Sprawozdanie Bazy Danych

Wydanie 1.0.0

Michał Pawlica, Wicenty Wensker

06 lip 2024

Contents

CHAPTER 1

Sprzęt dla bazy danych

1.1 Wstęp

System zarządzania bazami danych (DBMS) jest bardzo ważnym elementem większości nowoczesnych aplikacji i usług. Dwa popularne systemy DBMS to SQLite i PostgreSQL. PostgreSQL jest potężnym systemem zarządzania relacyjnymi bazami danych obiektowymi oraz oferuje wiele zaawansowanych funkcji, takich jak transakcje z pełnym wsparciem dla ACID, subzapytania, wyzwalacze, widoki i przechowywane procedury. Jest również znany ze swojej skalowalności i niezawodności, co czyni go idealnym wyborem dla dużych i złożonych systemów baz danych. SQLite z kolei to lekka baza danych SQL, która jest znana z tego, że nie wymaga skomplikowanego sprzętu ani dużych zasobów systemowych do działania. Jest to idealne rozwiązanie dla aplikacji wbudowanych, systemów operacyjnych o ograniczonych zasobach oraz dla urządzeń mobilnych.

1.2 Sprzęt dla bazy danych PostgreSQL

Wybór odpowiedniego sprzętu dla bazy danych PostgreSQL jest istotny aby osiągnięć optymalną wydajność i niezawodność. Postegresql wymaga odpowiedniego sprzętu aby działał optymalnie. Tworząc bazę danych za pomocą PostegreSQL powinniśmy zaopatrzyć się w sprzęt, który będzie odpowiedni pod PostegreSQL. Poniżej przedstawiono komponenty sprzętowe, które należy wziąć pod uwagę przy konfiguracji serwera dla PostgreSQL:

1.2.1 Procesor (CPU):

Procesor jest sercem każdego serwera i ma ogromne znaczenie dla wydajności bazy danych. PostgreSQL może skorzystać z wielu rdzeni, więc zaleca się wybór procesora z wieloma rdzeniami i wysoką częstotliwością taktowania. Procesory z rodziny Intel Xeon lub AMD EPYC są często wybierane do zastosowań serwerowych ze względu na ich wydajność i niezawodność.

1.2.2 Pamięć Operacyjna (RAM):

Pamięć RAM jest krytycznym zasobem dla PostgreSQL, ponieważ pozwala na przechowywanie aktywnych danych i indeksów w pamięci, co znacznie przyspiesza operacje odczytu i zapisu. Zaleca się posiadanie jak największej ilości pamięci RAM, która jest dostępna dla serwera, z minimalną rekomendacją wynoszącą 16 GB dla małych baz danych, aż do 256 GB lub więcej dla dużych wdrożeń baz danych.

1.2.3 Przestrzeń Dyskowa (Storage):

Wybór odpowiedniego typu i konfiguracji dysków jest istotny dla wydajności bazy danych. Dyski SSD (Solid State Drive) oferują znacznie lepszą wydajność niż tradycyjne dyski HDD, szczególnie w przypadku operacji o losowym dostępie, które są typowe dla baz danych. W przypadku dużych baz danych warto rozważyć zastosowanie rozwiązań RAID w celu zwiększenia niezawodności i wydajności.

1.2.4 Sieć (Networking):

Szybka i stabilna sieć jest niezbędna do zapewnienia komunikacji między serwerem bazy danych a klientami oraz innymi serwerami w klastrze. Zaleca się użycie przynajmniej gigabitowych interfejsów sieciowych, a w przypadku większych wdrożeń rozważenie 10 gigabitowych lub szybszych rozwiązań.

1.2.5 Zasilanie :

Nieprzerwane zasilanie jest bardzo istotne dla zapewnienia ciągłości działania serwera bazy danych. Zasilacz UPS może chronić sprzęt przed skutkami nagłych przerw w dostawie prądu i pozwala na bezpieczne wyłączenie serwera w przypadku dłuższej awarii zasilania.

1.2.6 Chłodzenie (Cooling):

Następną bardzo ważną sprawą przy sprzęcie dla baz danych jest adekwatne chłodzenie, aby zapewnić stabilną pracę komponentów serwera. Wysokie temperatury mogą skracać żywotność sprzętu i prowadzić do przestoju. Zaleca się stosowanie efektywnych systemów chłodzenia, szczególnie w serwerowniach z dużą liczbą urządzeń.

Baza danych PostegreSQL intensywnie korzysta z CPU, a zatem wybór procesora z wieloma rdzeniami i wysoką częstotliwości taktowania, będzie znacznie poprawiać wydajność tworzonej bazy danych. Oprócz tego warto zaopatrzyć się w odpowiednią ilość pamięci RAM, PostegreSQL przechowuje często używane dane w pamięci ze względu na szybszy dostęp do tych danych, co oznacza, że wymaganiem tej bazy danych będzie więcej pamięci RAM. Przechowywane dane znajdują się na dysku twardym, a więc czym więcej mamy pojemności dysku tym większą ilość danych możemy przechować. PostegreSQL nie jest wybredny co do rodzaju dysku, zarówno sprawdzi się dysk twardy HDD jak i SSD. Jednakże dyski SSD oferują znacznie szybsze czasy odczytu i zapisu w porównaniu do tradycyjnych dysków HDD, a to może przekładać się na szybsze zapytania i mniejsze opóźnienia. Szybka i niezawodna sieć jest niezbędna dla baz danych PostgreSQL, szczególnie jeśli są one używane w środowiskach rozproszonych. Ważnym aspektem też jest nieprzerwane zasilanie dla utrzymania ciągłości działania bazy danych i zapobiegania utracie danych. PostgreSQL jest zaawansowanym systemem zarządzania relacyjnymi bazami danych, który wymaga odpowiedniego sprzętu do efektywnego działania. Wybór sprzętu powinien być dostosowany do specyficznych potrzeb i obciążenia, jakie przewiduje się dla bazy danych.

1.3 SQLite

SQLite to lekki system zarządzania relacyjnymi bazami danych, który jest często używany w aplikacjach mobilnych i na komputerach stacjonarnych oraz jest szeroko stosowana w różnych aplikacjach, od urządzeń mobilnych po duże systemy. Ze względu na swoją prostotę i niewielkie wymagania sprzętowe, SQLite może działać na różnych platformach z minimalnymi wymaganiami sprzętowymi. SQLite jest znany z tego, że jest wyjątkowo lekki i może działać na szerokiej gamie sprzętu. Nie wymaga dedykowanego serwera ani skomplikowanej konfiguracji, co sprawia, że jest idealny dla aplikacji o ograniczonych zasobach sprzętowych. Poniżej przedstawiono podstawowe informacje dotyczące SQLite:

-Biblioteka SQLite może być zmniejszona do rozmiaru poniżej 300KiB, co czyni ją wyjątkowo kompaktową w porównaniu do innych systemów baz danych.

