідний код і функціонально ідентичні, за винятком деяких додаткових опцій, які необхідні для специфічних конфігурацій (наприклад, для підтримки кластерних архітектур необхідна опція Oracle9i Real Application Clusters).

Основна перевага такого підходу до побудови СУБД - це ідентичність коду для всіх варіантів сервера баз даних. Для всіх комп'ютерних платформ і архітектур існує єдина СУБД Oracle, що поставляється в різних версіях, яка поводиться однаково і надає однакову функціональність незалежно від платформи, на якій вона встановлена.

Однією з основних характеристик СУБД Oracle є функціонування системи на більшості платформ, у тому числі на великих ЕОМ, UNIX-серверах, персональних комп'ютерах і так далі.

Іншою важливою характеристикою є підтримка Oracle всіх можливих варіантів архітектур, в тому числі симетричних багатопроцесорних систем, кластерів, систем з масовим паралелізмом і т. Д. Очевидна значимість цих характеристик для великомасштабних організацій, де експлуатується безліч комп'ютерів різних моделей і виробників. В таких умовах фактором успіху є максимально можлива типізація пропонованих рішень, що ставить собі за мету

істотне зниження вартості володіння програмним забезпеченням. Уніфікація систем управління базами даних - один з найбільш значущих кроків на шляху досягнення цієї мети.

Підтримка Oracle більшості популярних комп'ютерних платформ і архітектур досягається за рахунок жорсткої технологічної схеми розробки коду СУБД. Розробку серверних продуктів виконує єдиний підрозділ корпорації Oracle, зміни вносяться централізовано. Після цього всі версії піддаються ретельному тестуванню в базовому варіанті, а потім переносяться на всі платформи, де також детально перевіряються. Можливість перенесення Oracle забезпечується специфічною структурою вихідного програмного коду сервера баз даних. Приблизно 80% програмного коду Oracle - це програми на мові програмування С, є від платформи незалежним. Приблизно 20% коду, що представляє собою ядро сервера, реалізовано на машинно-залежних мовами; і ця частина коду, зрозуміло, переписується для різних платформ.

Одна з відмінних рис сервера Oracle - можливість зберігання і обробки різних типів даних. Дана функціональність інтегрована в ядро СУБД і підтримується модулем interMedia в складі Oracle Database. Він забезпечує роботу з текстовими документами, включаючи різні види пошуку, в тому числі контекстного; роботу з графічними образами більше 20-ти форматів; роботу з аудіо-та відео.

СУБД Oracle не тільки надає розширений набір вбудованих типів даних, але і дозволяє за рахунок використання Object Option конструювати нові типи даних зі специфікацією методів доступу до них. Це означає фактично, що розробники отримують в руки інструмент, що дозволяє будувати структуровані типи даних, безпосередньо відображають об'єкти предметної області.

Oracle включає в себе дуже багато різних компонентів і модулів, ось деякі з них [7]:

– модуль Oracle interMedia - забезпечує підтримку всіх типів даних, у тому числі виконання операцій пошуку по великим текстових документів різних форматів;

– компонент Oracle Enterprise Manager - являє собою універсальний засіб адміністрування баз даних, забезпечене зручним графічним інтерфейсом і дозволяє адміністратору баз даних виконувати широкий спектр операцій над безліччю баз даних Oracle, включаючи створення, модифікацію і видалення будь-яких об'єктів в середині кожної з них;

– модуль Oracle Advanced Replication Option - дозволяє виконувати реплікацію даних в широкому діапазоні можливостей, включаючи синхронну, асинхронну, каскадну і інші типи реплікації;

– модуль Oracle Workflow - являє собою засіб для автоматизації стандартних бізнес-процедур організації, для розробки процедур управління потоками робіт. Він пропонує розширені можливості автоматизації проходження і обробки інформації довільного типу і формалізації складних бізнес процедур і алгоритмів обробки інформації.

Одна з ключових можливостей сервера БД Oracle - механізм зберігання і обробки черг повідомлень, який називається Oracle Advanced Queuing (AQ). Він поставляється разом з сервером баз даних, і його не потрібно ліцензувати окремо. Компонент AQ відноситься до класу Message Oriented Middleware (ПО проміжного шару для обробки повідомлень). Наявність такого компонента дозволяє побудувати на базі сервера повнофункціональну інфраструктуру для обробки повідомлень і виключає необхідність придбання для цієї мети додаткових коштів третіх фірм (таких як IBM MQ Series), забезпечуючи, в той же час, зв'язок з ними в неоднорідних середовищах за рахунок продукту Oracle Messaging Gateways.

Починаючи з версії Oracle8i, до складу сервера (в усі редакції) включена віртуальна Java-машина (JServer Enterprise Edition).

– компонент Oracle Obects for OLE - надає можливість доступу до баз даних Oracle-додатків, розроблених на C++, Microsoft Visual Basic, OLE 2.0.

1.3 Система моніторингу транзакцій

1.3.1 Структура таблиць бази даних

Таблиця «access_points» (таблиця 1.1) – містить перелік всіх точок обміну які зареєстровані в ПЗ Authentic. Дана таблиця розміщена у схемі «auth_system». Актуалізацією інформації, вибором назви IAP та подібними задачами по заповненню даних займаються прикладні адміністратори системи.

Таблиця *1*.1 - Структура таблиці "auth_system.access_points"

Назва поля	Тип даних	Опис поля
iap_name	VARCHAR2(50)	Текстова назва точки обміну
iap_active_flag	VARCHAR2(1)	Ознака активності IAP: ‘Y’ – активний; ‘N’ – неактивний;
iap_min_connections	NUMBER(5, 0)	Мінімальна кількість з’єднань
iap_last_update_ts	TIMESTAMP(6)	Дата останніх змін

Таблиця «processed_transactions» (таблиця 1.2) – містить записи кожної транзакції, яка потрапила до процесінгової системи. Дана таблиця також відноситься до схеми «auth_system».

Запис вставляється тільки після повної обробки фінансового повідомлення, тому відомий системний час завершення обробки транзакції та код результату обробки. Періодично виконується автоматична ротація записів цієї таблиці таким чином, щоби у ній лишались лише транзакції оброблені за останні 180 діб.

Таблиця *1*.2 - Структура таблиці "auth_system.processed_transactions"

Назва поля	Тип даних	Опис поля
trl_id	NUMBER(19, 0)	Унікальний ідентифікатор запису
trl_origin_iap_name	VARCHAR2(50)	Точка обміну походження транзакції
trl_dest_iap_name	VARCHAR2(50)	Точка обміну призначення транзакції

Продовження таблиці *1*.2

Назва поля	Тип даних	Опис поля
trl_masked_pan	VARCHAR2(20)	Маскований номер картки (перші 6 та останні 4 цифри)
trl_system_timestamp	TIMESTAMP(6)	Системний час завершення обробки транзакції
trl_action_result_code	NUMBER(4, 0)	Код відповіді

Таблиця «result_codes» (таблиця 1.3) – містить текстовий опис-розшифрування кодів відповідей системи Authentic. Застосовується там, де потрібно показувати з яким кодом відповіді обробилась транзакція. Таблиця знаходиться у схемі «auth_system», та поставляється у комплексі разом із ПЗ Authentic.

Таблиця *1*.3 - Структура таблиці "auth_system.result_codes"

Назва поля	Тип даних	Опис поля
arc_code	NUMBER(4, 0)	Унікальний ідентифікатор запису
arc_name	VARCHAR2(50)	Текстове значення (тлумачення) коду відповіді
arc_last_update_ts	TIMESTAMP(6)	Дата останніх змін

Ще один об’єкт схеми «auth_system» це таблиця «system_status» (таблиця 1.4) – містить інформацію про стан процесінгової системи, у тому числі дані про її ноди, процеси, точки обміну та активні з’єднання.

Система Authentic зберігає тут завжди актуальну інформацію про свій статус. У програмі моніторингу ці дані використовуються для відображення на вкладці «Статус системи».

Таблиця *1*.4 - Структура таблиці "auth_system.system_status"

Назва поля	Тип даних	Опис поля
sst_uri_schema	VARCHAR2(20)	Схема (тип) об’єкту, статус якого записано (‘iap’, ‘connection’, …)

Продовження таблиці 1.4

sst_uri_ssp	VARCHAR2(200)	Schema Specific Part (URL) об’єкту, статус якого записаної
sst_status	VARCHAR2(1000)	Поточний статус об’єкту
sst_properties	VARCHAR2(2000)	Рядок тексту з властивостями об’єкту. Формат залежить від типу об’єкту

Також є журнальна таблиця «system_event» (таблиця 1.5), яка також відноситься до схеми «auth_system». У цій таблиці міститься журнал всіх зареєстрованих подій що відбулись у авторизаційній системі, таких як: помилки парсингу повідомлення, перевищення часу очікування з’єднання з хостовим інтерфейсом чи успішне встановлення.

Ротація журналу виконується таким чином, щоб у таблиці не було записів старших за 30 днів.

Таблиця *1*.5 - Структура таблиці "auth_system.system_events"

Назва поля	Тип даних	Опис поля
eve_id	NUMBER(19, 0)	Унікальний ідентифікатор події
eve_timestamp	TIMESTAMP(6)	Системний час реєстрації події у системі
eve_short_msg	VARCHAR2(255)	Коротке повідомлення події
eve_long_msg	VARCHAR2(4000)	Розгорнуте повідомлення події
eve_iap_name	VARCHAR2(50)	Текстова назва Interchange Access Point (дивитись таблицю 3.1)
eve_last_update	TIMESTAMP(6)	Час створення запису у базі данних (по цьому полю побудовано індекс)

«authentic_mon» (таблиця 1.6) – це конфігураційна таблиця, що розміщена у домашній схеми «home_schema». Вона дозволяє налаштувати відображення елементів (IAP та заголовків груп – caption) на головній формі моніторингу.

Ці налаштування зчитуються під час кожного запуску програми. У цю таблицю реплікуються усі IAP з таблиці «auth_system.access_points», все інше

може редагувати користувач через спеціальну форму додатку, якщо має на це певні права доступу.

Таблиця *1*.6 - Структура таблиці "home_schema.authentic_mon"

Назва поля	Тип даних	Опис поля
am_item_text	VARCHAR2(50)	Текст елементу моніторингу. Якщо am_item_type = ‘iap’, то тут має бути текстова назва IAP (дивитись таблицю 3.1). Якщо am_item_type = ‘caption’, то значення цього поля буде відображено як напис на екрані (заголовок)
am_item_type	VARCHAR2(7)	Тип елементу моніторингу: caption – заголовок для групи IAP; iap – точка обміну (напрямок);
am_item_visible	NUMBER(1, 0)	Ознака видимості елемента: 1 – елемент показується екрані моніторингу; 0 – елемент ігнорується;
am_iap_role	VARCHAR2(1)	Тільки для am_item_type = ‘iap’. Ознака ролі IAP: ‘A’ – Acquirer – еквaєрський трафік (вхідний); ‘I’ – Issuer – емітентський трафік (вихідний); ‘B’ – Both – змішаний трафік
am_col	NUMBER(2)	Номер стовпчика в якому буде намальовано елемент
am_row	NUMBER(2)	Номер рядка в якому буде намальовано елемент

Ще одна службова таблиця домашньої схеми «authentic_stats» (таблиця 1.7) передбачена для зберігання журналу статистичних та рекордних показників по кількості оброблених транзакцій. Записи додаються з самої програми моніторингу,

що запущена на окремому обліковому записі. Таким чином веденням журналу статистики займається окрему запущений екземпляр додатку.

Таблиця 1.7 - Структура таблиці "home_schema.authentic_stats"

Назва поля	Тип даних	Опис поля
as_timestamp	TIMESTAMP(6)	Дата реєстрації рекорду
as_number	NUMBER(19, 0)	Значення рекорду (рекордна кількість оброблених транзакцій)
am_type	VARCHAR2(1)	Тип рекорду: ‘S’ – по кількості за секунду; ‘M’ – по кількості за хвилину; ‘H’ – по кількості за годину; ‘D’ – по кількості за добу;

1.3.2 Логічна модель бази даних

Діаграма структури бази даних на рисунку 1.3 наочно демонструє взаємозв’язок різних сутностей за ознакою назви точки обміну.

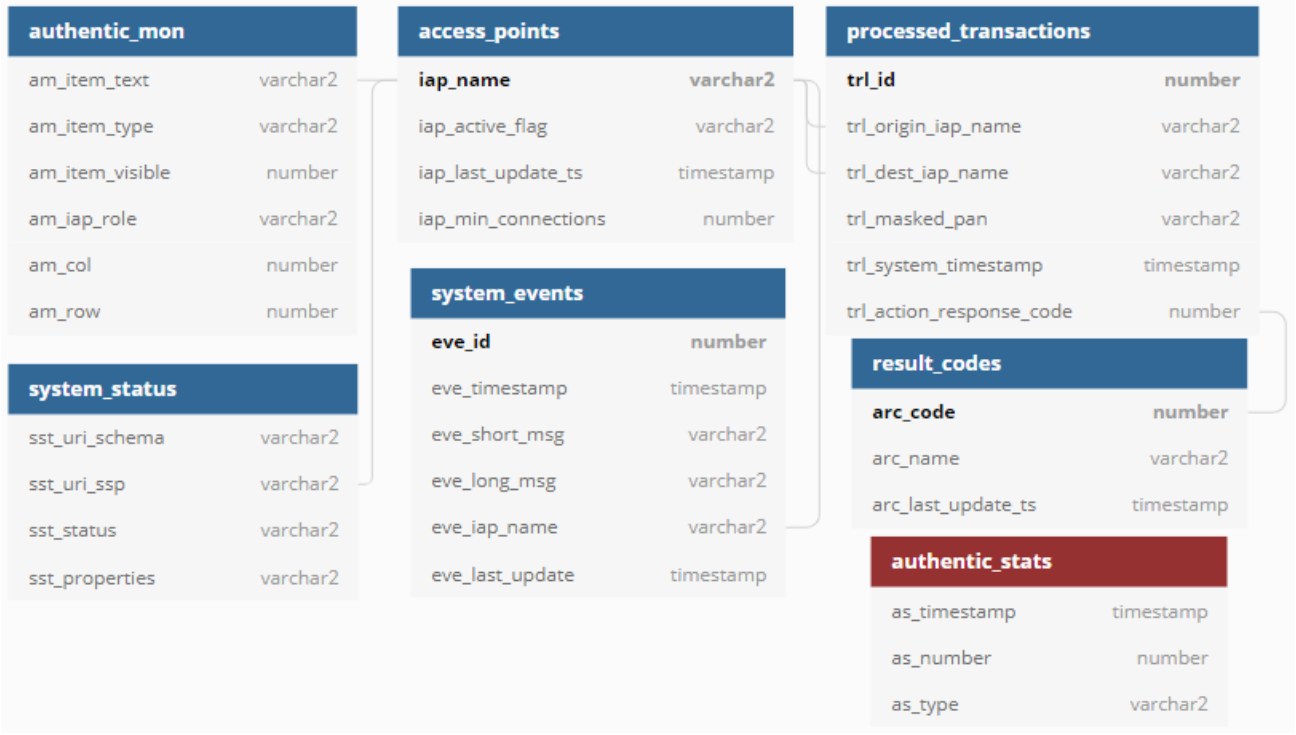

Рисунок 1.3 - Логічна модель бази даних

1.3.3 Алгоритмічна та програмна реалізація

1.3.3.1 Відображення напрямків

Як зазначено на блок-схемі (рисунок 1.4 та 1.5) при запуску моніторингу сканується файл AuthenticTxFlow.exe.config. У цьому конфігураційному файлі знаходяться параметри підключення до бази даних. Якщо цей файл не був знайдений, беруться стандартні дані, які прописані у коді моніторингу. При неуспішній спробі підключення до бази даних на пошту прийде повідомлення, програма видасть помилку та закриється.

