Sprawozdanie

Release 1.0

Mateusz Brokos, Szymon Blatkowski

Jul 03, 2025

CONTENTS

CHAPTER

ONE

WPROWADZENIE

Prowadzący: Piotr Czaja

Kurs: Bazy Danych 1

Student oddający prace: Mateusz Brokos

Celem niniejszego raportu jest szczegółowe przedstawienie procesu projektowania, implementacji oraz analizy wydajności relacyjnej bazy danych wspierającej obsługę przychodni lekarskiej. Opracowana baza umożliwia rejestrację pacjentów, lekarzy i wizyt, zapewniając przejrzyste modele danych oraz możliwość generowania raportów i analiz statystycznych. W pracy uwzględniono aspekty normalizacji struktury tabel, optymalizacji zapytań oraz przykłady wykorzystania narzędzi wspomagających dokumentację kodu. Raport dokumentuje zarówno etap projektowy, jak i praktyczne aspekty realizacji rozwiązania.

Celem następnej sekcji jest pogłębienie wiedzy na temat baz danych przeprowadzając rozeznania literaturowe a następnie wyciągnięcie i utrwalenie najważniejszych informacji.

CHAPTER

TWO

PRZEGLĄD LITERATURY

2.1 Wydajność, skalowanie i replikacja

Autorzy

- Mateusz Brokos
- Szymon Blatkowski
- Maciej Gołębiowski

2.2 Wstęp

Celem niniejszej pracy jest omówienie kluczowych zagadnień związanych z wydajnością, skalowaniem oraz replikacją baz danych. Współczesne systemy informatyczne wymagają wysokiej dostępności i szybkiego przetwarzania danych, dlatego odpowiednie mechanizmy replikacji i optymalizacji wydajności odgrywają istotną rolę w zapewnieniu niezawodnego działania aplikacji. Praca przedstawia różne podejścia do replikacji danych, sposoby testowania wydajności sprzętu oraz techniki zarządzania zasobami i kontrolowania dostępu użytkowników. Omówiono również praktyczne rozwiązania stosowane w popularnych systemach baz danych, takich jak MySQL i PostgreSQL.

2.3 Buforowanie oraz zarządzanie połączeniami

Buforowanie i zarządzanie połączeniami to kluczowe mechanizmy zwiększające wydajność i stabilność systemu.

2.3.1 Buforowanie połączeń:

- Unieważnienie (Inwalidacja) bufora: Proces usuwania nieaktualnych danych z pamięci podręcznej, aby aplikacja zawsze korzystała ze świeżych informacji. Może być wykonywana automatycznie (np. przez wygasanie danych) lub ręcznie przez aplikację.
- Buforowanie wyników: Polega na przechowywaniu rezultatów złożonych zapytań w pamięci podręcznej, co pozwala uniknąć ich wielokrotnego wykonywania i poprawia wydajność systemu, zwłaszcza przy operacjach na wielu tabelach.
- Zapisywanie wyników zapytań: Wyniki często wykonywanych zapytań są przechowywane w cache, dzięki czemu aplikacja może je szybko odczytać, co zmniejsza obciążenie bazy danych i przyspiesza odpowiedź.

2.3.2 Zarządzanie połączeniami:

- Monitorowanie parametrów połączeń: Śledzenie wskaźników takich jak czas reakcji, błędy łączenia i ilość przesyłanych danych. Regularne monitorowanie pozwala szybko wykrywać i usuwać problemy, zwiększając stabilność i wydajność systemu.
- Zarządzanie grupami połączeń: Utrzymywanie zestawu aktywnych połączeń, które mogą być wielokrotnie wykorzystywane. Ogranicza to konieczność tworzenia nowych połączeń, co poprawia wydajność i oszczędza zasoby.
- Obsługa transakcji: Kontrola przebiegu transakcji w bazie danych w celu zapewnienia spójności i integralności danych. Wszystkie operacje w transakcji są realizowane jako jedna niepodzielna jednostka, co zapobiega konfliktom.

2.4 Wydajność

Wydajność bazy danych to kluczowy czynnik wpływający na skuteczne zarządzanie danymi i funkcjonowanie organizacji. W dobie cyfrowej transformacji optymalizacja działania baz stanowi istotny element strategii IT. W tym rozdziale omówiono sześć głównych wskaźników wydajności: czas odpowiedzi, przepustowość, współbieżność, wykorzystanie zasobów, problem zapytań N+1 oraz błędy w bazie danych. Regularne monitorowanie tych parametrów i odpowiednie reagowanie zapewnia stabilność systemu i wysoką efektywność pracy. Zaniedbanie ich kontroli grozi spadkiem wydajności, ryzykiem utraty danych i poważnymi awariami.