-Ma minimalne Zapotrzebowanie na Pamięć: SQLite może działać przy bardzo małym zużyciu pamięci stosu (stack) - około 4KiB oraz niewielkim zużyciu pamięci sterty (heap) - około 100KiB.

-Może działać bez Serwera: SQLite nie wymaga serwera do działania. Baza danych SQLite jest zintegrowana z aplikacją, która uzyskuje dostęp do bazy danych, a aplikacje wchodzą w interakcję bezpośrednio z plikami bazy danych przechowywanymi na dysku. SQLite jest zaprojektowany tak, aby być jak najbardziej kompatybilnym z innymi silnikami baz danych SQL. Dzięki temu, programiści doświadczeni w SQL powinni znaleźć się w dialekcie SQLite gdyż raczej będzie intuicyjny i naturalny. SQLite może pomijać niektóre mniej znane funkcje SQL, ale jego dialekt może zawierać pewne ulepszenia, których nie znajdziemy w niektórych dokumentach standardowych.

Wymagania sprzętowe dla SQLite są dość minimalne, oczywiście im lepszy sprzęt tym lepsze można osiągnąć wydajności bazy danych.

Urządzenia mobilne i wbudowane Dla aplikacji mobilnych i wbudowanych, gdzie zasoby są ograniczone, SQLite jest doskonałym wyborem. Może działać na:

-Smartfonach i tabletach z systemem Android lub iOS.

-Systemach wbudowanych i IoT, takich jak Raspberry Pi czy Arduino.

-Telewizorach Smart TV i systemach rozrywkowych w samochodach.

-Komputery osobiste i serwery

SQLite może być również używany na komputerach osobistych i serwerach, gdzie zasoby nie są tak ograniczone. Może działać na:

-Komputerach z systemem operacyjnym Windows, macOS lub Linux. -Serwerach, które mogą obsługiwać większe obciążenia i przechowywać większe bazy danych.

Chociaż SQLite może działać na różnorodnym sprzęcie, istnieją pewne optymalizacje, które można przeprowadzić, aby poprawić wydajność bazy danych. Warto jednak pomyśleć o doborze odpowiedniego sprzętu w zależności jak dużą bazę danych SQLite chce się stworzyć, warto pomyśleć o tym aby sprzęt miał odpowiednią ilość pamięci zarówno RAM jak i przestrzeni dyskowej. Przy doborze sprzętu do tej bazy danych warto wziąć pod uwagę :

-Pamięć RAM: Im więcej pamięci RAM, tym lepiej, może to pomóc w przyspieszeniu operacji na bazie danych, ponieważ SQLite korzysta z pamięci do przechowywania tymczasowych tabel i buforowania danych.

-Przestrzeń dyskowa: SQLite przechowuje całą bazę danych w jednym pliku, więc ważne jest, aby mieć wystarczającą ilość przestrzeni dyskowej, szczególnie jeśli spodziewamy się wzrostu danych. Warto zwrócić uwagę też na rodzaj dysku, dysk SSD będzie lepiej się sprawował, dzięki temu, że osiąga szybsze wyniki odczytu i zapisu.

-Procesor: Szybszy procesor może poprawić czas odpowiedzi bazy danych, szczególnie przy skomplikowanych zapytaniach i dużych zbiorach danych.

1.3.1 Przykładowe konfiguracje sprzętowe dla aplikacji mobilnych:

Procesor: 4-rdzeniowy RAM: 2 GB Przestrzeń dyskowa: 32 GB Dla aplikacji desktopowych: Procesor: Intel Core i5 lub lepszy RAM: 8 GB Przestrzeń dyskowa: 256 GB SSD Dla serwerów: Procesor: Intel Xeon lub lepszy RAM: 16 GB lub więcej Przestrzeń dyskowa: 1 TB SSD lub więcej

SQLite jest niezwykle elastyczną bazą danych, która może działać na różnorodnym sprzęcie. Wybór odpowiedniego sprzętu zależy od wymagań aplikacji i oczekiwanej skali danych, ale zawsze warto zainwestować w lepszy sprzęt, aby zapewnić płynną pracę i szybką odpowiedź bazy danych.

CHAPTER 2

Konfiguracja bazy danych

Author

- Hubert Albanowski
- Filip Opac

2.1 Lokalizacja i struktura katalogów

2.1.1 Lokalizacja:

1) **Katalog danych:**
   - `/var/lib/postgresql/<wersja>/main` na na systemach Debian/Ubuntu
   - `/var/lib/pgsql/<wersja>/data` na systemach Red Hat/CentOS.
   - Zawiera wszystkie dane, pliki konfiguracyjne, logi i pliki kontrolne.
2) **Katalog Konfiguracyjny:**
   - Pliki konfiguracyjne zwykle znajdują się w katalogu danych, choć zdarza się, że mogą znajdować się w innym katalogu np `/etc/postgresql/<wersja>/main`
3) **Katalog logów:**
   - Domyślnie `/var/log/postgresql` na Debianie/Ubuntu oraz `/var/lib/pgsql/<wersja>/data/pg_log` na Red Har/CentOS
   - zawiera logi PostgreSQL

2.1.2 Struktura katalogów:

- `base/`: Zawiera dane użytkownika dla każdej bazy danych.
- `global/`: Przechowuje dane globalne, np. tabele systemowe.
- `pg_xlog/` lub pg_wal/ (od wersji 10): Zawiera dzienniki Write-Ahead Log (WAL).
- `pg_clog/` lub pg_xact/: Przechowuje dane dotyczące transakcji.
- `pg_tblspc/`: Linki symboliczne do tabel przestrzeni.
- `pg_multixact/`: Dane dotyczące wielokrotnych transakcji.
- `pg_subtrans/`: Dane dotyczące podrzędnych transakcji.
- `pg_stat/`: Dane statystyczne.
- `pg_snapshots/`: Przechowuje dane dotyczące snapshotów.

2.2 Tabele - rozmiar, planowanie i monitorowanie

2.2.1 Rozmiar

Rozmiar tabel jest kluczowym aspektem zarządzania bazą danych, ponieważ bezpośrednio wpływa na wydajność systemu. Duże tabele mogą prowadzić do dłuższych czasów odpowiedzi na zapytania, zwiększonego zużycia pamięci oraz większych obciążeń wejścia/wyjścia.