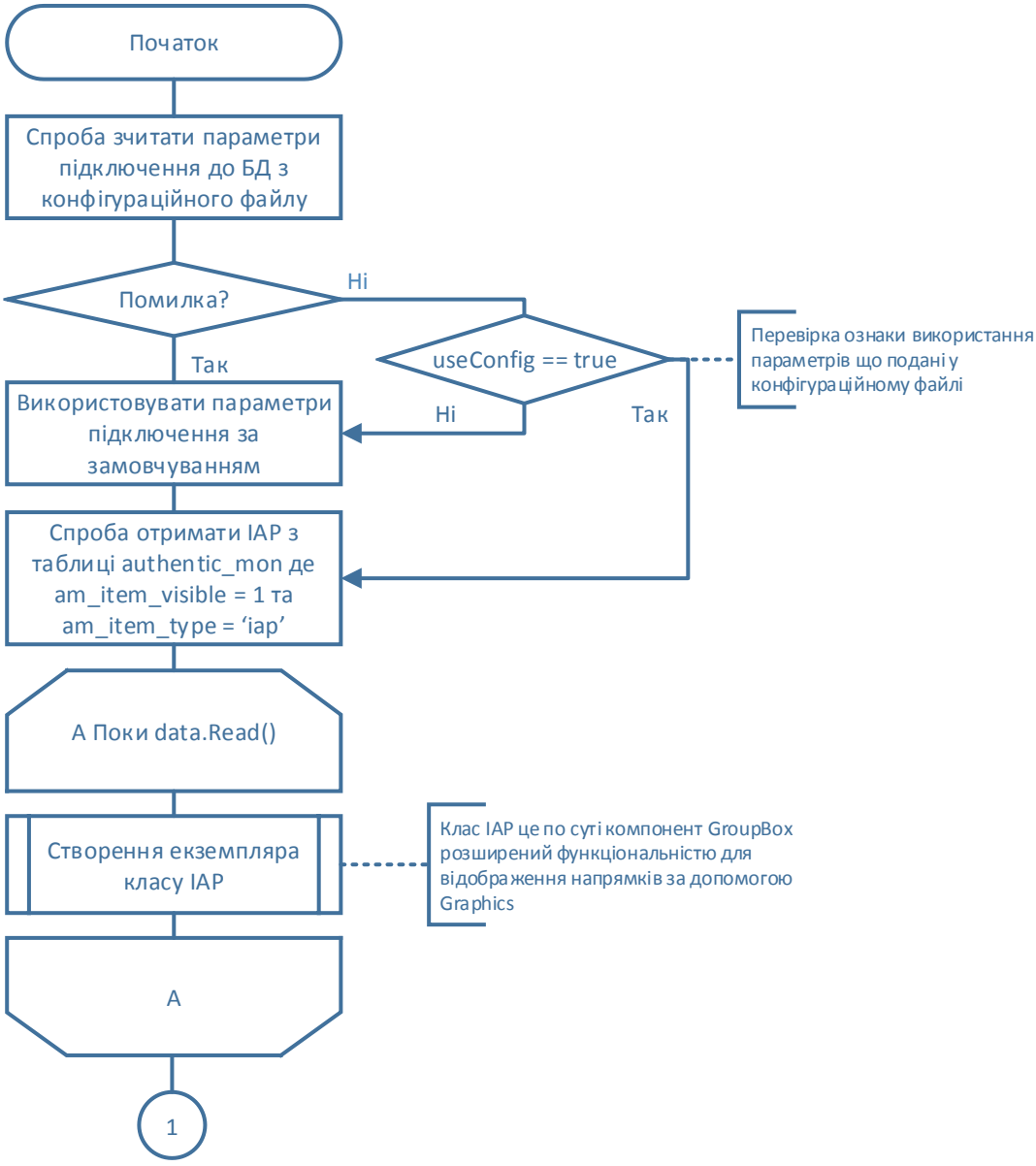

Рисунок 1.4 - Схема алгоритму відображення напрямків. Перша сторінка

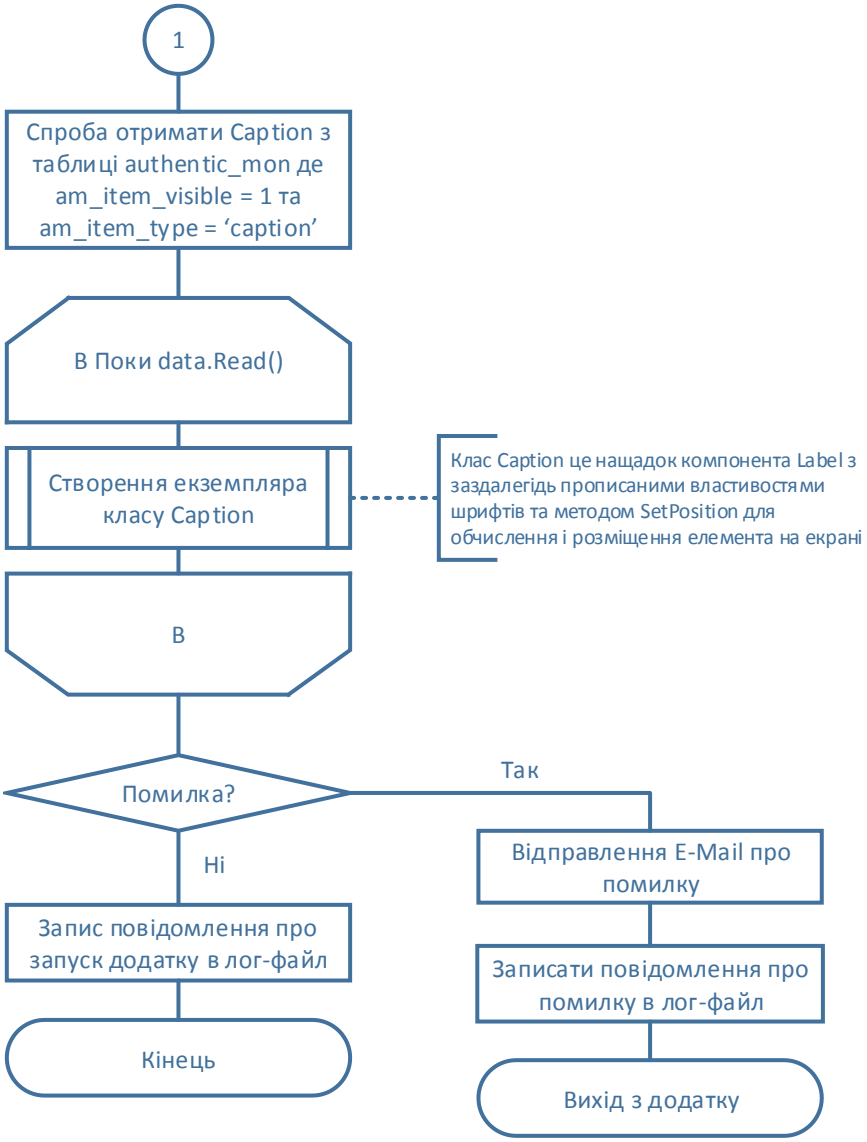

Рисунок 1.5 - Схема алгоритму відображення напрямків. Друга сторінка

Після підключення до БД перевіряється актуальність списку IAP в домашній таблиці home_schema.authentic_mon з таблицею auth_system.access_points. Після актуалізації інформації ініціалізується екземпляр для кожного з IAP та вони розташовуються на моніторі відповідно до значень полів am_col та am_row.

Клас IAP наслідує стандартний компонент GroupBox і розширює його додатковими полями і методами для візуалізації потоку транзакцій.

Клас Caption значно простіший і являє собою наслідника компоненту Label з заздалегідь вказаними стилями (більший розмір кеглю, стиль шрифту тощо).

В результаті роботи конструкторів цих класів елементи моніторингу додаються до форми головного вікна і перераховується їх положення на площині.

На прикладі методу SetPosition() класу Caption можна зрозуміти принцип калькуляції положення елементів на екрані:

```
public void SetPosition()
{
	int formWidth = mainForm.ClientRectangle.Width, formHeight =
mainForm.ClientRectangle.Height - 10;
	int x = formWidth / IAP.MaxCols * (col - 1), y = 0;

	foreach (var iap in IAP.IAPs.Values.Where(o => (o.col == col) && (o.row < row)))
	{
		y += ((IAP)iap).CalculateHeight() + 5;
	}

	foreach (var caption in Captions.Where(o => (o.col == col) && (o.row < row)))
	{
		y += ((Caption)caption).Height;
	}

	this.Width = formWidth / IAP.MaxCols;
	this.captionLabel.Width = this.Width;
	this.Location = new Point(x, y);
}
```

В залежності від «ролі» точки обміну з’являється одна (тільки для вхідного або тільки для вихідного трафіку) або дві (для змішаного трафіку) смужки. За цю логіку відповідає клас TrafficChart. Кожен екземпляр цього класу це одна смужка на екрані моніторингу.

1.3.3.2 Візуалізація потоку транзакцій

Результат запиту до бази даних записується у тип колекції Dictionary та у змінну типу long записується унікальний ідентифікатор останньої транзакції. Після сортування дані відрисовуються на моніторі у вигляді прямокутників за допомогою методу Draw що викликається при настанні події Paint (рисунок 1.6).

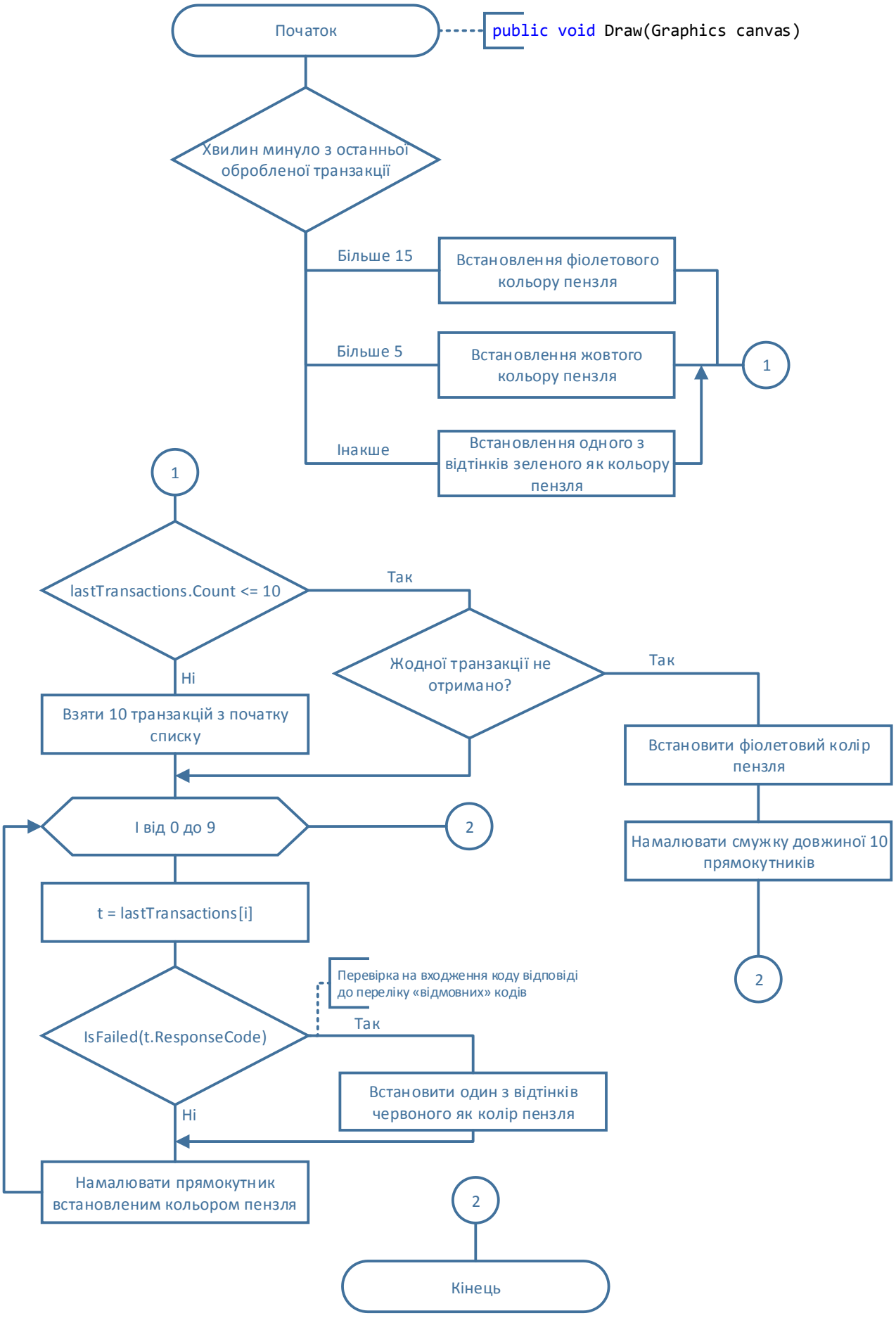

Рисунок 1.6 - Схема алгоритму візуалізації потоку транзакцій

Задля того, щоб прямокутники, що зображають кожну транзакцію не зливалися в одну смужку – колір кожного прямокутника успішно завершеної транзакції змінюється від салатового до темно-зеленого, а відхиленої транзакції – від яскраво червоного до більш темнішого відтінку.