2.4.1 Klastry oraz indeksy

- Klaster w bazie danych to metoda organizacji, w której powiązane tabele są przechowywane na tym samym obszarze dysku. Dzięki relacjom za pomocą kluczy obcych dane znajdują się blisko siebie, co skraca czas dostępu i zwiększa wydajność wyszukiwania.
- Indeks w bazie danych to struktura przypominająca spis treści, która pozwala szybko lokalizować dane w tabeli bez konieczności jej pełnego przeszukiwania. Tworzenie indeksów na kolumnach znacząco przyspiesza operacje wyszukiwania i dostępu.

2.4.2 1. Współbieżność w bazach danych

Współbieżność w bazach danych oznacza zdolność systemu do jednoczesnego przetwarzania wielu operacji, co ma kluczowe znaczenie tam, gdzie wielu użytkowników korzysta z bazy w tym samym czasie. Poziom współbieżności mierzy się m.in. liczbą transakcji na sekundę (TPS) i zapytań na sekundę (QPS).

Na wysoką współbieżność wpływają:

- Poziomy izolacji transakcji, które równoważą spójność danych i możliwość równoległej pracy – wyższe poziomy izolacji zwiększają dokładność, ale mogą ograniczać współbieżność przez blokady.
- Mechanizmy blokad, które minimalizują konflikty między transakcjami i zapewniają płynne działanie systemu.
- Architektura systemu, zwłaszcza rozproszona, umożliwiająca rozłożenie obciążenia na wiele węzłów i poprawę skalowalności.

Do głównych wyzwań należą:

- Hotspoty danych, czyli miejsca często jednocześnie odczytywane lub modyfikowane, tworzące wąskie gardła.

- Zakleszczenia, gdy transakcje wzajemnie się blokują, uniemożliwiając zakończenie pracy.
- Głód zasobów, kiedy niektóre operacje monopolizują zasoby, ograniczając dostęp innym procesom i obniżając wydajność.

2.4.3 2. Przepustowość bazy danych

Przepustowość bazy danych to miara zdolności systemu do efektywnego przetwarzania określonej liczby operacji w jednostce czasu. Im wyższa, tym więcej zapytań lub transakcji baza obsłuży szybko i sprawnie.

Na przepustowość wpływają:

- Współbieżność: Skuteczne zarządzanie transakcjami i blokadami pozwala na równoczesne operacje bez konfliktów, co jest ważne przy dużym obciążeniu (np. w sklepach internetowych).
- Bazy NoSQL: Często stosują model ewentualnej spójności, umożliwiając szybsze zapisy bez oczekiwania na pełną synchronizację replik.
- Dystrybuowanie danych: Techniki takie jak sharding (NoSQL) czy partycjonowanie (SQL) rozkładają dane na różne serwery, zwiększając zdolność przetwarzania wielu operacji jednocześnie.

Podsumowując, odpowiednie zarządzanie współbieżnością, wybór architektury i rozproszenie danych to klucz do wysokiej przepustowości bazy danych.

2.4.4 3. Responsywność bazy danych

Czasy odpowiedzi bazy danych są kluczowe w środowiskach wymagających szybkich decyzji, np. w finansach czy sytuacjach kryzysowych.

Na czas reakcji bazy wpływają:

- Architektura bazy: dobrze zaprojektowane partycjonowanie, indeksowanie oraz bazy działające w pamięci operacyjnej znacząco przyspieszają dostęp do danych.
- Topologia oraz stan sieci: opóźnienia, przepustowość i stabilność sieci w systemach rozproszonych wpływają na szybkość przesyłu danych; optymalizacja i kompresja zmniejszają te opóźnienia.
- Balansowanie obciążeń oraz dostęp równoczesny: pooling połączeń, replikacja i równoważenie obciążenia pomagają utrzymać krótkie czasy odpowiedzi przy dużym ruchu.

Szybkie odpowiedzi podnoszą efektywność, satysfakcję użytkowników i konkurencyjność systemu bazodanowego.

2.4.5 4. Zapytania N+1

Problem zapytań typu N+1 to częsta nieefektywność w aplikacjach korzystających z ORM, polegająca na wykonywaniu wielu zapytań – jednego głównego i osobnego dla każdego powiązanego rekordu. Na przykład, pobranie 10 użytkowników i osobne zapytanie o profil dla każdego daje łącznie 11 zapytań.

Przyczyny to:

- Błędna konfiguracja ORM, szczególnie „leniwe ładowanie”, powodujące nadmiar zapytań.
- Nieoptymalne wzorce dostępu do danych, np. pobieranie danych w pętlach.