Monitorowanie rozmiaru tabel PostgreSQL dostarcza kilka narzędzi i zapytań SQL, które umożliwiają monitorowanie rozmiaru tabel:

- **Rozmiar pojedynczej tabeli:**

```
SELECT pg_size_pretty(pg_total_relation_size('nazwa_tabeli')) AS size;
```

To zapytanie zwraca przyjazny dla użytkownika rozmiar całej tabeli, wliczając w to dane, indeksy oraz wszelkie towarzyszące pliki.

- **Rozmiar wszystkich tabel w bazie danych:**

```
SELECT relname AS "Table", pg_size_pretty(pg_total_relation_size(relid)) AS "Size"
FROM pg_catalog.pg_statio_user_tables
ORDER BY pg_total_relation_size(relid) DESC;
```

To zapytanie zwraca listę wszystkich tabel w bazie danych wraz z ich rozmiarami, posortowaną według rozmiaru.

- **Rozmiar indeksów tabeli:**

```
SELECT indexrelname AS "Index", pg_size_pretty(pg_relation_size(indexrelid)) AS "Size"
FROM pg_stat_user_indexes
WHERE relname = 'nazwa_tabeli';
```

To zapytanie zwraca listę indeksów dla określonej tabeli wraz z ich rozmiarami.

2.2.2 Planowanie

Planowanie tabel w PostgreSQL obejmuje kilka kluczowych aspektów, takich jak normalizacja, denormalizacja, indeksowanie oraz partycjonowanie.

1) **Normalizacja** - Normalizacja to proces organizowania danych w bazie danych w taki sposób, aby zminimalizować redundancję i zapewnić integralność danych. Składa się z kilku form normalnych (NF), z których każda eliminuje różne rodzaje redundancji:
    - **Pierwsza forma normalna (1NF)**: Każda komórka tabeli zawiera pojedynczą wartość, a każde pole jest atomowe.
    - **Druga forma normalna (2NF)**: Spełnia wymagania 1NF i zapewnia, że wszystkie kolumny niekluczowe są w pełni zależne od klucza głównego.
    - **Trzecia forma normalna (3NF)**: Spełnia wymagania 2NF i zapewnia, że nie ma zależności przejściowych między kolumnami niekluczowymi.
2) **Denormalizacja** - Denormalizacja to proces łączenia tabel, aby poprawić wydajność zapytań kosztem zwiększenia redundancji danych. Stosuje się ją wtedy, gdy zapytania często wymagają złączeń wielu tabel, co może wpływać na wydajność.
3) **Indeksowanie** - Indeksy są kluczowe dla optymalizacji wydajności zapytań. Pozwalają one na szybsze wyszukiwanie, sortowanie i filtrowanie danych. W PostgreSQL dostępnych jest kilka typów indeksów:
    - **B-tree**: Najczęściej używany typ indeksu, odpowiedni dla większości operacji wyszukiwania.
    - **Hash**: Szybszy dla operacji równościowych, ale mniej wszechstronny niż B-tree.
    - **GIN (Generalized Inverted Index)**: Używany do indeksowania danych wielowartościowych, takich jak tabele JSONB i dokumenty pełnotekstowe.
    - **GiST (Generalized Search Tree)**: Używany dla bardziej złożonych typów danych, takich jak geometria.

Przykład tworzenia indeksu B-tree:

```
CREATE INDEX idx_nazwa_kolumny ON nazwa_tabeli(nazwa_kolumny);
```

4) **Partycjonowanie** - Partycjonowanie polega na podzieleniu tabeli na mniejsze, bardziej zarządzalne części zwane partycjami. Może to znacznie poprawić wydajność zapytań, szczególnie w przypadku dużych tabel. PostgreSQL obsługuje kilka typów partycjonowania:
    - **Partycjonowanie zakresowe (Range Partitioning)**: Dzieli dane na zakresy wartości.
    - **Partycjonowanie listy (List Partitioning)**: Dzieli dane na podstawie wartości z określonej listy.
    - **Partycjonowanie haszowe (Hash Partitioning)**: Dzieli dane na podstawie wartości haszowej.

Przykład tworzenia partycji zakresowej:

```
CREATE TABLE orders (
order_id serial PRIMARY KEY,
order_date date NOT NULL,
customer_id int NOT NULL,
amount numeric
) PARTITION BY RANGE (order_date);
CREATE TABLE orders_2022 PARTITION OF orders
FOR VALUES FROM ('2022-01-01') TO ('2023-01-01');
CREATE TABLE orders_2023 PARTITION OF orders
FOR VALUES FROM ('2023-01-01') TO ('2024-01-01');
```

2.2.3 Monitorowanie

Dodatkowo możlive jest monitorowanie wydajności tabel za pomocą:

- **pg_stat_user_table:**

```
SELECT relname, seq_scan, seq_tup_read, idx_scan, idx_tup_fetch, n_tup_ins, n_tup_upd, n_
↪tup_del
FROM pg_stat_user_tables;
```

- **pg_stat_activity:**

```
SELECT pid, usename, datname, state, query_start, query
FROM pg_stat_activity;
```

2.3 Podstawowe parametry konfiguracyjne

2.3.1 Plik postgresql.conf

Plik *postgresql.conf* zawiera ustawienia dotyczące wydajności, logowania, sieci i wielu innych aspektów.

Kluczowe ustawienia:

1) **Słuchanie połączeń:**

```
listen_addresses = 'localhost'  # Adresy IP, na których PostgreSQL będzie nasłuchiwać␣
↪połączeń
port = 5432                     # Port, na którym PostgreSQL będzie nasłuchiwać połączeń
```

2) **Pamięć i wydajność:**

```
shared_buffers = 128MB           # Ilość pamięci RAM przeznaczona na buforowanie danych
work_mem = 4MB                   # Ilość pamięci RAM na operacje sortowania i agregacji␣
↪na użytkownika
maintenance_work_mem = 64MB      # Ilość pamięci RAM na operacje utrzymaniowe (np.␣
↪VACUUM, CREATE INDEX)
```

3) **Autovacuum:**

```
autovacuum = on                  # Automatyczne czyszczenie i analiza tabel
autovacuum_naptime = 1min        # Częstotliwość uruchamiania procesu autovacuum
```

2.3.2 Plik pg_hba.conf

Plik **pg_hba.conf** odpowiada za kontrolę dostępu do bazy danych PostgreSQL.

Przykład konfiguracji:

```
# TYPE  DATABASE        USER            ADDRESS                 METHOD

# Zezwól lokalnym użytkownikom na połączenie
local   all             all                                     md5

# Zezwól zdalnym użytkownikom z sieci 192.168.1.0/24 na połączenie
host    all             all             192.168.1.0/24          md5
```

2.3.3 Plik pg_ident.conf

Plik **pg_ident.conf** pozwala mapować systemowych użytkowników do użytkowników PostgreSQL.