Якщо по якомусь процесу не було транзакцій більш ніж 5 хвилин то уся смуга стає жовтою, якщо 15 хвилин – вона стає фіолетовою.

1.3.3.3 Отримання останніх транзакцій

Якщо відрисовка напрямків пройшла вдало – запускається таймер що працює у окремому потоці, щоб не блокувати основний потік з GUI. При першому спрацюванні таймера відбувається запит транзакцій за останню годину, таким чином користувач, який щойно запустив додаток одразу отримує коректну статистику за останню годину. При послідуючих запитах потрібні транзакції лише за останні 5 секунд, а інші беруться з описаної вище колекції.

До Dictionary кожного IAP додаються нові транзакції та видаляються ті які обробилися більше години тому та відрисовується актуальна інформація по кожному процесу (рисунок 1.7 та 1.8).

Оскільки дані потрібно отримувати майже в псевдо реальному часі з мінімальною затримкою, до отримувати результат від БД потрібно за максимально короткий час. До того ж обов’язково треба вимірювати і контролювати навантаження на сервери бази даних, яке може бути наслідком неправильних та «важких» запитів.

Така реалізація запиту до БД з’явилась після декількох спроб написання і виявилась найбільш вдалою:

```
SELECT
	trl_origin_iap_name,
	trl_dest_iap_name,
	trl_action_response_code,
	trl_system_timestamp,
	trl_id,
	trl_masked_pan
```

```
FROM auth_system.processed_transactions
WHERE trl_system_timestamp BETWEEN sysdate - 5/24/60/60 AND sysdate - 24/60/60
ORDER BY trl_system_timestamp ASC
```

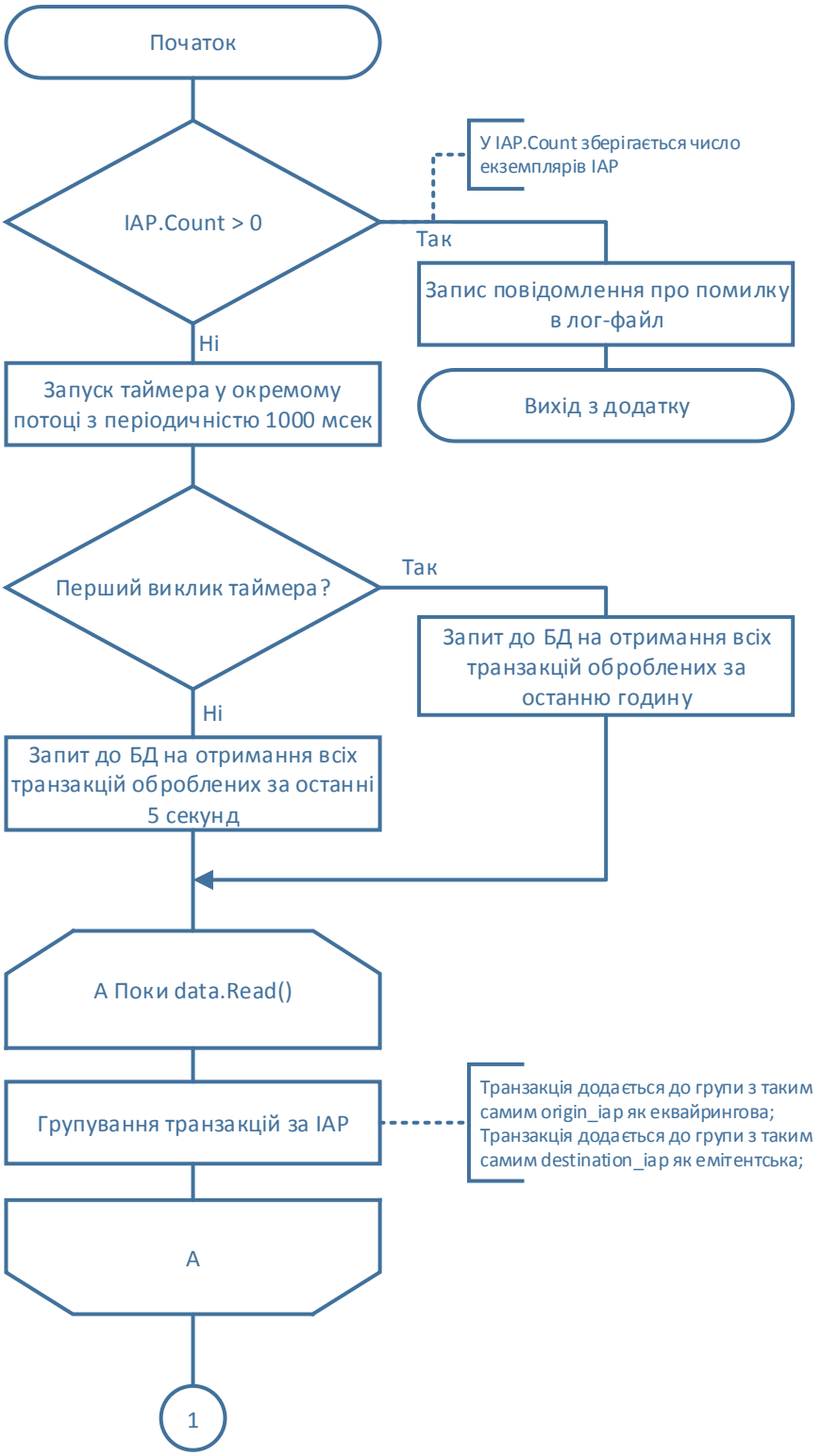

Рисунок 1.7 - Схема алгоритму отримання останніх транзакцій. Перша сторінка

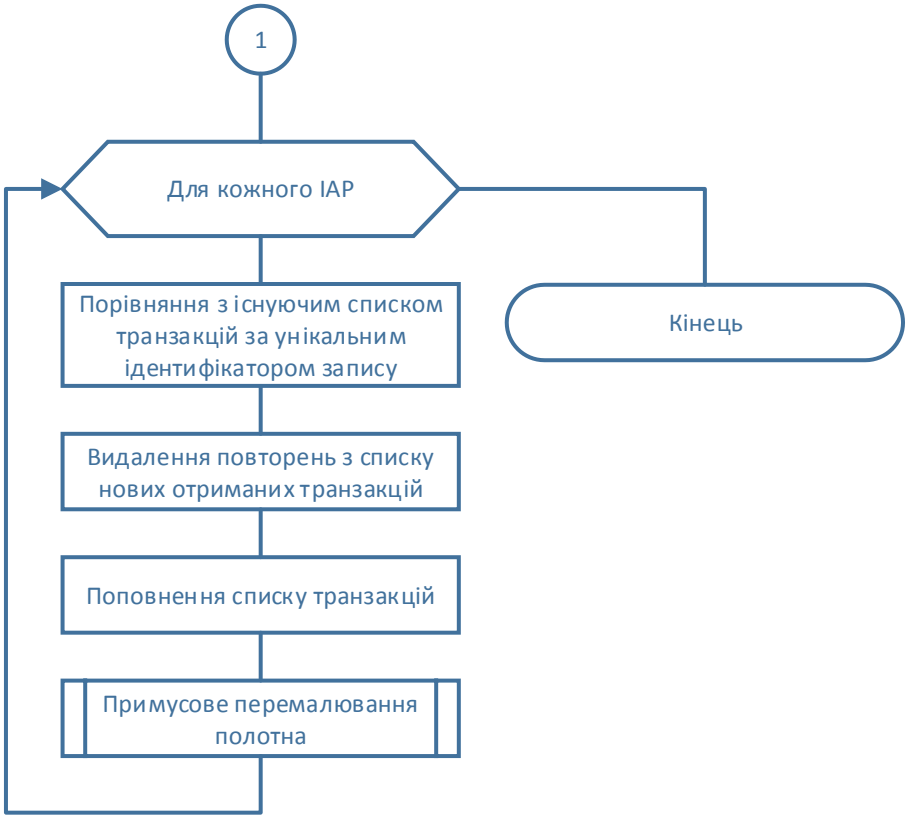

Рисунок 1.8 - Схема алгоритму отримання останніх транзакцій. Друга сторінка

1.3.3.4 Відображення статусу системи

Даний алгоритм спрацьовує у методі iapStatusDraw() (рисунок 1.9 та 1.10), який в свою чергу викликається кожен раз при виклику події відображення (відрисовки) полотна.

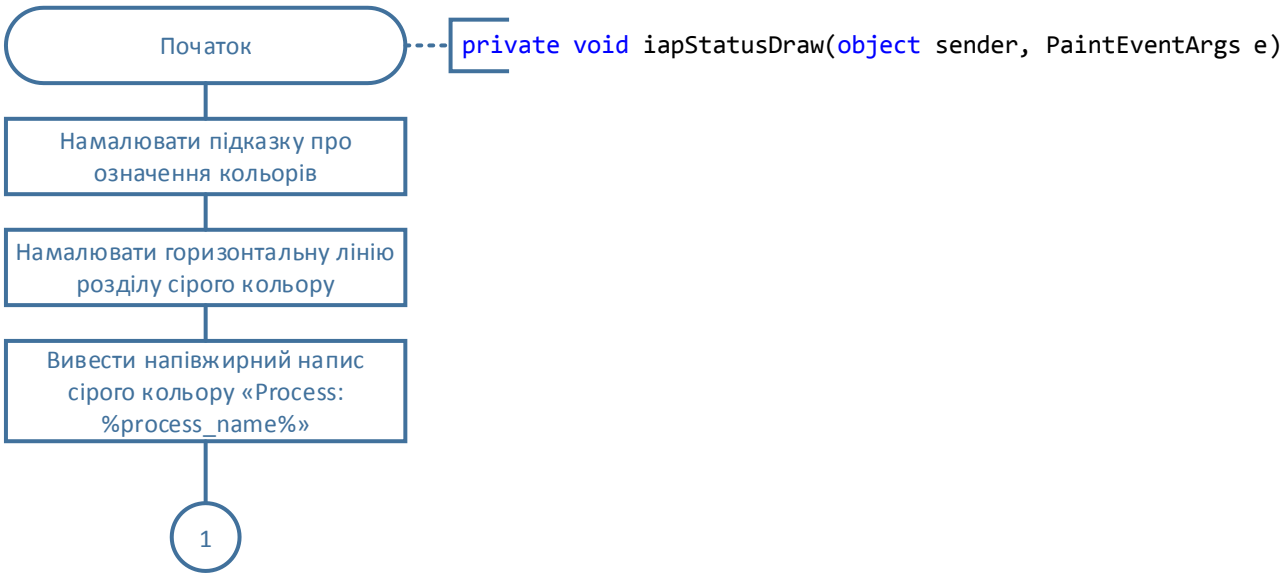

Рисунок 1.9 – Схема алгоритму відображення статусу системи. Перша сторінка

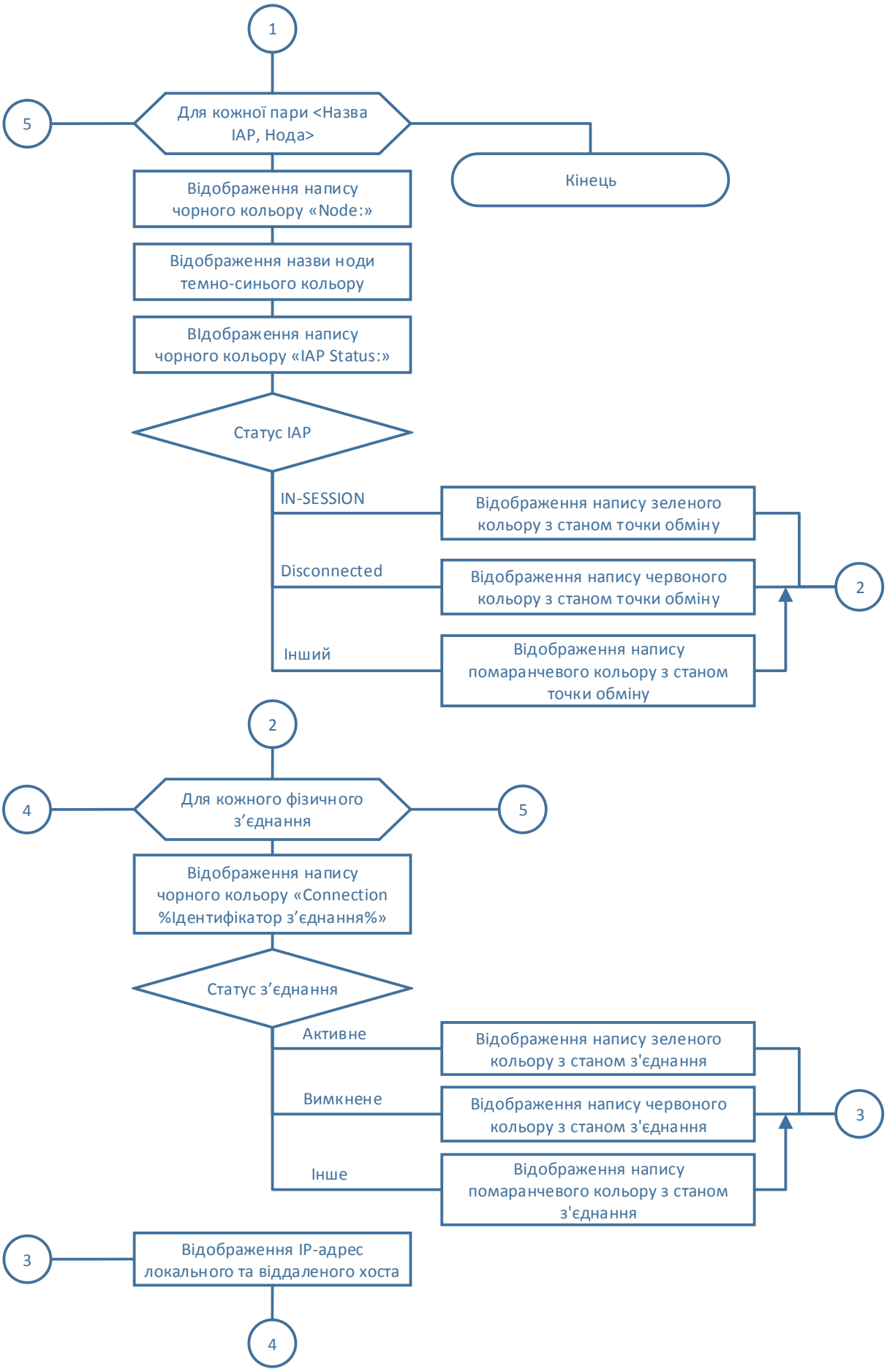

Рисунок 1.10 – Схема алгоритму відображення статусу системи. Друга сторінка

Як і при візуалізації потоку транзакцій, тут використовується бібліотека System.Drawing, яка дозволяє створювати двовимірні зображення.