- Niewykorzystanie złączeń SQL (JOIN), które pozwalają na pobranie danych w jednym zapytaniu.

2.4.6 5. Błędy w bazach danych

Błędy wpływające na wydajność bazy danych to istotny wskaźnik kondycji systemu.

Najczęstsze typy błędów to:

- Błędy składni zapytań – wynikają z niepoprawnej składni SQL, powodując odrzucenie zapytania.
- Błędy połączenia – problemy z nawiązaniem połączenia, często przez awarie sieci, błędne konfiguracje lub awarie serwera.
- Błędy limitów zasobów – gdy system przekracza dostępne zasoby (dysk, CPU, pamięć), co może spowalniać lub zatrzymywać działanie.
- Naruszenia ograniczeń – próby wstawienia danych łamiących zasady integralności (np. duplikaty tam, gdzie wymagana jest unikalność).
- Błędy uprawnień i zabezpieczeń – brak odpowiednich praw dostępu skutkuje odmową operacji na danych.

Skuteczna identyfikacja i usuwanie tych błędów jest kluczowa dla stabilności i wydajności bazy danych.

2.4.7 6. Zużycie dostępnych zasobów

Zużycie zasobów w bazach danych to kluczowy czynnik wpływający na ich wydajność.

Najważniejsze zasoby to:

- CPU: Odpowiada za przetwarzanie zapytań i zarządzanie transakcjami. Nadmierne obciążenie może wskazywać na przeciążenie lub nieoptymalne zapytania.
- Operacje I/O na dysku: Odczyt i zapis danych. Wysoka liczba operacji może oznaczać słabe buforowanie; efektywne cache'owanie zmniejsza potrzebę częstego dostępu do dysku i eliminuje wąskie gardła.
- Pamięć RAM: służy do przechowywania często używanych danych i buforów. Jej niedobór lub złe zarządzanie powoduje korzystanie z wolniejszej pamięci dyskowej, co obniża wydajność.

Dobre zarządzanie CPU, pamięcią i operacjami dyskowymi jest niezbędne dla utrzymania wysokiej wydajności i stabilności systemu bazodanowego.

2.4.8 Prostota rozbudowy:

Bazy danych SQL typu scale-out umożliwiają liniową skalowalność przez dodawanie nowych węzłów do klastra bez przestojów i zmian w aplikacji czy sprzęcie. Każdy węzeł aktywnie przetwarza transakcje, a logika bazy jest przenoszona do tych węzłów, co ogranicza transfer danych w sieci i redukuje ruch. Tylko jeden węzeł obsługuje zapisy dla danego fragmentu danych, eliminując rywalizację o zasoby, co poprawia wydajność w porównaniu do tradycyjnych baz, gdzie blokady danych spowalniają system przy wielu operacjach jednocześnie.

2.4.9 Analityka czasu rzeczywistego:

Analityka czasu rzeczywistego w Big Data umożliwia natychmiastową analizę danych, dając firmom przewagę konkurencyjną. Skalowalne bazy SQL pozwalają na szybkie przetwarzanie danych operacyjnych dzięki technikom działającym w pamięci operacyjnej i wykorzystującym szybkie dyski SSD, bez potrzeby stosowania skomplikowanych rozwiązań. Przykłady Google (baza F1 SQL w Adwords) i Facebooka pokazują, że relacyjne bazy danych są efektywne zarówno w OLTP, jak i OLAP, a integracja SQL z ekosystemem Hadoop zwiększa możliwości analityczne przy jednoczesnym ograniczeniu zapotrzebowania na specjalistów.

2.4.10 Dostępność w chmurze:

Organizacje wymagają nieprzerwanej pracy aplikacji produkcyjnych, co zapewnia ciągłość procesów biznesowych. W przypadku awarii chmury szybkie przywrócenie bazy danych bez utraty danych jest kluczowe. Skalowalne bazy SQL realizują to poprzez mechanizmy wysokiej dostępności, które opierają się na replikacji wielu kopii danych, minimalizując ryzyko ich utraty.

2.4.11 Unikanie wąskich gardeł:

W skalowalnych bazach danych SQL rozwiązano problem logu transakcyjnego, który w tradycyjnych systemach często stanowił wąskie gardło wydajności. W klasycznych rozwiązaniach wszystkie rekordy muszą być najpierw zapisane w logu transakcyjnym przed zakończeniem zapytania. Niewłaściwa konfiguracja lub awarie mogą powodować nadmierne rozrosty logu, czasem przekraczające rozmiar samej bazy, co skutkuje znacznym spowolnieniem operacji zapisu, nawet przy użyciu szybkich dysków SSD.