Przykład konfiguracji:

```
# MAPNAME       SYSTEM-USERNAME         PG-USERNAME

mymap           johndoe                 john
mymap           janedoe                 jane
```

W pliku **pg_hba.conf** można użyć tej mapy:

```
host    all             all             127.0.0.1/32            ident map=mymap
```

CHAPTER 3

Temat 3: Kontrola i Konserwacja

Author

- 272780
- 272763

3.1 Konserwacja bazy danych

Regularne utrzymywanie bazy danych w optymalnym stanie ma kluczowe znaczenie dla jej wydajności. Zapewnienie regularnej konserwacji jest niezbędne do utrzymania systemu na najwyższym poziomie działania. Zaleca się wprowadzenie harmonogramu konserwacji, który obejmie przynajmniej tygodniowe interwały.

3.2 Kontrola danych

Kontrola danych to proces systematycznego zbierania, monitorowania i analizowania informacji w celu uzyskania wglądu, podejmowania świadomych decyzji oraz realizacji określonych celów. Obejmuje to stałe śledzenie i rejestrację istotnych punktów danych związanych z różnymi aspektami działalności firmy lub systemu, takimi jak zachowanie klientów, wskaźniki operacyjne czy trendy rynkowe. Surowe dane są przetwarzane w istotne spostrzeżenia, które mają wpływ na podejmowanie strategicznych decyzji oraz doskonalenie procesów.

3.3 Znaczenie kontroli danych

Nie można przecenić znaczenia kontroli danych w dzisiejszym środowisku biznesowym. W miarę jak firmy operują w coraz bardziej złożonym i konkurencyjnym otoczeniu, wykorzystanie danych poprzez systematyczne monitorowanie przynosi liczne korzyści. Oto kilka kluczowych powodów, dla których zarządzanie danymi jest tak istotne:

- Umożliwia podejmowanie świadomych decyzji.
- Pomaga w ocenie wydajności.
- Stanowi fundament ciągłego doskonalenia dzięki analizie danych.
- Poprawia zrozumienie i postrzeganie klientów.

3.4 Planowanie skutecznej kontroli danych

1. **Określenie celów kontroli danych:** Dokładne określenie celów zbierania danych, czyli jasne zdefiniowanie, co chcemy osiągnąć poprzez analizę gromadzonych informacji. Cele powinny być zgodne z celami biznesowymi i strategicznymi organizacji.

2. **Identyfikacja kluczowych wskaźników:** Skoncentrowanie się na kluczowych wskaźnikach, które bezpośrednio odpowiadają ustalonym celom biznesowym. Kluczowe wskaźniki powinny być mierzalne i umożliwiać monitorowanie postępów w realizacji celów.

3. **Zidentyfikowanie źródeł danych:** Pełna identyfikacja i udokumentowanie wszystkich potencjalnych źródeł danych, obejmujących zarówno wewnętrzne bazy danych, jak i interfejsy API stron trzecich, interakcje z klientami, analitykę witryn internetowych i inne istotne kanały.

4. **Zapewnienie jakości danych:** Ustanowienie standardów i procedur dotyczących jakości danych w celu zapewnienia, że gromadzone dane są dokładne, kompletne i wiarygodne. Wdrożenie kontroli walidacyjnych pozwoli na wczesne wykrywanie ewentualnych błędów.

5. **Wybór narzędzi gromadzenia danych:** Dobór odpowiednich narzędzi gromadzenia danych, które będą skutecznie wspierać proces kontroli danych, uwzględniając specyfikę analizowanych danych i potrzeby organizacji.

6. **Opracowanie strategii bezpieczeństwa danych:** Wypracowanie solidnej strategii bezpieczeństwa danych, uwzględniającej zgodność z obowiązującymi przepisami o ochronie danych osobowych (np. RODO, HIPAA) oraz wdrożenie odpowiednich środków zapewniających ochronę wrażliwych informacji.

7. **Planowanie przechowywania i zarządzania danymi:** Sporządzenie planu przechowywania i zarządzania danymi, uwzględniającego kwestie takie jak skalowalność, dostępność i zasady przechowywania danych, aby zapewnić ich bezpieczne i efektywne zarządzanie.

8. **Integracja danych:** Integracja danych z różnych źródeł w celu stworzenia jednolitego i kompleksowego widoku, umożliwiającego bardziej wszechstronne analizy i wnioskowanie.

9. **Wdrożenie narzędzi do wizualizacji danych i raportowania:** Implementacja narzędzi do wizualizacji danych i generowania raportów, które umożliwią zainteresowanym stronom łatwe zrozumienie danych oraz podejmowanie informowanych decyzji na ich podstawie.

10. **Automatyzacja gromadzenia danych:** Jeśli to możliwe, automatyzacja procesu gromadzenia danych w celu zapewnienia spójności i terminowości, co przyczyni się do efektywniejszego wykorzystania zasobów oraz poprawy jakości analiz.

11. **Przeszkolenie członków zespołu:** Zainwestuj w szkolenia dla członków zespołu, aby posiadali niezbędną wiedzę i umiejętności związane z gromadzeniem, analizowaniem, bezpieczeństwem i interpretacją danych. Regularne szkolenia pozwolą na ciągłe doskonalenie umiejętności zespołu i lepsze wykorzystanie potencjału danych.

12. **Wprowadzenie systemu ciągłej kontroli i konserwacji:** Stwórz system ciągłej kontroli i konserwacji, który obejmuje regularne przeglądy jakości danych, aktualizacje procedur oraz dostosowanie się do zmian technologicznych. Zapewnienie elastyczności i adaptacyjności systemu pozwoli na utrzymanie wydajności i celowości działań w długim okresie czasu.

13. **Implementacja pętli informacji zwrotnej:** Wprowadź mechanizmy pętli informacji zwrotnej, które umożliwią ciągłe doskonalenie procesów kontrolnych. Zachęcaj użytkowników do przekazywania opinii na temat jakości i użyteczności danych oraz wykorzystaj te informacje do doskonalenia procedur i procesów kontroli.

14. **Projektowanie skalowalnego systemu kontroli danych:** Zaprojektuj system kontroli danych w sposób skalowalny, aby mógł elastycznie rosnąć wraz z ilością danych i ich złożonością. Uwzględnij przyszłe potrzeby i zmiany technologiczne, aby infrastruktura była w stanie sprostać rosnącym wymaganiom organizacji.