Дані для виведення знаходяться у структурі KeyValuePair<string, Node>, де string – текстова назва ноди, а Node – об’єкт який містить два поля (рисунок 1.11):

– Status – стан точки обміну на цій ноді у вигляді рядка (string);

– Connections – список (List) об’єктів класу Connection, який містить IP-адресу, стан та порядковий номер фізичного з’єднання.

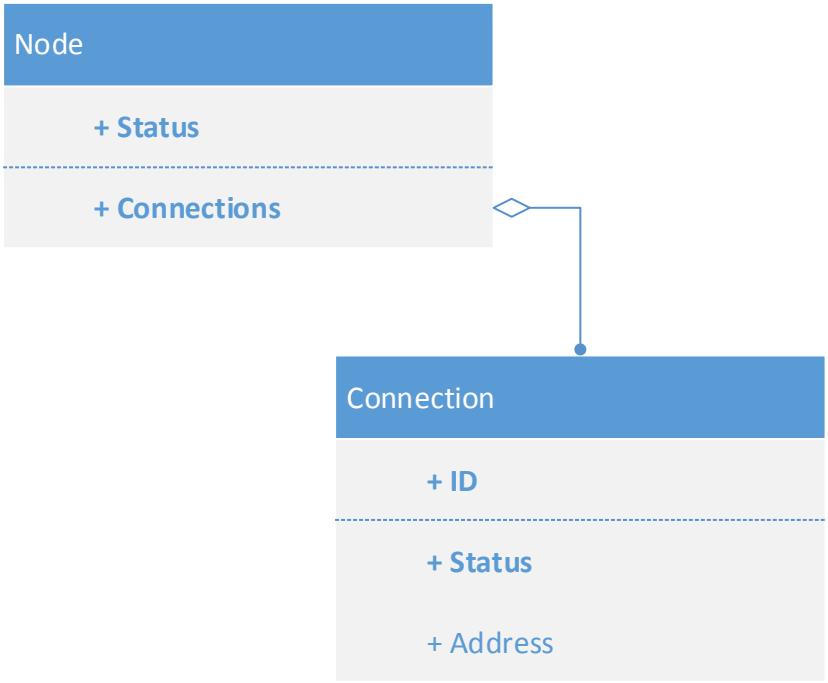

Рисунок 1.11 - Композиційний зв'язок класів Node та Connection

1.3.3.5 Отримання статусу системи

При відкриванні форми детальної інформації про точку обміну, запускається таймер, що викликає метод GetStatus() з відліком у 1 секунду (рисунок 1.12). Задля того щоб при очікуванні на виконання запиту до БД не блокувався основний поток, який відповідає за роботу графічного інтерфейсу.

Отримані після виконання запиту до БД дані підлягають синтаксичному аналізу за допомогою регулярних виразів та записуються до структури KeyValuePair<string, Node>

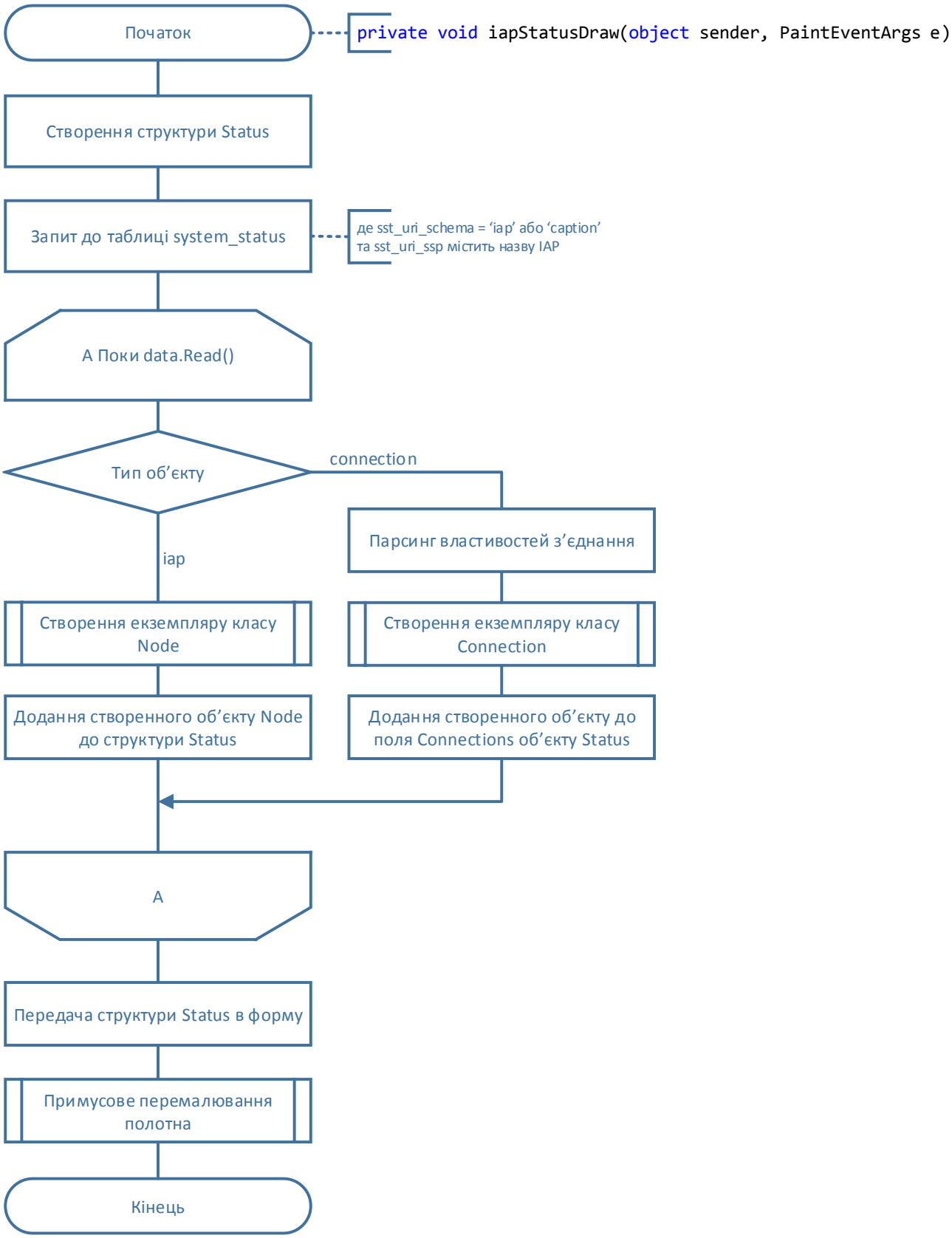

Рисунок 1.12 - Схема алгоритму отримання статусу системи

Для отримання поточного стану нод, процесів, фізичних з'єднань та їх параметрів використовується наступний запит:

```
SELECT
```

```
        sst_uri_schema,
        sst_uri_ssp,
        sst_status,
        sst_properties
FROM auth_system.system_status
WHERE (
        sst_uri_schema = 'iap' OR sst_uri_schema = 'connection'
) AND sst_uri_ssp LIKE '%' || :iapname || '%'
ORDER BY
        sst_uri_schema DESC,
        sst_uri_ssp ASC
```

Де :iapname – параметр запиту, до нього прив’язується текстова назва IAP по якому необхідно отримати дані.

1.3.3.6 Отримання та відображення транзакцій за точкою обміну

Оскільки отриманням транзакцій з бази даних займається інший метод, то в цьому випадку робити запит до БД не потрібно. Всі оброблені транзакції за останню годину зберігаються у списках у екземплярах класу TrafficChart.

Таким чином потрібно лише приєднати списки вихідних та вхідних транзакцій до списку transactionLog та використати його як джерело даних для компоненту DataGridView (рисунок 1.13 та 1.14).

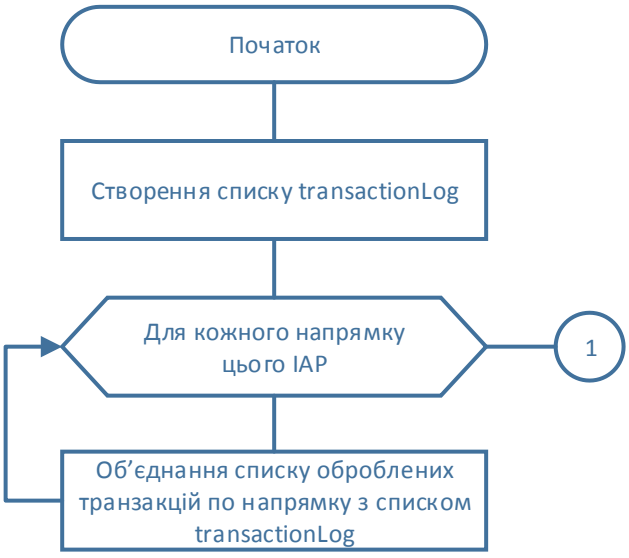

Рисунок 1.13 - Схема алгориту отримання та відображення транзакцій. Перша сторінка

Рисунок 1.14 - Схема алгориту отримання та відображення транзакцій. Друга сторінка

1.3.3.7 Отримання та відображення журналу подій

Щоб отримати список подій за останню добу, надсилається запит з оператором BETWEEN:

```
SELECT
	eve_timestamp,
	eve_short_msg,
	eve_long_msg
FROM auth_system.system_events
WHERE
	eve_iap_name = :iapname AND eve_last_update BETWEEN sysdate - 1 AND sysdate
ORDER BY eve_last_update DESC
```

Де :iapname – параметр запиту, до нього прив’язується текстова назва IAP по якому необхідно отримати дані.

Далі список отриманих подій передається до компоненту DataGridView як джерело даних (рисунок 1.15 та 1.16).

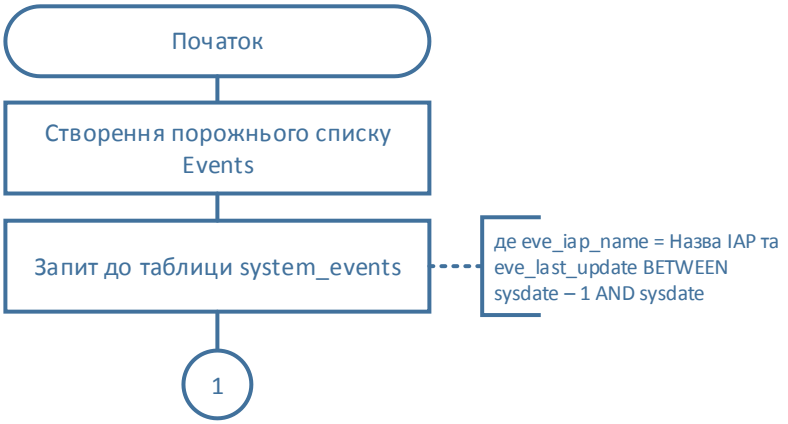

Рисунок 1.15 - Схема алгоритму отримання та відображенн журналу подій. Перша сторінка

Рисунок 1.16 - Схема алгоритму отримання та відображенн журналу подій. Друга сторінка

1.3.4 Користувацький інтерфейс системи

Так як Authentic - інтелектуальна платформа для обробки транзакцій, головними аспектами в моніторингу цієї системи є:

– показник кількості транзакцій, код відповіді котрих не є успішним за годину (менше ніж 3% неуспішних);

– чи всі процеси цієї системи запущені і працюють в штатному режимі;

– чи зі всіма віддаленими хостами є з’єднання.

1.3.4.1 Головне вікно моніторингу

На головному вікні зображені усі IAP згідно конфігурації, поточний час, час обробки останньої транзакції, статистичні дані та рекорди (рисунок 1.17).