2.5 Skalowanie

Bazy danych SQL nie są tak kosztowne w rozbudowie, jak się często uważa, ponieważ oferują możliwość skalowania poziomego. Ta cecha jest szczególnie cenna w analizie danych biznesowych, gdzie rośnie potrzeba przetwarzania danych klientów z wielu źródeł w czasie rzeczywistym. Obok tradycyjnych rozwiązań dostępne są również bazy NoSQL, NewSQL oraz platformy oparte na Hadoop, które odpowiadają na różne wyzwania związane z przetwarzaniem dużych ilości danych. Skalowanie poziome z optymalnym balansem pomiędzy pamięcią RAM a pamięcią flash pozwala osiągnąć wysoką wydajność. Przykłady nowoczesnych skalowalnych baz SQL, takich jak InfiniSQL, ClustrixDB czy F1, potwierdzają, że tradycyjne bazy SQL mogą efektywnie skalować się wszerz.

2.6 Replikacja

Replikacja danych to proces kopiowania informacji między różnymi serwerami baz danych, który przynosi wiele korzyści: - Zwiększenie skalowalności – obciążenie systemu jest rozdzielane między wiele serwerów; zapisy i aktualizacje odbywają się na jednym serwerze, natomiast odczyty i wyszukiwania na innych, co poprawia wydajność. - Poprawa bezpieczeństwa – tworzenie kopii bazy produkcyjnej pozwala chronić dane przed awariami sprzętu, choć nie zabezpiecza przed błędnymi operacjami wykonywanymi na bazie (np. DROP TABLE). - Zapewnienie separacji środowisk – kopia bazy może być udostępniona zespołom programistycznym i testerskim, umożliwiając pracę na izolowanym środowisku bez ryzyka wpływu na bazę produkcyjną. - Ułatwienie analizy danych – obciążające analizy i obliczenia mogą być wykonywane na oddzielnym serwerze, dzięki czemu nie obciążają głównej bazy danych i nie wpływają na jej wydajność.

2.6.1 Mechanizmy replikacji

Replikacja w bazach danych polega na kopiowaniu i synchronizowaniu danych oraz obiektów z serwera głównego (master) na serwer zapasowy (slave), aby zapewnić spójność i wysoką dostępność danych.

Mechanizm replikacji MySQL działa w następujący sposób: - Serwer główny zapisuje wszystkie zmiany w plikach binarnych (bin-logach), które zawierają instrukcje wykonane na masterze. - Specjalny wątek na masterze przesyła bin-logi do serwerów slave. - Wątek SQL, który odczytuje relay-logi i wykonuje zapisane w nich zapytania, aby odtworzyć zmiany w lokalnej bazie. - Wątek I/O, który odbiera bin-logi i zapisuje je do relay-logów (tymczasowych plików na slave). Podsumowując, replikacja w MySQL polega na automatycznym przesyłaniu i odtwarzaniu zmian, dzięki czemu baza na serwerze zapasowym jest na bieżąco synchronizowana z bazą główną.

2.6.2 Rodzaje mechanizmów replikacji

- Replikacja oparta na zapisie (Write-Ahead Logging): Ten typ replikacji jest często wykorzystywany w systemach takich jak PostgreSQL. Polega na tym, że zmiany w transakcjach są najpierw zapisywane w dzienniku zapisu, a następnie jego zawartość jest kopiowana na serwery repliki.
- Replikacja oparta na zrzutach (Snapshot-Based Replication): W niektórych systemach stosuje się okresowe tworzenie pełnych zrzutów bazy danych, które są przesyłane do serwerów repliki.
- Replikacja oparta na transakcjach (Transaction-Based Replication): W tym modelu każda transakcja jest przekazywana i odtwarzana na serwerach repliki, co sprawdza się w systemach wymagających silnej spójności.
- Replikacja asynchroniczna i synchroniczna: W replikacji asynchronicznej dane najpierw trafiają do głównej bazy, a potem na repliki. W replikacji synchronicznej zapisy są wykonywane jednocześnie na serwerze głównym i replikach.
- Replikacja dwukierunkowa (Bi-Directional Replication): Pozwala na wprowadzanie zmian na dowolnym z serwerów repliki, które są synchronizowane z pozostałymi, co jest szczególnie użyteczne w systemach o wysokiej dostępności.