15. **Przeprowadzanie regularnych audytów:** Regularnie przeprowadzaj audyty, aby upewnić się, że procesy kontroli danych są zgodne z pierwotnymi celami i spełniają wymogi prawne. Audyty pozwalają na identyfikację obszarów wymagających ulepszeń oraz zapewnienie ciągłej zgodności z najnowszymi standardami i regulacjami.

CHAPTER 4

Monitorowanie i Diagnostyka

Author

- Kajetan Parka
- Mikołaj Piłat

1. **Sesje i użytkownicy**: Monitorowanie sesji i użytkowników w administracji baz danych jest kluczowym elementem. Pozwala to na śledzenie aktywności użytkowników, zarządzanie zasobami oraz wykrywanie potencjalnych problemów.

   1.1 **Szczegółowe monitorowanie aktywności użytkowników**: Administratorzy baz danych mogą śledzić, które zapytania są wykonywane przez użytkowników, jak długo trwają te zapytania i jakie są ich wyniki. Można to zrobić za pomocą narzędzi takich jak SQL Profiler (w przypadku SQL Server) lub AWR (w przypadku Oracle). Te narzędzia mogą dostarczyć szczegółowych informacji na temat zapytań, takich jak czas wykonania, zużycie procesora, ilość odczytów i zapisów na dysk, a nawet plan wykonania zapytania. Dodatkowo, administratorzy mogą monitorować aktywność na poziomie sesji, taką jak liczba otwartych transakcji, czas trwania sesji, używane zasoby i inne.

   1.2 **Zarządzanie zasobami na poziomie sesji**: Administratorzy baz danych mogą używać narzędzi do monitorowania zasobów, takich jak Performance Monitor (w przypadku Windows) lub top (w przypadku Linux), aby zobaczyć, które procesy zużywają najwięcej zasobów. Można również skonfigurować limity zasobów dla poszczególnych użytkowników lub sesji, aby zapobiec monopolizowaniu zasobów przez pojedyncze sesje. Administratorzy mogą również monitorować użycie zasobów na poziomie sesji, takie jak zużycie procesora, pamięci, dysku i sieci, co może pomóc w identyfikowaniu i rozwiązywaniu problemów z wydajnością.

   1.3 **Wykrywanie problemów na poziomie sesji**: Administratorzy baz danych mogą używać narzędzi do monitorowania bazy danych, aby wykrywać problemy takie jak blokady, długotrwałe transakcje, czy błędy w zapytaniach. Na przykład, SQL Server oferuje narzędzie o nazwie Activity Monitor, które pokazuje informacje o blokadach, a Oracle oferuje narzędzie o nazwie Automatic Database Diagnostic Monitor (ADDM), które automatycznie wykrywa i diagnozuje problemy z wydajnością. Administratorzy mogą również monitorować logi błędów i ostrzeżeń generowane przez DBMS, co może pomóc w identyfikowaniu i rozwiązywaniu problemów.

   1.4 **Bezpieczeństwo**: Monitorowanie sesji i użytkowników jest również ważne z punktu widzenia bezpieczeństwa. Administratorzy mogą śledzić, kto loguje się do systemu, kiedy to robią i co robią.

Można to zrobić za pomocą logów audytu bazy danych. W przypadku podejrzanej aktywności, takiej jak próby logowania poza normalnymi godzinami pracy, administratorzy mogą podjąć odpowiednie działania, takie jak zmiana hasła użytkownika lub zablokowanie konta.

1.5 **Zgodność**: Monitorowanie sesji i użytkowników jest również ważne z punktu widzenia zgodności z przepisami. Administratorzy mogą używać narzędzi do audytu, takich jak SQL Server Audit lub Oracle Database Audit, aby zobaczyć, kto miał dostęp do jakich danych i kiedy. Te informacje mogą być użyte do udowodnienia zgodności z przepisami dotyczącymi ochrony danych, takimi jak RODO.

2. **Śledzenie dostępu użytkowników do poszczególnych tabel**: Śledzenie dostępu użytkowników do poszczególnych tabel jest istotnym elementem monitorowania i diagnostyki bazy danych. Pozwala to na zrozumienie, jak użytkownicy korzystają z danych, a także na wykrywanie potencjalnych problemów lub podejrzanej aktywności.

   2.1 **Zrozumienie użycia danych**: Administratorzy baz danych mogą używać narzędzi do monitorowania dostępu do tabel, aby zrozumieć, jakie tabele są najczęściej odwiedzane i jakie operacje są najczęściej wykonywane. Na przykład, mogą używać narzędzi do monitorowania wydajności, takich jak SQL Server Profiler lub Oracle Enterprise Manager, aby zobaczyć, które zapytania są najczęściej wykonywane i jak długo trwają. Mogą również używać narzędzi do analizy logów, takich jak Log Miner w Oracle, aby zobaczyć, kiedy i przez kogo dane tabele były modyfikowane. Te narzędzia mogą dostarczyć szczegółowych informacji na temat zapytań, takich jak czas wykonania, zużycie procesora, ilość odczytów i zapisów na dysk, a nawet plan wykonania zapytania.

   2.2 **Wykrywanie problemów**: Jeśli użytkownik nagle zaczyna wykonywać dużą liczbę zapytań do określonej tabeli, może to być oznaką problemu. Na przykład, może to oznaczać, że zapytanie jest niewłaściwie skonstruowane lub że aplikacja jest zapętlona. Administratorzy mogą używać narzędzi do monitorowania wydajności, aby zidentyfikować te problemy i podjąć odpowiednie działania. Narzędzia te mogą dostarczyć szczegółowych informacji na temat zapytań, takich jak czas wykonania, zużycie procesora, ilość odczytów i zapisów na dysk, a nawet plan wykonania zapytania.

   2.3 **Bezpieczeństwo i zgodność**: Śledzenie dostępu do tabel jest również ważne z punktu widzenia bezpieczeństwa. Administratorzy mogą używać narzędzi do audytu, takich jak SQL Server Audit lub Oracle Database Audit, aby zobaczyć, kto miał dostęp do jakich danych i kiedy. Mogą również skonfigurować alerty, które powiadamiają ich o podejrzanej aktywności, takiej jak próby dostępu do danych poza normalnymi godzinami pracy. Wszystko to pomaga w utrzymaniu zgodności z przepisami dotyczącymi ochrony danych, takimi jak RODO.

   2.4 **Analiza trendów**: Administratorzy mogą analizować dane z monitorowania dostępu do tabel, aby zidentyfikować trendy w użyciu danych. Na przykład, mogą zauważyć, że pewne tabele są odwiedzane częściej w określonych godzinach dnia lub dniach tygodnia. Te informacje mogą pomóc w planowaniu przyszłych potrzeb zasobów i optymalizacji systemu.