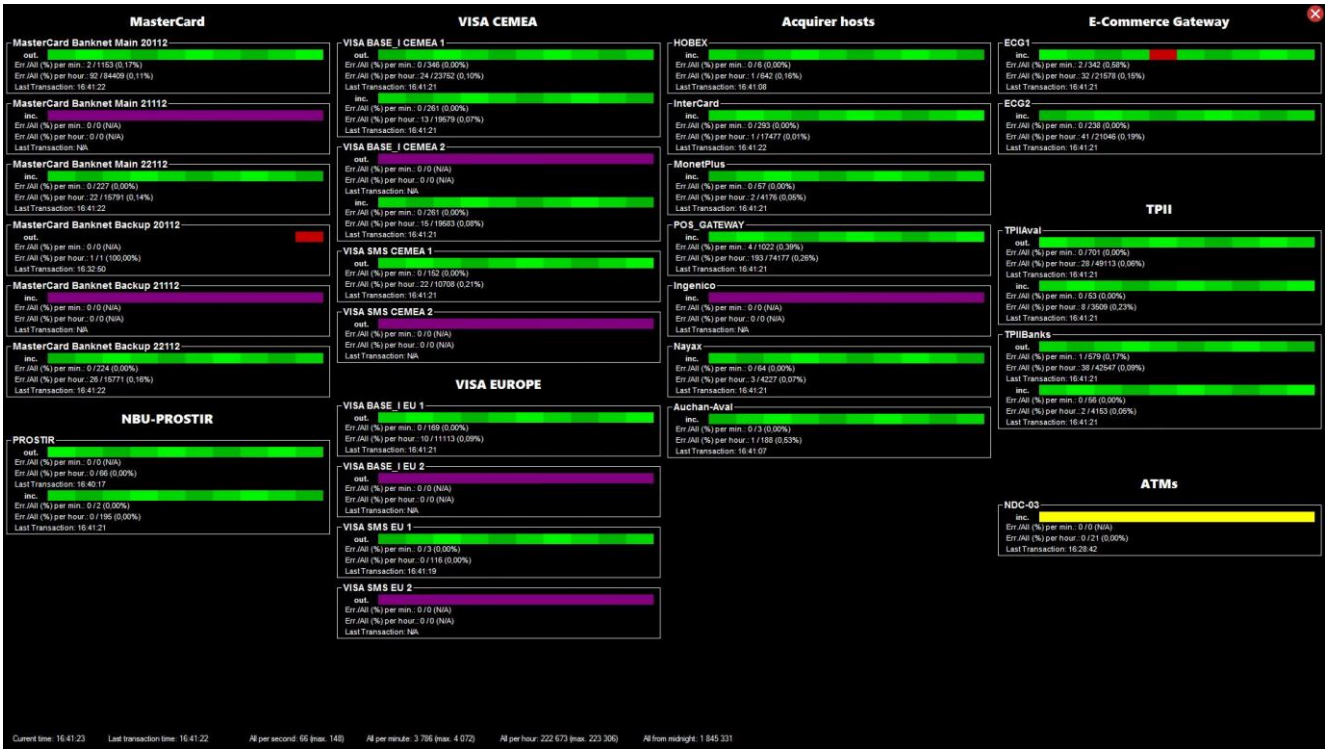

Рисунок 1.17 - Головне вікно моніторингу

Точка обміну може мати як вхідний чи вихідний трафік, так і обидва типи одночасно. Для наочності для кожного напрямку трафіку виокремлено свою смужку. Смуга складається с 10 прямокутників-транзакцій різного кольору.

Зелений колір показує що транзакція пройшла успішно, червоний – транзакція завершилась неуспішно, жовтий – по цьому хосту не було транзакцій більше 5 хвилин, фіолетовий – по цьому хосту не було транзакцій більше 15 хвилин.

1.3.4.2 Детальна інформація про точку обміну

Якщо двічі клацнути лівою кнопкою миші по точці обміну, то відкриється форма з детальною інформацією по даній точці (рисунок 1.18).

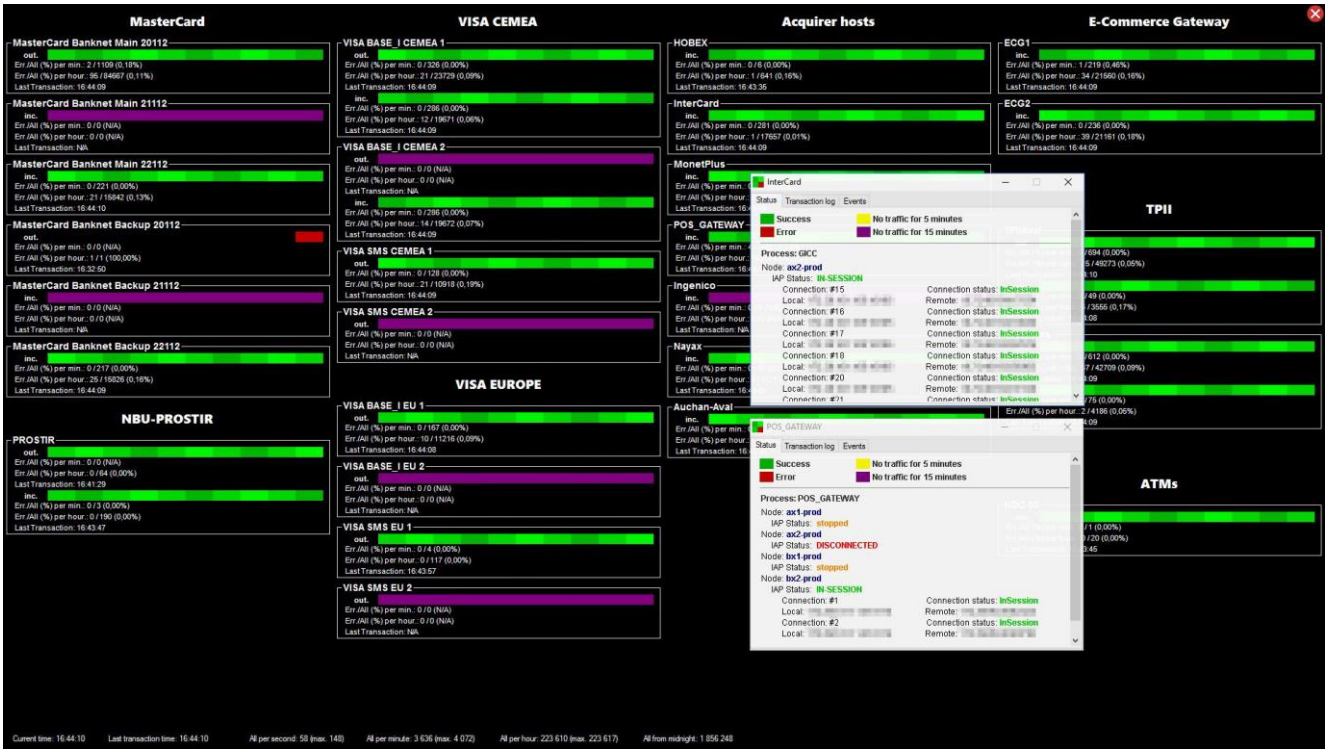

Рисунок 1.18 - Вікно детальної інформації про точку обміну

Для зручності форма поділяється на 3 вкладки:

– «Статус системи»;

– «Історія транзакцій»;

– «Історія подій».

У заголовку форми відображається текстова назва точки обміну, кнопка «Згорнути» та кнопка «Закрити».

1.3.4.3 Статус системи

На першій вкладці показується на якій ноді, від яких процесів зараз працює цей IAP та дані про фізичне з’єднання (IP-адреса та порт локального хосту та IP-адреса та порт віддаленого хосту). На рисунку 1.19 приклад з двома IAP.

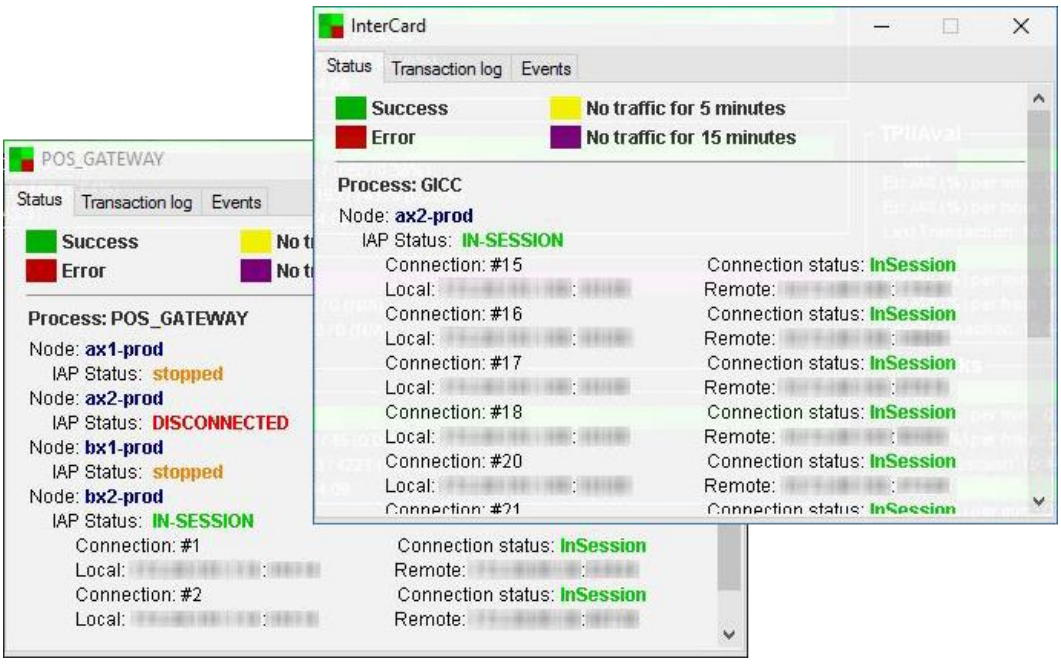

Рисунок 1.21 - Вкладка інформації про статус системи

1.3.4.4 Історія транзакцій

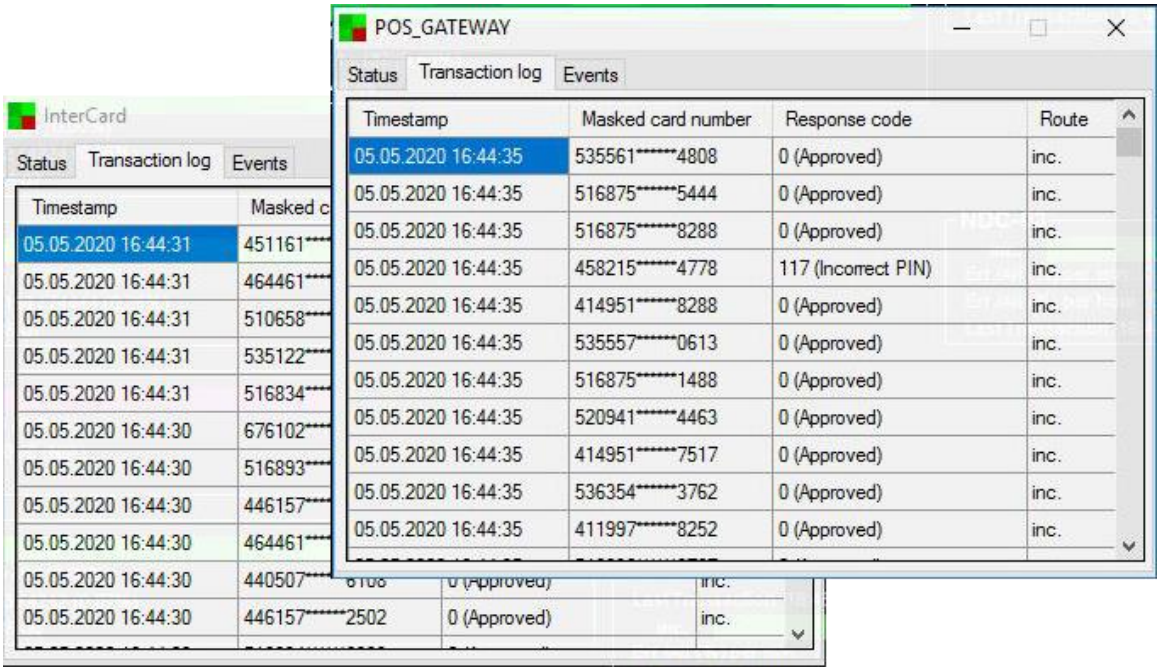

Рисунок 1.22 - Список останніх транзакцій за годину

На рисунку 1.22 показано останні транзакції за годину по напрямках POS_GATEWAY та InterCard, масковані номер картки, коди відповіді та час коли обробка була завершена.

1.3.4.5 Історія подій

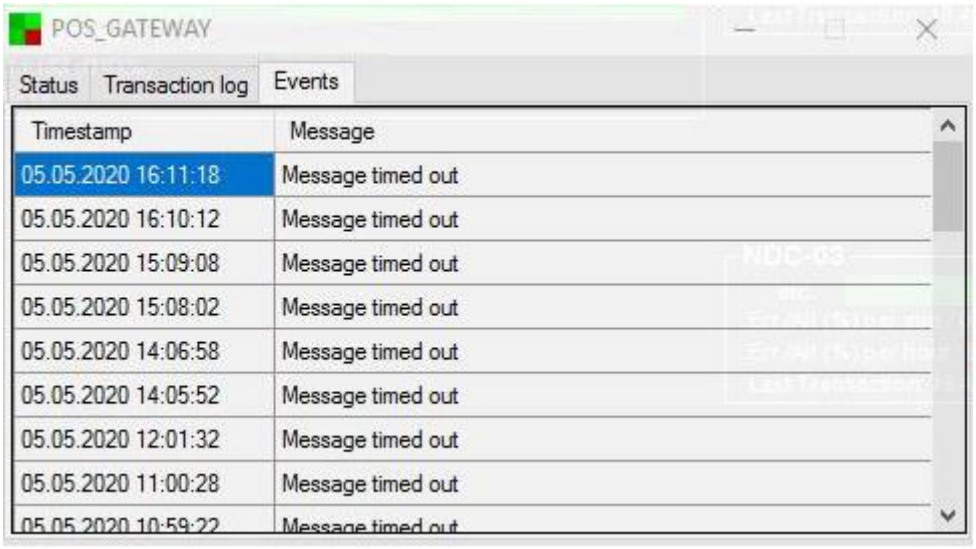

Рисунок 1.23 - Список подій

На рисунку 1.23 показані усі події за останню добу, що пов’язані з цією точкою обміну та час коли ці помилки відбулися.

1.3.4.6 Статистика та рекорди

Також у окрему таблицю в домашній схемі записується рекорди по кількості транзакцій за секунду, хвилину, годину та за цілу добу, як показано на рисунку 1.24.