PostgreSQL oferuje różne metody replikacji, w tym opartą na zapisie (WAL), asynchroniczną, synchroniczną oraz replikację logiczną. Mechanizm WAL zapewnia bezpieczeństwo danych przez zapisywanie wszystkich zmian w dzienniku przed ich zastosowaniem i przesyłanie go na repliki. W trybie asynchronicznym dane trafiają najpierw na serwer główny, a potem na repliki, natomiast w trybie synchronicznym zapisy są realizowane jednocześnie. Dodatkowo, replikacja logiczna umożliwia kopiowanie wybranych tabel lub baz, co jest przydatne w przypadku bardzo dużych zbiorów danych.

2.6.3 Zalety i Wady replikacji

Zalety:

- Zwiększenie wydajności i dostępności: Replikacja pozwala rozłożyć obciążenie zapytań na wiele serwerów, co poprawia wydajność systemu. Użytkownicy mogą kierować zapytania do najbliższych serwerów repliki, skracając czas odpowiedzi. W przypadku awarii jednego serwera pozostałe repliki kontynuują obsługę zapytań, zapewniając wysoką dostępność.
- Ochrona danych: Replikacja wspiera tworzenie kopii zapasowych i odzyskiwanie danych. W razie awarii głównej bazy replika może służyć jako źródło do odtworzenia informacji.
- Rozproszenie danych geograficzne: Umożliwia przenoszenie danych do różnych lokalizacji. Międzynarodowa firma może replikować dane między oddziałami, co pozwala lokalnym użytkownikom na szybki dostęp.

- Wsparcie analizy i raportowania: Dane z replik mogą być wykorzystywane do analiz i raportów, co odciąża główną bazę danych i utrzymuje jej wysoką wydajność.

Wady:

- Replikacja nie gwarantuje, że po wykonaniu operacji dane na serwerze głównym zostaną w pełni odzwierciedlone na serwerze zapasowym.
- Mechanizm nie chroni przed skutkami działań, takich jak przypadkowe usunięcie tabeli (DROP TABLE).

2.7 Kontrola dostępu i limity systemowe

Limity systemowe w zarządzaniu bazami danych określają maksymalną ilość zasobów, które system jest w stanie obsłużyć. Są one ustalane przez system zarządzania bazą danych (DBMS) i zależą od zasobów sprzętowych oraz konfiguracji. Na przykład w Azure SQL Database limity zasobów różnią się w zależności od wybranego poziomu cenowego. W MySQL maksymalny rozmiar tabeli jest zwykle ograniczony przez parametry systemu operacyjnego dotyczące wielkości plików.

Kontrola dostępu użytkowników w DBMS to mechanizm umożliwiający lub blokujący dostęp do danych. Składa się z dwóch elementów: uwierzytelniania, czyli potwierdzania tożsamości użytkownika, oraz autoryzacji, czyli ustalania jego uprawnień. Wyróżnia się modele takie jak Kontrola Dostępu Uzależniona (DAC), Obowiązkowa (MAC), oparta na Rolach (RBAC) czy na Atrybutach (ABAC).

PostgreSQL oferuje narzędzia do zarządzania limitami systemowymi i kontrolą dostępu. Administratorzy mogą ustawiać parametry takie jak maksymalna liczba połączeń, limity pamięci, maksymalny rozmiar pliku danych czy wielkość tabeli. W zakresie kontroli dostępu PostgreSQL zapewnia mechanizmy uwierzytelniania i autoryzacji. Administratorzy mogą tworzyć role i nadawać uprawnienia dotyczące baz danych, schematów, tabel i kolumn. PostgreSQL obsługuje uwierzytelnianie oparte na hasłach i certyfikatach SSL, umożliwiając skuteczne zarządzanie bezpieczeństwem i poufnością danych.

2.8 Testowanie wydajności sprzętu na poziomie OS

Testy wydajności kluczowych komponentów sprzętowych na poziomie systemu operacyjnego są niezbędne do optymalizacji działania baz danych. Obejmują oceny pamięci RAM, procesora (CPU) oraz dysków twardych (HDD) i SSD — elementów mających największy wpływ na szybkość i efektywność systemu. Analiza wyników pomaga wskazać elementy wymagające modernizacji lub optymalizacji, co pozwala podnieść ogólną wydajność systemu bazodanowego, niezależnie od używanego oprogramowania.

Testy pamięci RAM pozwalają zmierzyć jej szybkość i stabilność, co przekłada się na wydajność bazy danych. W tym celu często stosuje się narzędzia takie jak MemTest86.

Testy procesora oceniają jego moc obliczeniową i zdolność do przetwarzania zapytań. Popularnym programem jest Cinebench R23.

Testy dysków sprawdzają szybkość operacji odczytu i zapisu, co jest kluczowe, ponieważ baza danych przechowuje dane na nośnikach dyskowych. Do pomiarów wykorzystuje się narzędzia takie jak CrystalDiskMark 8 czy Acronis Drive Monitor.