   2.5 **Optymalizacja wydajności**: Na podstawie informacji uzyskanych z monitorowania dostępu do tabel, administratorzy mogą podjąć działania w celu optymalizacji wydajności systemu. Na przykład, jeśli zauważą, że pewne zapytania są wykonywane bardzo często, mogą zdecydować się na optymalizację tych zapytań, na przykład poprzez dodanie indeksów do odpowiednich tabel.

3. **Błędy dziennika i raporty**: Monitorowanie błędów dziennika i raportów jest kluczowym elementem zarządzania bazą danych. Pozwala to na szybkie wykrywanie i rozwiązywanie problemów, zanim wpłyną one na działanie systemu.

   3.1 **Dzienniki błędów**: Dzienniki błędów są zapisywane przez system zarządzania bazą danych (DBMS) i zawierają informacje o wszelkich błędach, które wystąpiły podczas działania systemu. Mogą zawierać informacje takie jak kod błędu, czas wystąpienia błędu, zapytanie, które spowodowało błąd, i inne szczegóły, które mogą pomóc w diagnozowaniu problemu. Administratorzy baz danych powinni regularnie sprawdzać dzienniki błędów i reagować na wszelkie poważne błędy. Dzienniki błędów mogą również pomóc w identyfikowaniu wzorców błędów, co może pomóc w przyszłych

działaniach prewencyjnych.

3.2 **Raporty**: Raporty są generowane przez narzędzia monitorujące i mogą zawierać informacje na temat wydajności systemu, użycia zasobów, aktywności użytkowników i innych aspektów działania bazy danych. Raporty mogą być generowane na żądanie lub automatycznie w określonych interwałach czasu. Mogą być również konfigurowane do wysyłania powiadomień e-mail lub SMS w przypadku wykrycia określonych warunków, takich jak przekroczenie progu użycia zasobów. Raporty mogą pomóc w identyfikowaniu obszarów, które wymagają uwagi, i mogą dostarczyć cennych informacji do analizy trendów i planowania przyszłości.

3.3 **Analiza i diagnoza**: Dzienniki błędów i raporty są kluczowymi narzędziami do analizy i diagnozy problemów z bazą danych. Na przykład, jeśli system jest wolny, administrator może sprawdzić raporty wydajności, aby zobaczyć, które zapytania są najwolniejsze, a następnie sprawdzić dzienniki błędów, aby zobaczyć, czy te zapytania powodują jakiekolwiek błędy. Na podstawie tych informacji, administrator może podjąć działania w celu optymalizacji zapytań lub zwiększenia dostępnych zasobów.

3.4 **Planowanie i optymalizacja**: Dzienniki błędów i raporty mogą również pomóc w planowaniu przyszłych potrzeb zasobów i optymalizacji systemu. Na przykład, jeśli raporty pokazują, że użycie procesora jest regularnie wysokie, administrator może zdecydować o zwiększeniu liczby rdzeni procesora dostępnych dla systemu. Podobnie, jeśli dzienniki błędów pokazują, że często występują błędy związane z brakiem pamięci, administrator może zdecydować o zwiększeniu dostępnej pamięci.

3.5 **Zabezpieczenia i audyt**: Monitorowanie błędów dziennika i raportów jest również ważne z punktu widzenia bezpieczeństwa. Administratorzy mogą używać narzędzi do audytu, takich jak SQL Server Audit lub Oracle Database Audit, aby zobaczyć, kto miał dostęp do jakich danych i kiedy. Mogą również skonfigurować alerty, które powiadamiają ich o podejrzanej aktywności, takiej jak próby dostępu do danych poza normalnymi godzinami pracy. Wszystko to pomaga w utrzymaniu zgodności z przepisami dotyczącymi ochrony danych, takimi jak RODO.

4. **Monitorowanie na poziomie systemu operacyjnego**: Monitorowanie na poziomie systemu operacyjnego jest kluczowym elementem zarządzania bazą danych. Pozwala to na śledzenie użycia zasobów systemowych, takich jak procesor, pamięć, dysk twardy i sieć, co może pomóc w wykrywaniu i rozwiązywaniu problemów z wydajnością.

   4.1 **iostat** (*Linux*): Narzędzie iostat jest częścią pakietu sysstat w systemach Linux. Umożliwia monitorowanie statystyk wejścia/wyjścia (I/O) dla urządzeń I/O i partycji. Może pomóc w identyfikowaniu problemów z dyskiem twardym, takich jak nadmierne użycie dysku lub długie czasy oczekiwania na I/O. iostat dostarcza szczegółowych informacji na temat obciążenia dysku, takich jak prędkość transferu danych, ilość operacji wejścia/wyjścia na sekundę, średni czas oczekiwania na operację I/O i inne.

   4.2 **htop** (*Linux*): htop to zaawansowany menedżer zadań dla systemów Linux. Wyświetla listę aktualnie działających procesów i umożliwia sortowanie ich według różnych kryteriów, takich jak użycie procesora, pamięci, czasu procesora i innych. Pozwala to na szybkie zidentyfikowanie procesów, które zużywają najwięcej zasobów. htop oferuje również funkcje takie jak wyświetlanie drzewa procesów, wyszukiwanie procesów, filtrowanie procesów według użytkownika i inne.

   4.3 **vmstat** (*Linux*): vmstat to narzędzie, które dostarcza informacji o procesach, pamięci, stronach, blokach wejścia/wyjścia, aktywności procesora i dysku. Jest to przydatne narzędzie do monitorowania wydajności systemu. vmstat dostarcza szczegółowych informacji na temat użycia procesora, pamięci, dysku i sieci, co może pomóc w identyfikowaniu wąskich gardeł i optymalizacji wydajności systemu.

   4.4 **Menedżer zadań** (*Windows*): Menedżer zadań w systemie Windows umożliwia monitorowanie użycia procesora, pamięci, dysku i sieci przez poszczególne procesy i usługi. Może pomóc w identyfikowaniu aplikacji lub procesów, które zużywają nadmierną ilość zasobów. Menedżer zadań oferuje również funkcje takie jak zakończanie procesów, zmiana priorytetu procesów, monitorowanie

użycia sieci i inne.

4.5 **Monitor systemu** (*Windows*): Monitor systemu w systemie Windows jest zaawansowanym narzędziem do monitorowania wydajności systemu. Umożliwia śledzenie wielu różnych wskaźników wydajności, takich jak użycie procesora, pamięci, dysku i sieci, a także statystyki dotyczące systemu plików, bazy danych i innych komponentów systemu. Monitor systemu umożliwia tworzenie niestandardowych zestawów wskaźników wydajności, zapisywanie danych wydajności do plików logów i generowanie raportów wydajności.