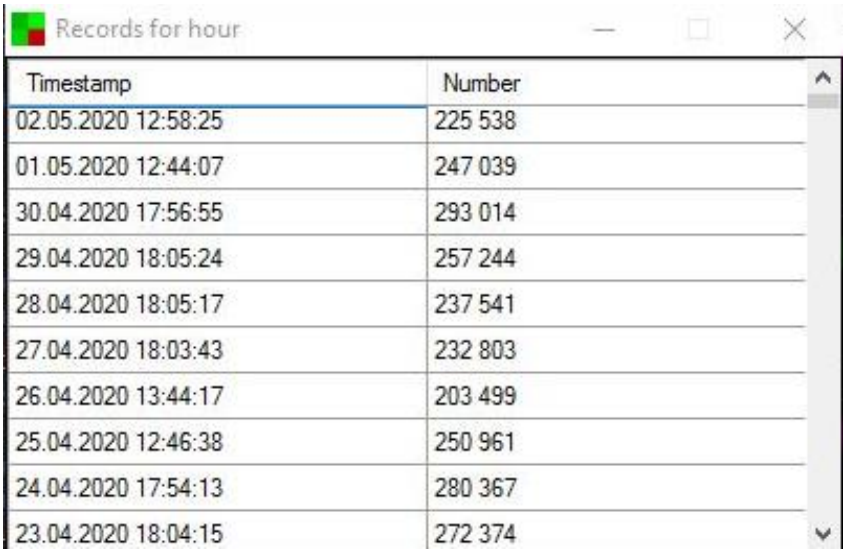

Рисунок 1.24 - Рекорди по кількості транзакцій за добу

1.3.4.7 Авторизацією у вікні налаштувань через Active Directory

Якщо натиснути у будь-якому місці на головному екрані праву клавішу миші, то з’являється доступ до контекстного меню (рисунок 1.25) з двома пунктами. Оскільки програма моніторингу завжди відкривається в повноекранному режимі, то для можливості згорнути програму був зроблений пункт «Minimize».

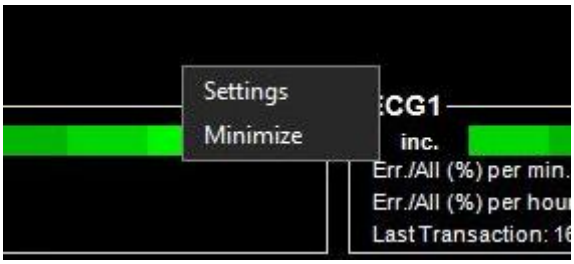

Рисунок 1.25 - Контекстне меню програми

Натиснувши на «Settings», відкривається вікно авторизації. Тут потрібно ввести логін та пароль користувача Active Directory (рисунок 1.26).

Рисунок 1.26 - Вікно авторизації

При відкриванні вікна авторизації у поле «ім’я користувача» автоматично підставляється логін поточного користувача під обліковим записом якого було запущено додаток.

Після заповнення паролю та натискання кнопки «Login» відбувається перевірка на наявність облікового запису з таким паролем у домені Active Directory на підприємстві. Потім перевіряється входження користувача до групи «techsupport» таким чином обмежується несанкціонований доступ до налаштувань моніторингу інших співробітників. У разі виникнення проблем на будь-якому з цих етапів перевірок, користувача буде сповіщено про це а потом знову буде відкрито

вікно авторизації. Якщо всі перевірки пройдені, то відкривається вікно налаштувань (рисунок 1.27).

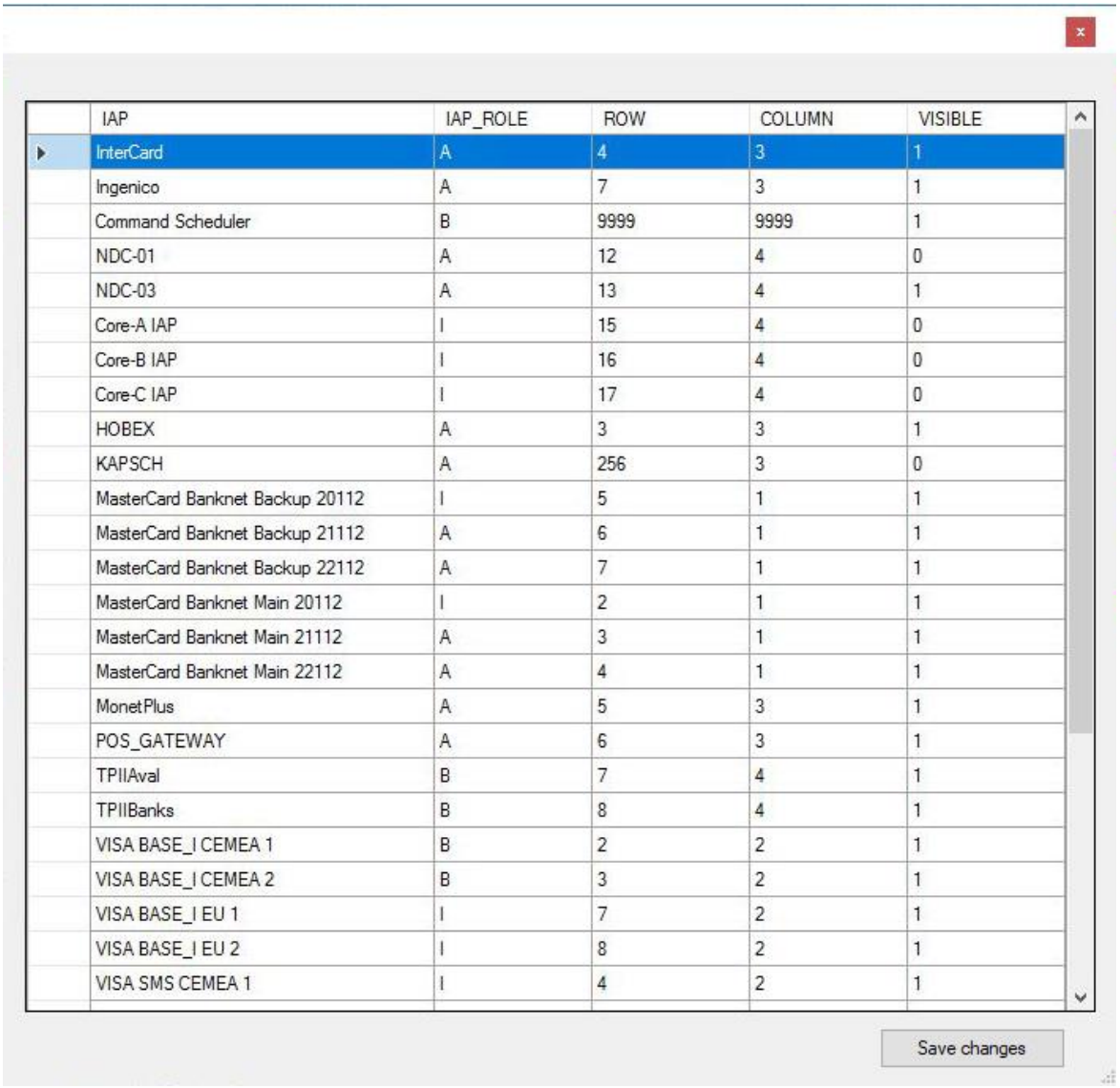

Рисунок 1.27 - Вікно налаштувань

Для початку редагування значення потрібно натиснути на комірку і ввести нове значення. По натисканню на кнопку «Save changes» змінені налаштування будуть записані у БД і вступлять в силу одразу після перезапуску додатку.

2 АНАЛІЗ МЕТОДІВ ЗМЕНШЕННЯ НАВАНТАЖЕННЯ НА СУБД

2.1 Хід обробки запиту

У цьому розділі розглядається, як база даних обробляє оператори DDL для створення об'єктів, DML для модифікації даних та запити для отримання даних [8].

Обробка SQL – це синтаксичний розбір, оптимізація, генерація джерела рядків та виконання оператора SQL. Залежно від твердження, база даних може опускати деякі з цих етапів.

На рисунку 2.1 зображені загальні етапи обробки SQL запиту.

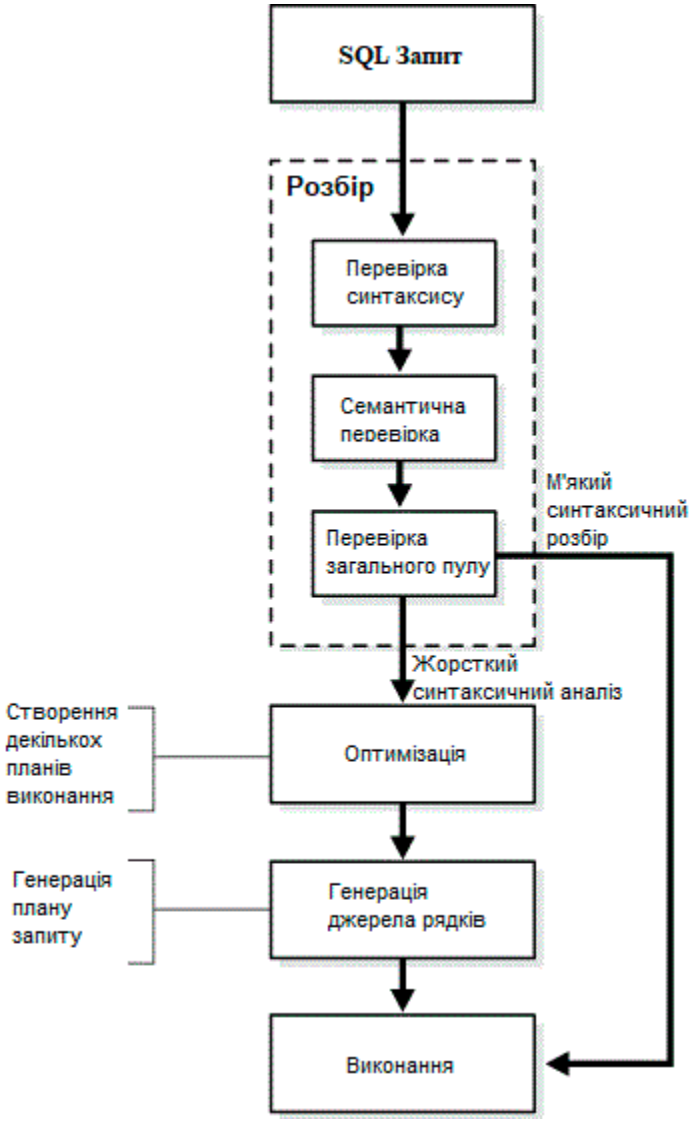

Рисунок 2.1 - Загальні етапи обробки SQL запитів

2.2 Розбір та аналіз плану запиту

База даних Oracle надає можливість переглядати план для кожного запиту, тобто дозволяє визначити послідовність операцій, необхідних для отримання результату SQL запиту в реляційній СУБД. План поділяється на три стадії:

– вибірка результатів;

– сортування й групування;

– здійснення агрегацій.

Для того щоби отримати план запиту необхідно виконати команду EXPLAIN PLAN FOR текст_запиту.

Розглянемо результат роботи EXPLAIN PLAN (рисунок для запиту на отримання транзакцій з підрозділу 1.3.3.3.

```
Plan hash value: 2272195063

-----------------------------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name                        | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |                             |     1 |    58 |     8  (13)| 00:00:37 |
|   1 |  SORT ORDER BY                      |                             |     1 |    58 |     8  (13)| 00:00:37 |
|*  2 |   FILTER                            |                             |       |       |            |          |
|   3 |    TABLE ACCESS BY INDEX ROWID BATCHED| PROCESSED_TRANSACTIONS      |     1 |    58 |     7   (0)| 00:00:37 |
|*  4 |     INDEX SKIP SCAN                 | UPC_TRL_ISS_BANK_TIME_INDX  |     1 |       |     6   (0)| 00:00:37 |
-----------------------------------------------------------------------------------------------------------------------
```

Рисунок 2.2 - План запиту на отримання останніх транзакцій

На скріншоті плану запиту можемо бачити що:

– для вибірки записів використовується пошук за складним індексом (див. підрозділ 2.3.3) UPC_TRL_ISS_BANK_TIME_INDX, побудованим за декількома полями, включаючи trl_system_timestamp;

– доступ до таблиці PROCESSED_TRANSACTIONS здійснюється за допомогою певних пачок (batch) ідентифікаторів рядків (rowid), а не з залученням повного сканування таблиці, що є доброю ознакою;

– прогнозований час виконання запиту за таким планом складає 370 мілісекунд;

– вартість виконання запиту за таким планом складає 8 одиниць та використовує 58 байтів.

Plan hash value – унікальний ідентифікатор запиту, за яким оптимізатор розміщує відомості про план запиту у своєму власному пулі, тобто запам’ятовує сценарій виконання запиту для пришвидшення повторних виконань.

Можемо зробити висновок, що оптимізатор підібрав хороший план виконання запиту на основі його вартості, використовується існуючий у таблиці індекс. Час виконання у 370 мілісекунд є досить великим, враховуючи що запит надсилається кожну секунду, але це все вже пов’язано з великою кількістю записів у таблиці (кожної хвилини у таблицю PROCESSED_TRANSACTIONS вставляється близько 15000 записів!).

2.3 Індексація полів

Індекси Oracle забезпечують швидкий доступ до рядків таблиць, зберігаючи відсортовані значення зазначених стовпців та використовуючи ці відсортовані значення для швидкого знаходження асоційованих рядків таблиці. Індекси дозволяють знаходити рядок із певним значенням стовпця, переглядаючи при цьому лише невелику частину загального обсягу рядків таблиці. Таким чином, правильне використання індексів скорочує до мінімуму кількість дорогих операцій введення-виводу.

Застосування індексів є компромісом між прискоренням отримання результатів запитів та уповільненням оновлень та вставок даних. Перша частина цього компромісу – прискорення запитів – є досить очевидною: якщо пошук виконується за відсортованим індексом замість повного сканування всієї таблиці, то запит проходить набагато швидше. Але коли ви оновлюєте, вставляєте або видаляєте рядок таблиці з індексами, індекси також повинні бути оновлені відповідним чином. Тобто такі операції на таблицях з індексами коштують дорожче.

Взагалі, якщо таблиці в основному використовуються для читання (вибірки) інформації, як у сховищах даних, то краще мати багато індексів. Якщо база даних

відноситься до типу OLTP, з великою кількістю вставок, оновлень і видалень, краще обійтися меншою кількістю індексів.

Якщо вам не потрібно звертатися до більшості записів таблиці, індексовані запити забезпечують швидше отримання результатів, ніж запити, що не використовують індекси. Не існує обмежень на кількість індексів, які можуть стосуватися однієї таблиці Oracle, але, як згадувалося раніше, від їх кількості залежить продуктивність. Індекс повністю прозорий користувача – тобто. оператор SQL користувача не повинен змінюватися внаслідок створення індексів. Однак розробникам додатків для побудови ефективних запитів слід добре уявляти, що таке індекси та як вони працюють.