2.9 Podsumowanie

W pracy przedstawiono kluczowe zagadnienia związane z zarządzaniem bazami danych, w tym rodzaje replikacji, metody kontroli dostępu użytkowników, limity systemowe oraz znaczenie testów wydajności komponentów sprzętowych. Omówiono zalety i wady replikacji, takie jak zwiększenie dostępności czy ryzyko niespójności danych. Scharakteryzowano mechanizmy uwierzytelniania i autoryzacji, które zapewniają bezpieczeństwo informacji, oraz wskazano, jak limity zasobów wpływają na działanie systemu. Zwrócono także uwagę na rolę testów pamięci RAM, procesora i dysków w optymalizacji wydajności środowiska bazodanowego. Całość podkreśla znaczenie świadomego projektowania i utrzymywania infrastruktury baz danych w celu zapewnienia jej niezawodności, bezpieczeństwa i wysokiej efektywności pracy.

2.10 Bibliografia

[1] PostgreSQL Documentation – Performance Tips https://www.postgresql.org/docs/current/performance-tips.html

[2] SQLite Documentation – Query Optimizer Overview https://sqlite.org/optoverview.html

[3] F. Hecht, Scaling Database Systems https://www.cockroachlabs.com/docs/stable/scaling-your-database.html

[4] DigitalOcean, How To Optimize Queries and Tables in PostgreSQL https://www.digitalocean.com/community/tutorials/how-to-optimize-queries-and-tables-in-postgresql

[5] PostgreSQL Documentation – High Availability, Load Balancing, and Replication https://www.postgresql.org/docs/current/different-replication-solutions.html

[6] SQLite Documentation – How Indexes Work https://www.sqlite.org/queryplanner.html

[7] Redgate, The Importance of Database Performance Testing https://www.red-gate.com/simple-talk/sql/performance/the-importance-of-database-performance-testing/

[8] Materiały kursowe przedmiotu „Bazy Danych”, Politechnika Wrocławska, Piotr Czaja.

CHAPTER

THREE

PROJEKTOWANIE BAZY DANYCH - MODELE

author
Mateusz Brokos, Szymon Blatkowski

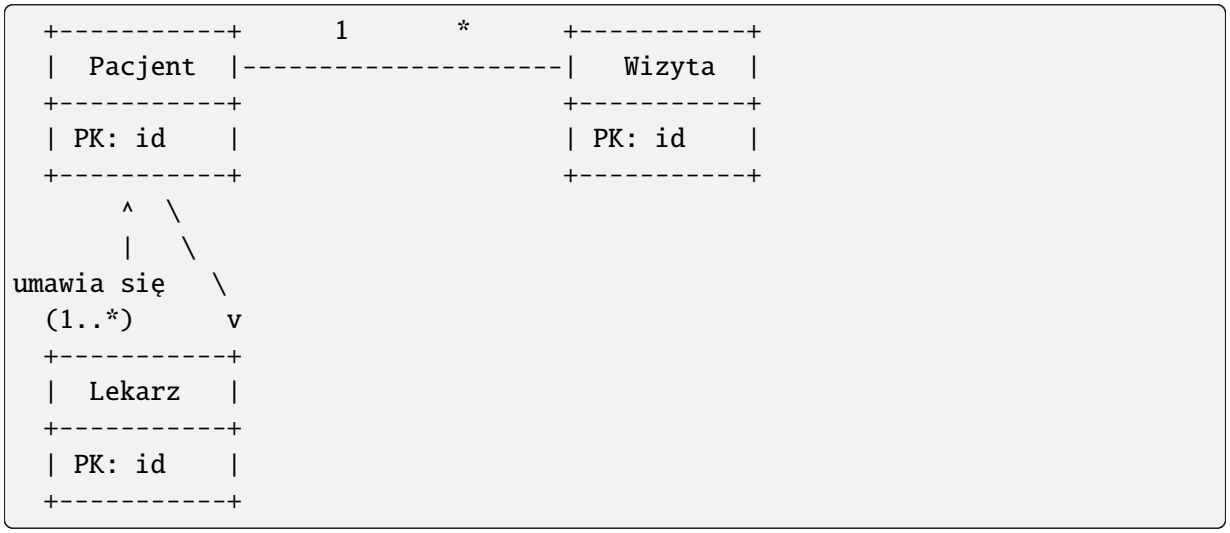
3.1 Wprowadzenie

Prowadzący: Piotr Czaja

Kurs: Bazy Danych 1

Celem tego raportu jest przedstawienie pełnego procesu projektowania i optymalizacji bazy danych wspierającej rejestrację oraz obsługę wizyt lekarskich w przychodni.