5. **Monitorowanie serwera** (*np. Nagios, Grafana*): Monitorowanie serwera jest kluczowym elementem zarządzania infrastrukturą IT. Pozwala na śledzenie stanu serwerów i usług, wykrywanie problemów i analizę wydajności.

   5.1 **Nagios**: Nagios to potężne narzędzie do monitorowania systemów, sieci i infrastruktury. Pozwala na śledzenie stanu serwerów, usług sieciowych, urządzeń sieciowych i innych zasobów IT.

   Nagios oferuje funkcje takie jak:

   5.1.1 **Monitorowanie zasobów**: Nagios może monitorować zużycie procesora, pamięci, dysku, obciążenie sieci i inne metryki zasobów na serwerach i innych urządzeniach. Może dostarczyć szczegółowych informacji na temat użycia zasobów, co może pomóc w identyfikowaniu wąskich gardeł i optymalizacji wydajności systemu.

   5.1.2 **Wykrywanie problemów**: Nagios może automatycznie wykrywać problemy, takie jak awarie serwerów, przeciążenie sieci, brak miejsca na dysku i inne problemy. Może wysyłać powiadomienia e-mail, SMS lub inne, gdy wykryje problem. Może również skonfigurować alerty, które powiadamiają administratora o podejrzanej aktywności, takiej jak niezwykle wysokie zużycie zasobów.

   5.1.3 **Raporty i analizy**: Nagios generuje szczegółowe raporty o stanie infrastruktury IT, które mogą pomóc w analizie wydajności, planowaniu przyszłych potrzeb zasobów i identyfikowaniu obszarów, które wymagają uwagi. Raporty mogą zawierać informacje takie jak historia użycia zasobów, statystyki wydajności, logi błędów i inne.

   5.2 **Grafana**: Grafana to otwarte oprogramowanie do wizualizacji danych, które jest często używane do monitorowania wydajności serwerów i usług.

   Grafana oferuje funkcje takie jak:

   5.2.1 **Wizualizacja danych**: Grafana umożliwia tworzenie interaktywnych wykresów, histogramów, map ciepła i innych wizualizacji danych. Można go używać do wizualizacji metryk takich jak zużycie procesora, pamięci, obciążenie sieci, liczba użytkowników online i inne. Grafana oferuje szeroki zakres opcji personalizacji, co pozwala na tworzenie wysoce szczegółowych i informatywnych wykresów.

   5.2.2 **Integracja z różnymi źródłami danych**: Grafana może pobierać dane z wielu różnych źródeł, takich jak bazy danych SQL, systemy monitorowania jak Prometheus czy Graphite, pliki CSV i inne. Dzięki temu jest to bardzo elastyczne narzędzie, które można dostosować do różnych środowisk i potrzeb.

   5.2.3 **Alerty**: Grafana umożliwia konfigurację alertów, które mogą wysyłać powiadomienia, gdy określone warunki są spełnione. Na przykład, można skonfigurować alert, który wysyła powiadomienie, gdy zużycie procesora na serwerze przekracza określony próg. Alerty mogą być wysyłane za pośrednictwem różnych kanałów, takich jak e-mail, Slack, PagerDuty i inne.

CHAPTER 5

Wydajność, skalowanie i replikacja

5.1 Kontrola i buforowanie połączeń

Author

- Adrian Czubaty
- Kamil Nicoś

Kontrola i buforowanie połączeń z bazą danych to kluczowe aspekty zarządzania bazami danych, które mają na celu optymalizację wydajności i zapewnienie niezawodności.

5.1.1 Kontrola połączeń z bazą danych:

- Zarządzanie pulą połączeń: Technika ta pozwala aplikacjom na utrzymanie puli aktywnych połączeń z bazą danych, które są wielokrotnie używane. Poprzez efektywne zarządzanie pulą połączeń, aplikacje mogą minimalizować koszty związane z nawiązywaniem nowych połączeń, co przyczynia się do zwiększenia wydajności i efektywności systemu.
- Monitorowanie wydajności połączenia: Proces ten obejmuje śledzenie metryk związanych z połączeniem z bazą danych, takich jak czas odpowiedzi, błędy połączenia, ilość przesyłanych danych itp. Regularne monitorowanie tych metryk umożliwia szybkie wykrywanie problemów, co pozwala na ich natychmiastowe rozwiązanie i poprawę ogólnej wydajności systemu.
- Zarządzanie transakcjami: Kontrola nad transakcjami obejmuje precyzyjne określenie, kiedy i w jaki sposób transakcje są przetwarzane w bazie danych. Poprzez skuteczne zarządzanie transakcjami, można zapewnić spójność danych oraz uniknąć konfliktów. Przykładowo, dbając o to, aby operacje w ramach jednej transakcji były wykonywane jako jedna, niepodzielna jednostka pracy, zapewniamy integralność danych.

5.1.2 Buforowanie połączeń z bazą danych:

- Buforowanie zapytań: Technika ta polega na przechowywaniu wyników często używanych zapytań w pamięci podręcznej. Dzięki temu, gdy aplikacja ponownie potrzebuje wyników tego samego zapytania, może je szybko pobrać z pamięci podręcznej, co znacząco przyspiesza czas odpowiedzi i zmniejsza obciążenie bazy danych.
- Buforowanie wyników: Podobnie jak buforowanie zapytań, buforowanie wyników polega na przechowywaniu wyników kosztownych zapytań w pamięci podręcznej na przyszłe użycie. To pozwala uniknąć ponownego przetwarzania zapytań, które wymagają skomplikowanych obliczeń lub dostępu do wielu tabel, co przyczynia się do poprawy wydajności systemu.
- Inwalidacja bufora: Proces inwalidacji bufora polega na usuwaniu danych z pamięci podręcznej, gdy stają się przestarzałe lub nieaktualne. Jest to istotny aspekt zarządzania pamięcią podręczną, który zapewnia, że przechowywane dane są zawsze aktualne. Mechanizmy inwalidacji bufora mogą być zautomatyzowane (np. usuwanie danych po określonym czasie) lub sterowane manualnie przez aplikację

5.2 Indeks i klaster

- Indeks w bazie danych to struktura danych, która poprawia szybkość operacji na tabeli poprzez działanie podobne do indeksu w książce. Zamiast przeszukiwać całą tabelę, aby znaleźć konkretną informację, możemy skorzystać z indeksu, który bezpośrednio wskazuje, gdzie ta informacja się znajduje. W bazie danych indeksy są używane do szybkiego wyszukiwania i dostępu do danych w tabeli poprzez tworzenie indeksów na kolumnach tabeli.
- Klaster w bazie danych to technika przechowywania danych dwóch lub więcej powiązanych tabel w tym samym obszarze dysku. Tabele są powiązane za pomocą kluczy obcych, co umożliwia szybki dostęp do powiązanych danych. Klasterowanie może znacznie poprawić wydajność baz danych, ponieważ dane powiązane są przechowywane blisko siebie na dysku, co redukuje czas potrzebny na przeszukiwanie i dostęp do danych.