Індекси можуть належати до кількох типів, найважливіші з яких наведені нижче [9]:

2.3.1 Унікальні та неунікальні індекси

Унікальні індекси ґрунтуються на унікальному стовпці – зазвичай на кшталт номера картки соціального страхування співробітника. Хоча унікальні індекси можна створювати очевидно, Oracle не рекомендує це робити. Натомість слід використовувати унікальні обмеження (unique constraint). Коли накладається обмеження унікальності на стовпець таблиці, Oracle автоматично створює унікальні індекси цих стовпців.

2.3.2 Первинні та вторинні індекси

Первинні індекси – це унікальні індекси в таблиці, які завжди повинні мати якесь значення і не можуть дорівнювати null. Вторинні індекси – це інші індекси таблиці, які можуть бути не унікальними.

2.3.3 Складні (зчеплені) індекси

Складні індекси – індекси, що містять два або більше стовпця з однієї й тієї ж таблиці. Вони також відомі як зчеплені індекси (concatenated index). Складові індекси особливо корисні для забезпечення унікальності поєднання стовпців таблиці в тих випадках, коли немає унікального стовпця, що однозначно ідентифікує рядок.

2.4 Використання підказок оптимізатора

Підказка в мові SQL (англ. hint) – засіб, що дозволяє явно впливати на план запиту [10].

Сам SQL-запит містить вказівку, яку інформацію необхідно отримати з бази даних, але не містить вказівок, як це робити. У випадку, реляційні СУБД за власними правилами визначають план запиту і, відповідно, його виконують. Однак на практиці може виникнути випадок, що такий план запиту, через невраховані засобами СУБД фактори, недосконаліть логіки або особливі вимоги, може виявитися неоптимальним. Підказка дозволяє явно втрутитися у формування плану запиту, повністю не покладаючись на автоматику.

Синтаксис і набір підказок не описаний у стандарті SQL, він залежить від конкретної реалізації СУБД.

Виділяються такі призначення підказок:

- вказівка порядку з'єднання таблиць;
- вказівку методу з'єднання таблиць;
- вказати конкретний індекс для доступу до таблиці.

Нижче наведено всі можливі підказки оптимізатора які актуальні для поточної версії Oracle Database 12c:

2.4.1 Загальні цілі оптимізатора

– /*+ ALL_ROWS */ - визначає на меті якнайшвидше виконання всього запиту з мінімальною витратою ресурсів (best throughput при вилученні всього результуючого набору даних). При одночасному з ALL_ROWS використанні вказівки FIRST_ROWS, що визначає методи доступу до даних (NO_INDEX_SS, INDEX_COMBINE,..) або вказівні методи об'єднання об'єктів БД (LEADING, USE_NL_WITH_INDEX,..), оптимізатор надає перевагу підказкам методів доступу;

– /*+ FIRST_ROWS */ - визначає вартісний підхід (cost-based approach) для оптимізації блоків запиту (statement block) для кращого часу відгуку (response time, мінімальної витрати ресурсів для повернення перших рядків запиту). Відповідно до цієї підказки оптимізатор робить такі переваги (у виборі операцій доступу до даних та методів з'єднання): за наявності оптимізатор використовує сканування індексом (index scan) замість повного сканування таблиці (full table scan). Якщо доступне сканування за індексом (index scan), оптимізатор вибирає nested loops join замість sort-merge join у випадку, коли сканована індексована таблиця може бути використана як ведена таблиця (inner table) для операції nested loops. Якщо використання індексу (index scan) може бути використане для отримання відсортованих даних (у порядку, визначеному фразою ORDER BY), оптимізатор вибирає індексний доступ, щоб уникнути додаткового сортування.

– /*+ FIRST_ROWS(n) */. Оптимізація, заснована на вартості (Cost Based Optimization) та використання правил (уподобань у виборі плану) з метою отримання кращого часу відгуку для отримання перших n рядків. План розраховується з урахуванням значення n як цільової кількості обраних запитом рядків (query cardinality).

Оптимізатор ігнорує цю підказку в SQL пропозиціях DELETE та UPDATE та у запитах SELECT, що включають блокуючі операції, такі як сортування та угруповання. Такі SQL пропозиції не можуть бути оптимізовані з метою найменшого часу відгуку (best response time), оскільки Oracle має опрацювати всі рядки запиту до того, як повернути перший рядок результату. При зазначенні цієї

підказки запити вказаного типу оптимізуються з метою кращого часу отримання всіх рядків запиту з мінімальною витратою ресурсів (best throughput, як із використанням підказки ALL_ROWS);

2.4.2 Хінти порядку доступу

– /*+ LEADING([@query_block] [tablespec],[tablespec],..) */ - вказує оптимізатору використовувати перерахований порядок доступу до таблиць при побудові плану виконання запиту, гнучкіша, ніж підказка ORDERED. Повністю ігноруються при використанні двох або більше підказок LEADING, що конфліктують. Для оптимізатора підказка ORDERED має перевагу проти LEADING;

– /*+ ORDERED */ - вказує Oracle (при виконанні запиту) проводити з'єднання таблиць у тому самому порядку, в якому таблиці перераховані в конструкції FROM. Oracle рекомендує замість ORDERED використовувати підказку LEADING, що має більшу гнучкість, тобто. дає оптимізатору більше можливостей у виборі плану виконання.

2.4.3 Хінти методів з’єднання

– /*+ USE_HASH([@query_block] [tablespec] [tablespec]…) */, /*+ NO_USE_HASH([@query_block] [tablespec] [tablespec]…) */

Ці підказки вказують оптимізатору використовувати або не використовувати операцію hash join для з'єднання кожної вказаної таблиці з іншими джерелами даних;

– /*+ USE_NL ([@query_block] [inner_table]) */ вказує оптимізатору використовувати індексний доступ до таблиць віддаленого представлення;

– /*+ USE_HASH_AGGREGATION([@query_block]) */, /*+ NO_USE_HASH_AGGREGATION([@query_block]) */ вказують оптимізатору

використовувати або не використовувати замість класичної операції Sort group by дещо «нову» операцію угруповання Hash Group By;

– /*+ NATIVE_FULL_OUTER_JOIN */, /*+ NO_NATIVE_FULL_OUTER_JOIN */ - управління використанням механізму Native Full Outer Join;

– /*+ INDEX_JOIN ([@query_block] tablespec [indexspec],…) */ використовувати для отримання результатів запиту тимчасовий індекс, що отримується в результаті об'єднання існуючих індексів методом Index [Hash] Join;

– /*+ INDEX_COMBINE ([@query_block] tablespec [indexspec],…) */ - використовувати для отримання результатів запиту Bitmap операції із ROWID, отриманими при індексному доступі;

– /*+ NUM_INDEX_KEYS(table index numkeys) */ - додана у версії 10.2 для керування кількістю індексних ключів, що використовуються в INLIST ITERATOR з індексом за декількома полями;

2.4.4 Хінти способу виконання запитів/підзапитів

– /*+ DRIVING_SITE ([@query_block] [tablespec]) */. Підказка вказує оптимізатору виконувати запит на сайті (сайт таблиці, вказаної в хінті), відмінному від обраного БД Oracle. Хінт корисний для оптимізації виконання розподілених запитів;

– /*+ MATERIALIZE */. Підказка Materialize є технікою оптимізації запитів і може бути особливо корисною для великих наборів даних. Матеріалізація підзапиту означає створення певного типу динамічної тимчасової таблиці (dynamic temporary table) для використання під час виконання запиту;

– /*+ INLINE */

Протилежна за змістом попередньої підказки /*+ MATERIALIZE*/. Дозволяє у випадках, коли Oracle зобов'язаний матеріалізувати підзапит конструкції WITH без підказок (згідно з правилами виконання subquery factoring, наприклад, коли

підзапит використовується в основному запиті > 1 раз), не робити цього, розглядаючи підзапит як inline view. Корисний у випадках, коли підзапит, що матеріалізується, використовується рідко в з'єднаннях з малою кількістю рядків і накладні витрати на створення temporary table, що передбачає ALL ROWS mode, виявляються більше вартості декількох виконань підзапиту в режимі FIRST ROWS, наприклад, з використанням індексів;

– /*+ PRECOMPUTE_SUBQUERY */ - дозволяє виділити виконання підзапиту в окремий курсор;

2.4.5 Хінти статистики об’єктів

– /*+ DYNAMIC_SAMPLING ([@query_block] [tablespec] degree_of_sampling) */. При використанні підказки DYNAMIC_SAMPLING якщо в базі даних для таблиці є (актуальна) статистика за кількістю рядків, оптимізатор використовує цю статистику. Інакше запити dynamic sampling буде виконано для оцінки цієї статистики.

Підказка підходить для оптимізації звітних запитів з тривалим часом виконання, у випадках, коли час виконання запиту (наприклад, десятки секунд) багато більше часу підготовки запиту (SQL hard parse, що включає dynamic sampling і вибір плану виконання запиту з урахуванням результатів), що становить мілісекунди при значенні рівня degree_of_sampling менше 4.

Рекомендується застосовувати з точною вказівкою таблиці або синоніма - такий синтаксис вказує оптимізатору використовувати dynamic sampling для зазначеної таблиці безумовно;

– /*+ CARDINALITY(objectname [,] integer) */. Підказка застосовується для явної вказівки оптимізатор кількості рядків (cardinality), що повертаються об'єктом БД або з набору даних для розрахунків плану виконання.

Об'єктом можуть бути звичайна (heap) таблиця, глобальна тимчасова таблиця (global temporary table), табличні функції (pipelined function), підзапит тощо;

– /*+ OPT_ESTIMATE([query block] operation_type identifier adjustment) */ - призначений для впливу на план виконання через коригування оцінки кількості рядків (cardinality), одержуваних в результаті операцій доступу до даних і, як наслідок, вартості плану виконання. Часто використовується при створенні SQL Profile, є розвитком/заміщенням підказки CARDINALITY.

2.4.6 Хінти трансформації запитів/підзапитів

– /*+ NO_QUERY_TRANSFORMATION*/ - інструктує оптимізатор пропустити весь етап перетворень запиту (query transformations), включаючи, але не обмежуючись OR-expansion, view merging, subquery unnesting, star transformation, materialized view rewrite.

Крім власне поліпшення плану виконання рахунок виключення неоптимальних операцій/трансформацій, можливо корисно скорочення часу розбору (parse time) запитів з дуже великою кількістю таблиць;

– /*+ PUSH_SUBQ([@query_block]) */ - вказує оптимізатору виконувати неперетворений підзапит (nonmerged subqueries) на ранньому можливому кроці плану виконання запиту. У той час, як завжди, неперетворені підзапити (nonmerged subqueries) виконуються на останньому етапі плану виконання. Раннє виконання підзапиту може значно прискорити продуктивність, якщо підзапит щодо легкий і може значно зменшити кількість рядків, що обробляються»;

– /*+ NO_PUSH_SUBQ([@query_block]) */ - вказує оптимізатору виконувати неперетворений підзапит (nonmerged subqueries) на останньому етапі плану виконання запиту (тобто так, як це зазвичай робить оптимізатор). Може позитивно впливати на продуктивність, якщо підзапит відносно важкий або незначно зменшує кількість рядків». Синтаксис аналогічний синтаксису підказки /*+ PUSH_SUBQ */;

– /*+ NO_UNNEST*/ - додається до підзапиту для запобігання операції об'єднання підзапиту з головним запитом (subquery unnesting)

– /*+ UNNEST*/. Використання підказки UNNEST, навпаки, форсує по можливості операцію об'єднання (unnests specified subquery block if possible) і в цьому прикладі план виконання повертається до дешевшого плану, що використовується за замовчуванням оптимізатором;

– /*+ MERGE([@query_block] [tablespec]) */, /*+ NO_MERGE([@query_block] [tablespec]) */. Використовуються для явного дозволу /*+ MERGE*/ або заборони /*+ NO_MERGE*/ використання оптимізатором механізму об'єднання complex view merging основного запиту та вбудованих оглядів у секції FROM (inline view);

– /*+ NO_EXPAND ([@query_block]) */. Підказка протилежна хінту USE_CONCAT: сигналізує оптимізатору не використовувати перетворення OR-expansion для запитів, що містять диз'юнктивні умови OR або IN-lists у секції WHERE, незважаючи на вартість;

– /*+ PUSH_PRED ([@query_block] [tablespec]) */, /*+ NO_PUSH_PRED ([@query_block] [tablespec]) */. Вказує використовувати або не використовувати CBQT Join Predicate Push-Down з оглядом/таблицею, вказаними у параметрах підказки, наприклад.

– /*+ OR_EXPAND([@query_block] tablespec Column1 [Column2 …]) */

Може бути використаний для ініціювання OR EXPAND query transformation Query Transformation Hints

2.4.7 Підказки, пов'язані з генерацією курсорів

– /*+ CURSOR_SHARING_EXACT */. При вказанні цієї підказки Oracle виконує запит без спроб замінити текстові значення (literals) пов'язаними змінними (bind variables). Незважаючи на значення CURSOR_SHARING = SIMILAR | FORCE, як мінімум, для кожного нового набору literals буде виконуватись розбір SQL (hard parse) - може бути прийнятним рішенням при нечастому виконанні складних звітів.

– /*+ BIND_AWARE*/. Починаючи з версії 11.1.0.7 відключає Adaptive Cursor Sharing (фазу моніторингу для оцінки необхідності подальшого застосування ECS), форсуючи застосування Extended Cursor Sharing для курсорів з різними наборами пов'язаних змінних у випадках, коли ECS не спрацьовує автоматично, наприклад, за відсутності гістограм;

– /*+ NO_BIND_AWARE*/. Відключає застосування технології Extended Cursor Sharing (і, як наслідок, Adaptive Cursor Sharing за відсутністю необхідності) на рівні запиту;

2.4.8 Підказки паралельного виконання

– /*+ PARALLEL */, /*+ SHARED */ - зазвичай використовується Oracle для управління паралельним виконанням підзапиту на дальній стороні db link

2.4.9 Інші підказки

– /*+ QB_NAME(query_block_name) */. Додано з Oracle 10g та використовується для точного зовнішнього визначення назви блоку запиту;

– /*+ OPT_PARAM(parameter_name [,] parameter_value) */ - доданий починаючи з 10g R2, згідно з документацією 11g R2, дозволяє встановити ініціалізаційні параметри оптимізатора на час виконання запиту.