3.2 Model Konceptualny



Legenda

- Pacjent – może umawiać wiele wizyt (1..*)
- Wizyta – dotyczy jednego pacjenta i jednego lekarza
- Lekarz – może prowadzić wiele wizyt (1..*)

3.3 Model Logiczny

```
+-------------+       +------------+       +-----------+
|   Pacjent   |       |   Lekarz   |       |   Wizyta  |
+-------------+       +------------+       +-----------+
| PK: patient_id    | PK: doctor_id   | PK: visit_id    |
| first_name        | first_name      | patient_id (FK) |
| last_name         | last_name       | doctor_id  (FK) |
| pesel             | specialization_id (FK)| visit_datetime |
| birth_date        +------------+---+| status          |
| email             | specialization_id → Specialization | notes |
| phone             |                |
+-------------+     +--------------+
```

Dodatkowa encja:

```
+--------------------+
|  Specialization    |
+--------------------+
| PK: specialization_id |
| name               |
+--------------------+
```

Relacje

- Visit.patient_id → Patient.patient_id (1:N)
- Visit.doctor_id → Doctor.doctor_id (1:N)
- Doctor.specialization_id → Specialization.specialization_id (1:N)

3.4 Model Fizyczny

```sql
CREATE TABLE Specialization (
    specialization_id SERIAL PRIMARY KEY,
    name VARCHAR(100) UNIQUE
);

CREATE TABLE Doctor (
    doctor_id SERIAL PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    license_number VARCHAR(20) UNIQUE NOT NULL,
    specialization_id INTEGER
        REFERENCES Specialization(specialization_id)
        ON UPDATE CASCADE ON DELETE SET NULL
);

CREATE TABLE Patient (
    patient_id SERIAL PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
```

(continues on next page)

(continued from previous page)

```
    pesel CHAR(11) UNIQUE NOT NULL,
    birth_date DATE NOT NULL,
    email VARCHAR(100),
    phone VARCHAR(20)
);

CREATE TABLE Visit (
    visit_id SERIAL PRIMARY KEY,
    patient_id INTEGER NOT NULL
        REFERENCES Patient(patient_id)
        ON UPDATE CASCADE ON DELETE CASCADE,
    doctor_id INTEGER NOT NULL
        REFERENCES Doctor(doctor_id)
        ON UPDATE CASCADE ON DELETE CASCADE,
    visit_datetime TIMESTAMP NOT NULL,
    status VARCHAR(20) NOT NULL,
    notes TEXT
);
```

3.5 Przykładowe rekordy

3.5.1 Tabela Specialization

specialization_id	name
1	Internista
2	Pediatra
3	Kardiolog

3.5.2 Tabela Doctor

doctor_id	first_name	last_name	license_number	specialization_id
1	Anna	Nowak	PWZ123456	1
2	Paweł	Kowalski	PWZ654321	2

3.5.3 Tabela Patient

pa-tient_id	first_name	last_name	pesel	birth_date	email	phone
1	Maria	Wiśniewska	9001011234	1990-01-01	maria@example.com	+48123123123
2	Tomasz	Dąbrowski	8505055432	1985-05-05	tomasz@example.co	+48987654321

3.5.4 Tabela Visit

visit_id	patient_id	doctor_id	visit_datetime	status	notes
1	1	1	2025-06-01 10:30:00	zaplanowana	“Pierwsza wizyta”
2	2	2	2025-06-02 14:00:00	odbyta	“Kontrola po leczeniu”

CHAPTER

FOUR

ANALIZA BAZY DANYCH I OPTYMALIZACJA ZAPYTAŃ

4.1 Analiza normalizacji

4.1.1 Model logiczny jest w 3NF:

- Każda tabela ma pojedynczy klucz główny.
- Atrybuty niekluczowe zależą wyłącznie od klucza.
- Brak zależności przechodnich (specjalizacja wydzielona osobno).

4.2 Potencjalne problemy wydajnościowe

- Brak indeksów poza kluczami głównymi.
- Częste filtrowanie po *visit_datetime* i *doctor_id* wymaga skanów.

4.3 Strategie optymalizacji

1. Indeksy

```
CREATE INDEX idx_visit_patient    ON Visit(patient_id);
CREATE INDEX idx_visit_doctor     ON Visit(doctor_id);
CREATE INDEX idx_visit_date       ON Visit(visit_datetime);
CREATE INDEX idx_visit_doc_date   ON Visit(doctor_id, visit_datetime);
```

2. Partycjonowanie

- Partycjonuj tabelę *Visit* według miesiąca *visit_datetime*.