– /*+ APPEND */, /*+ NOAPPEND*/. Підказка APPEND форсує використання оптимізатором direct-path INSERT у запитах виду INSERT INTO… SELECT …;

– /*+ APPEND_VALUES */. Підказка APPEND_VALUES форсує використання оптимізатором режиму прямої вставки direct-path INSERT (тільки) у запитах виду INSERT INTO… VALUES …, доступна з версії 11.2.

2.5 Масштабування бази даних

Масштабованість (scalability) – здатність системи справлятись з зростаючими навантаженнями.

З цього виходить, що масштабування – це процес збільшення здатності системи (в нашому випадку бази даних) до обробки/зберігання більших об’ємів інформації.

Масштабування баз даних поділяють на два типи (рисунок 2.3):

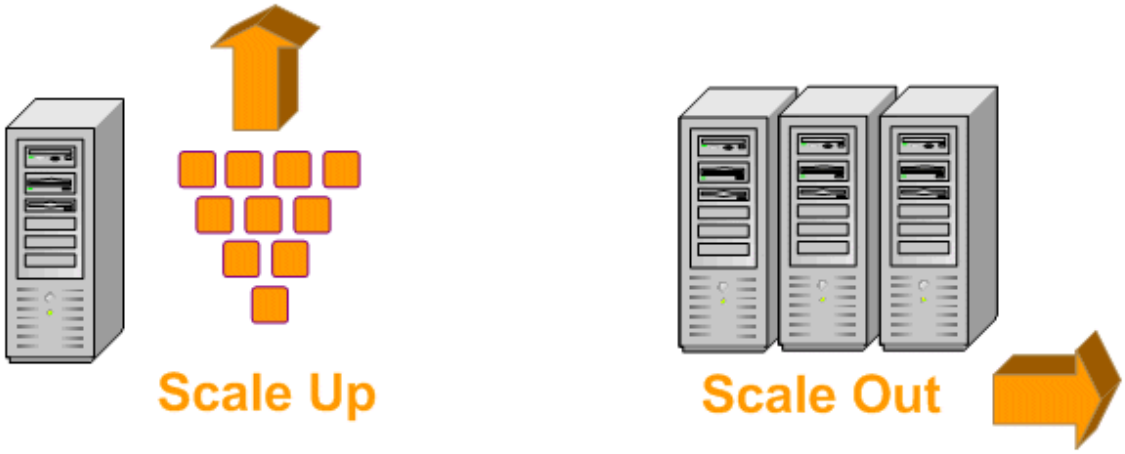

Рисунок 2.3 - Схематичне пояснення різниці між типами масштабування

2.5.1 Вертикальне масштабування

Або scaling up – збільшення кількості доступних для програмного забезпечення ресурсів (рисунок 2.4) за рахунок нарощування потужності використовуваних серверів.

Основною перевагою цього метода є його простота. Немає необхідності переписувати код при нарощуванні потужності, та й керувати одним сервером значно простіше, ніж цілою системою. Однак, це і являє собою головний недолік – масштабування ресурсів одного сервера має конкретні апаратні обмеження. Також варто взяти до уваги вартість такого рішення: сервер з кратним об’ємом

обчислювальних ресурсів переважно виявляється дорожчим, ніж декілька менш потужних серверів, які б давали таку саму сумарну ефективність.

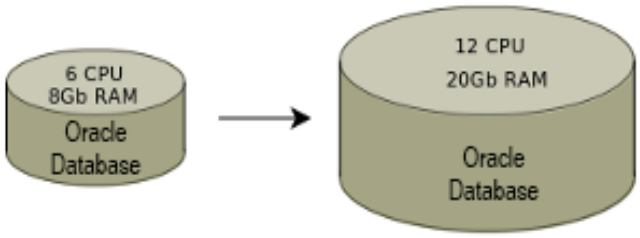

Рисунок 2.4 - Схематичне зображення сутності вертикального масштабування баз даних Oracle

2.5.2 Горизонтальне масштабування

Scale-out або горизонтальне масштабування (рисунок 2.5) є закономірним етапом розвитку інфраструктури. Будь-який сервер має обмеження і коли вони досягнуті або коли вартість потужнішого сервера виявляється невиправдано високою додаються нові машини. Навантаження розподіляється між ними. Також це дає відмовостійкість.

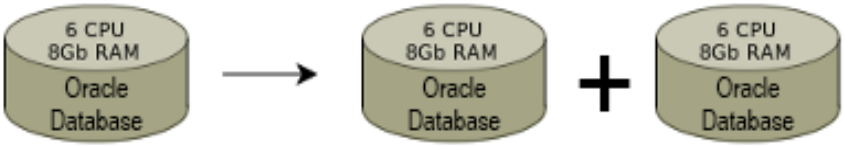

Рисунок 2.5 - Схематичне зображення сутності горизонтального масштабування баз даних Oracle

Існує три основних типи горизонтального масштабування:

2.5.2.1 Реплікація

Цей термін має на увазі копіювання даних між серверами. При використанні такого методу виділяють два типи серверів: master та slave. Майстер використовується для запису або зміни інформації, слейви - для копіювання інформації з майстра та її читання. Найчастіше використовується один майстер і кілька слейв, тому що зазвичай запитів на читання більше, ніж запитів на зміну. Головна перевага реплікації – велика кількість копій даних. Так, якщо навіть головний сервер виходить із ладу, будь-який інший зможе його замінити. Проте як механізм масштабування реплікація не дуже зручна. Причина цього — розсинхронізація та затримки під час передачі даних між серверами. Найчастіше реплікація використовується як засіб для забезпечення стійкості до відмови разом з іншими методами масштабування.

2.5.2.2 Секціонування

Цей метод масштабування полягає у розбитті даних на частини за якоюсь ознакою. Наприклад, таблицю можна розбити на дві за ознакою парності. Причиною для використання партикування є необхідність підвищення продуктивності. Це відбувається через те, що пошук здійснюється не по всій таблиці, а лише за її частиною. Іншою перевагою цього є можливість швидкого видалення неактуального фрагмента таблиці.

2.5.2.3 Шардинг або сегментування

Шардинг — це принцип проєктування бази даних, за яким частини таблиці розміщені на різних фізичних серверах. Шардінг є найбільш підходящим рішенням для великомасштабної діяльності, особливо у поєднанні з реплікацією. Однак слід

зазначити, що це досить складно організувати через необхідність врахування міжсерверного зв’язку.

2.6 Кешування результатів запиту

Кешування є одним із способів оптимізації додатків. У кожній програмі є повільні операції (запити SQL або запити до зовнішніх API), результати яких можна зберігати деякий час. Це дозволить виконувати меншу кількість цих операцій, і більшість користувачів зможуть переглядати раніше збережені дані.

Кешування є найефективнішим, коли екземпляр клієнта багаторазово зчитує одні й ті самі дані, особливо якщо до вихідного сховища даних застосовуються всі наведені нижче умови:

- він залишається відносно статичним;
- це повільно порівняно зі швидкістю кешу;
- він є предметом високого рівня суперечок.

Кешування може значно покращити продуктивність, масштабованість та доступність. Чим більше у вас даних і чим більше користувачів потребує доступу до цих даних, тим більше переваг від кешування. Це пояснюється тим, що кешування зменшує затримку та суперечки, пов’язані з обробкою великої кількості одночасних запитів у вихідному сховищі даних.

Включення належного кешування також може допомогти зменшити затримку, усуваючи повторювані виклики мікросервісів, API та сховища даних. Ключ до ефективного кешування полягає у визначенні найбільш підходящих даних для кешування та кешування у відповідний час. Дані можуть кешуватися на вимогу, коли вперше отримані програмою. Це означає, що програмі потрібно лише один раз отримати дані зі сховища даних і що наступний запит може бути задоволений за допомогою кешу.

3 ВТІЛЕННЯ ЗАХОДІВ З ОПТИМІЗАЦІЇ СИСТЕМИ МОНІТОРИНГУ ТРАНЗАКЦІЙ

3.1 Розділення бази даних на еквайєрингову та емітентську

База даних Українського процесінгового центру містить багато важких таблиць з такими даними як:

- лог оброблених транзакцій;
- параметри конфігурацій банків;
- параметри BIN та їх діапазонів;
- авторизаційні профілі;
- параметри верифікації карток;
- журнали активних операцій тощо.

Оскільки підприємство веде діяльність з надання послуг як банкам-еквайєрам та торговим точкам, так і банкам-емітентам, то явною ознакою за якою можна поділити таблиці є тип банку.

Зважаючи на цей фактор, було прийнято рішення організувати горизонтальне масштабування бази даних Oracle методом секціонування (детальніше про це написано в підрозділі 2.5.2.2).

3.1.1 Виділення нового серверу баз даних

Було виділено додатковий сервер (таблиця 3.1), на якому розміщено новий екземпляр бази даних. Цей екземпляр призначений для зберігання інформації, що стосується лише банків-еквайєрів та вихідних транзакцій, що мають відношення до еквайрингу.

Таблиця 3.1 – Характеристики серверу HPE DL580 Gen9 CTO SVR

Назва	Опис/значення
Процесор	Intel Xeon E7-8890v3
Кількість ядер	18
Тактова частота	2,5 ГГц
Об’єм кешу рівня L3	45 Вт
TDP	150 Вт
Шина QPI	9.6 GT/s
Частота DDR4	1866 Гц
Чіпсет	Intel C602J chipset
Оперативна пам’ять	HPE SmartMemoryDDR4 Registered (RDIMM) and Quad Rank Load Reduced (LRDIMM) 128 Гб
Блок живлення	HPE 1500 W common slot platinum plus hot plug power supply
Здатність до апгрейду	– Upgradeable four (4) processors – Up to Ninety-Six (96) DDR4 DIMM slots – Nine (9) PCIe Gen3 expansion slots – 10 SFF internal HDD/SSD SAS drive bays or 5 SFF internal express bays + 5 SFF internal HDD/SSD drive bays – 5 NVMe SSD drive bays – 4 redundant hot-plug power supplies
Графіка	Integrated Matrox G200 video standard 16MB Flash 256MB DDR 3 with ECC (112MB after ECC and video)
Формфактор	4U rack
Операційня система	Red Hat Enterprise Linux (RHEL)

Нова машина має такі ж самі характеристики потужності як і існуюча.

Існуючий екземпляр бази даних відтепер слугує для збереження інформації тільки по банках-емітентах та вхідних транзакціях.

3.2 Розповсюдження логу транзакцій за допомогою Apache Kafka

Обмін повідомленнями – це спосіб обміну певними даними між процесами, додатками і серверами (як віртуальними, так і фізичними). Такі повідомлення, необхідні для вирішення деяких технічних потреб, вони можуть складатися з будь-яких даних, це може бути як прості текстові повідомлення, так і двійкові дані. Для цього необхідний інтерфейс, керований сторонньою програмою (його називають middleware, також проміжне або міжплатформене ПО). Даним інтерфейсом виступає брокер повідомлень (Message Broker) [11].

Брокери повідомлень – це, як правило, програма-посередник, яка перекладає мову системи з однієї глобальної мови на іншу за допомогою телекомунікаційного середовища. Брокери справляються з прийомом, створенням черги і передачею повідомлень набагато краще і швидше, ніж будь-які непрофільні аналоги і прості обхідні варіанти (як бази дані, демон cron і тому подібне). Для роботи брокери повідомлень використовують буфери, в яких вони можуть зберігати повідомлення, створюючи чергу повідомлень, яка оброблюється автоматично або шляхом запиту [12].

Брокери повідомлень працюють як посередники для різних сервісів (наприклад, для веб-сервера-додатка). Вони можуть значно скоротити навантаження і терміни доставки повідомлень, оскільки завдання, обробка яких зазвичай займає зовсім трохи часу, передаються сторонній програмі, єдина робота якої полягає у виконанні цих завдань (тобто, робочих процесів). Вони також корисні, коли необхідно гарантувати збереженість передачі інформації з одного місця на інше.

Банківська транзакція є нічим іншим як фінансовим повідомленням. Кожен користувач системи моніторингу ініцією свій екземпляр моніторингу, що в свою чергу призводить до кратного збільшення кількості запитів до БД.

3.2.1 Доцільність використання брокеру повідомлень для зменшення навантаження на базу даних

Nsum = N * E;

Де E – кількість співробітників, що одночасно запустили систему моніторингу

N – кількість запитів до БД, які надсилає кожен екземпляр моніторингу щосекунди

Nsum – сумарна кількість запитів, яку надсилають всі одночасно запущені екземпляри додатку

Csum = C * Nsum

C – очікувана кількість циклів операцій, що виконує серверний процесор

Звідси виходить, що

Csum = C * N * E;

Перехід на брокер повідомлень зменшує E до 1, тобто один екзмпляр програми буде отримувати останні транзакції з баз даних і розповсюджувати їх споживачам.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи магістра було проведено аналіз методів оптимізації SQL запитів та розглянуті сучасні методи оптимізації та розподілення навантаження на СУБД Oracle з метою оцінки їх можливостей, переваг та недоліків.

Дослідження показали, що існуючі методи вибірки даних з бази даних є достатньо оптимальними і не мають резервів для оптимізацію. Тому було вирішено застосувати горизонтальне масштабування методом секціонування. Також систему моніторингу транзакцій було переписано з врахуванням використання брокеру повідомлень Apache Kafka для зменшення кількості запитів до СУБД.

У першому розділі дипломної роботи було проведено аналіз об’єкту дослідження, розглянуто архітектуру системи моніторингу транзакцій та її алгоритмічну реалізацію, принципи побудови і основні функціональні можливості системи Authentic.

Проведено проектування та доопрацювання моніторингу системи Authentic.

Результатом роботи є розподілення навантаження на базу даних та зменшення витрат процесорного часу на забезпечення роботи системи моніторингу транзакцій при одночасному користуванні багатьма співробітниками.