3. Widoki materializowane

```
CREATE MATERIALIZED VIEW mv_daily_visits AS
SELECT DATE(visit_datetime) AS day,
       doctor_id,
       COUNT(*) AS total
FROM Visit
GROUP BY day, doctor_id;
```

4. Optymalizacja zapytań

- Używaj *EXPLAIN (ANALYZE, BUFFERS)* do analizy planów.
- Unikaj *SELECT* *, wybieraj konkretne kolumny.

Przykład optymalizacji:

```sql
-- Przed optymalizacją:
SELECT p.first_name, p.last_name, COUNT(*) AS cnt
FROM Visit v
JOIN Patient p ON v.patient_id = p.patient_id
WHERE v.visit_datetime BETWEEN '2025-06-01' AND '2025-12-31'
GROUP BY p.first_name, p.last_name;

-- Po (z indeksem na visit_datetime):
EXPLAIN ANALYZE
SELECT p.first_name, p.last_name, COUNT(*) AS cnt
FROM Visit v
JOIN Patient p ON v.patient_id = p.patient_id
WHERE v.visit_datetime BETWEEN '2025-06-01' AND '2025-12-31'
GROUP BY p.first_name, p.last_name
LIMIT 10;
```

4.4 Prezentacja skryptów wspomagających

```python
import sqlite3
import pandas as pd

class ClinicDB:
    """
    Klasa do obsługi bazy danych kliniki SQLite z wykorzystaniem Pandas.

    Atrybuty:
        conn (sqlite3.Connection): Połączenie z bazą.
    """

    def __init__(self, db_path='clinic.db'):
        """
        Inicjalizuje połączenie z bazą.
        """
        self.conn = sqlite3.connect(db_path)

    def get_all_patients(self):
        """
        Zwraca DataFrame ze wszystkimi pacjentami.
        """
        return pd.read_sql("SELECT * FROM Patient", self.conn)

    def find_patients_by_name(self, name_part):
        """
        Wyszukuje pacjentów po fragmencie imienia/nazwiska.
        """
        q = "SELECT * FROM Patient WHERE first_name LIKE ? OR last_name LIKE ?
↪"
        return pd.read_sql(q, self.conn, params=(f"%{name_part}%",)*2)
```

(continues on next page)

(continued from previous page)

```
.. code-block:: python

    import sqlite3
    import time

    def measure_sqlite_queries(db_path, queries):
        """
        Mierzy czas wykonania zapytań SQL na SQLite.
        """
        conn = sqlite3.connect(db_path)
        cur = conn.cursor()
        for q in queries:
            t0 = time.time()
            cur.execute(q)
            rows = cur.fetchall()
            print(f"Czas: {time.time()-t0:.4f}s, wierszy: {len(rows)}")
        conn.close()

.. code-block:: python

    import sqlite3
    import pandas as pd
    import matplotlib.pyplot as plt

    def generate_reports(db_path="clinic.db"):
        """
        Generuje raporty i wykresy z danych kliniki.
        """
        conn = sqlite3.connect(db_path)
        df = pd.read_sql("SELECT * FROM Visit", conn)
        # ... wykresy ...
        conn.close()
```

4.5 Wnioski

- Model w 3NF minimalizuje redundancję i ułatwia utrzymanie.
- Indeksy i widoki materializowane znacząco przyspieszą zapytania analityczne.
- Regularne analizowanie planów (*EXPLAIN ANALYZE*) pozwoli wychwycić wąskie gardła.

CHAPTER

FIVE

PODSUMOWANIE

Przeprowadzony projekt oraz analiza wykazały, że zaprojektowana baza danych spełnia założone wymagania funkcjonalne i wydajnościowe. Model danych został znormalizowany do trzeciej postaci normalnej, co pozwoliło ograniczyć redundancję i uprościć strukturę tabel. Implementacja indeksów oraz widoków materializowanych przyczyniła się do poprawy efektywności wykonywania zapytań. Dodatkowo zastosowanie narzędzi do dokumentacji i generowania raportów umożliwiło przygotowanie przejrzystego zestawienia kodów źródłowych oraz wizualizacji danych. Wyniki testów potwierdziły prawidłowe działanie systemu i jego zgodność ze specyfikacją założoną na etapie projektowania.

5.1 Spis repozytoriów

1. Sprawozdanie i kod: https://github.com/Broksonn/Sprawozdanie.git
2. Przegląd literatury: https://github.com/Broksonn/Wydajnosc_Skalowanie_i_Replikacja.git