5.3 Wydajność

Wydajność bazy danych to kluczowy aspekt zarządzania danymi, który ma bezpośredni wpływ na funkcjonowanie i sukces organizacji. W dobie cyfryzacji i rosnącej zależności od danych, zarządzanie wydajnością baz danych stało się nieodzownym elementem strategii IT. W tym podpunkcie opiszemy sześć kluczowych aspektów wydajności baz danych: czasy odpowiedzi, przepustowość, współbieżność, wykorzystanie zasobów, zapytania N+1 i błędy bazy danych. Parametry wydajności baz danych są podstawowymi wskaźnikami zdrowia systemu baz danych. Monitorowanie tych parametrów i proaktywne zarządzanie nimi jest kluczowe dla utrzymania zdrowia i wydajności bazy danych. Niewidoczne lub niemonitorowane problemy mogą prowadzić do poważnych zakłóceń, takich jak spadek wydajności, potencjalna utrata danych, a nawet awaria systemu, dlatego parametry wydajności nie mogą być ignorowane.

5.3.1 1. Czasy odpowiedzi

Czasy odpowiedzi bazy danych są kluczowym elementem w środowiskach, gdzie szybkie decyzje mają kluczowe znaczenie, jak w usługach finansowych czy sytuacjach awaryjnych. Kilka czynników wpływa na czas odpowiedzi:

- Architektura bazy danych: Fizyczny i logiczny projekt bazy danych, w tym partycjonowanie, indeksowanie i przechowywanie danych, mają istotny wpływ. Odpowiednie partycjonowanie i indeksowanie danych może skrócić czas wyszukiwania, a korzystanie z baz danych w pamięci może znacząco przyspieszyć operacje poprzez uniknięcie dostępu do dysku.

- Topologia i kondycja sieci: W rozproszonych bazach danych, konfiguracja sieci, opóźnienia, przepustowość i utrata pakietów mają znaczący wpływ na czas pobierania i zwracania danych. Optymalizacja sieci i kompresja danych mogą pomóc zminimalizować opóźnienia.
- Równoczesny dostęp i równoważenie obciążenia: Techniki takie jak łączenie połączeń, równoważenie obciążenia i replikacja odczytu mogą równomiernie rozłożyć obciążenie, co przekłada się na optymalne czasy odpowiedzi nawet przy dużym ruchu.

Wydajne czasy odpowiedzi są kluczowe dla operacyjnej wydajności, zadowolenia klientów i wyników finansowych firm. Są one wskaźnikiem kondycji systemu baz danych, wpływającym na przepustowość i skalowalność infrastruktury IT. W sektorach zwiększającej się ilości danych utrzymanie szybkich czasów odpowiedzi może stanowić przewagę konkurencyjną.

5.3.2 2. Przepustowość

Przepustowość bazy danych jest kluczowym wskaźnikiem efektywności systemu w obsłudze danych w określonym czasie. Wysoka przepustowość oznacza zdolność bazy danych do obsłużenia większej liczby żądań lub operacji w sposób szybki i wydajny. Wpływ na przepustowość mają różne czynniki, takie jak:

- Współbieżność: Mechanizmy zarządzania transakcjami i blokowania odgrywają istotną rolę w zapewnieniu integralności danych i zwiększeniu wydajności. Efektywne zarządzanie transakcjami pozwala wielu użytkownikom na jednoczesne operacje bez zakłóceń, co jest istotne w środowiskach o dużym obciążeniu, jak podczas wyprzedaży online. Więcej o współbieżności w kolejnym punkcie.
- Bazy danych NoSQL: Systemy NoSQL korzystają z podejścia opartego na ewentualnej spójności, co pozwala na szybsze operacje zapisu poprzez unikanie oczekiwania na zaktualizowanie wszystkich kopii danych we wszystkich węzłach.
- Dystrybucja danych: Techniki takie jak sharding w NoSQL lub partycjonowanie w bazach SQL pozwalają na podział danych na wiele serwerów lub partycji, co zmniejsza obciążenie poszczególnych elementów systemu i poprawia ogólną zdolność do obsługi dużych ilości operacji.

Warto zauważyć, że odpowiednie zarządzanie współbieżnością, wybór odpowiedniego typu bazy danych oraz efektywna dystrybucja danych mają kluczowe znaczenie dla osiągnięcia wysokiej przepustowości bazy danych.

5.3.3 3. Współbieżność

Współbieżność w bazach danych odnosi się do zdolności systemu do obsługi wielu operacji jednocześnie, co jest kluczowe w środowiskach, gdzie wiele użytkowników lub aplikacji korzysta z bazy danych równocześnie. Parametry wydajności baz danych, takie jak transakcje na sekundę (TPS) i zapytania na sekundę (QPS), mierzą współbieżność bazy danych poprzez liczbę operacji, jakie może obsłużyć w jednostce czasu. Czynniki wpływające pozytywnie na współbieżność to:

- Mechanizmy blokujące: Efektywne zarządzanie blokadami umożliwia uniknięcie rywalizacji między transakcjami, co przyczynia się do płynniejszego działania bazy danych.
- Poziomy izolacji transakcji: Wybór odpowiedniego poziomu izolacji transakcji ma wpływ na dokładność danych i współbieżność. Wyższe poziomy izolacji zapewniają większą dokładność, ale mogą ograniczać współbieżność poprzez blokowanie transakcji.
- Architektura bazy danych: Ogólny projekt bazy danych, zwłaszcza w przypadku rozproszonych baz danych, może wpłynąć na zdolność systemu do obsługi wielu równoczesnych żądań poprzez rozłożenie obciążenia na wiele węzłów.

Wyzwania dla współbieżności obejmują:

- Zakleszczenia (Deadlocki): Sytuacje, w których transakcje blokują się nawzajem, uniemożliwiając kontynuację, co może spowolnić bazę danych.

- Głód zasobów: Kiedy procesy zużywają zbyt dużo zasobów, ograniczając dostępność dla innych procesów i zmniejszając współbieżność.
- Hotspoty danych: Częsty dostęp do tych samych punktów danych może tworzyć wąskie gardła, ograniczając współbieżność poprzez tworzenie kolejek dostępu do zasobów.

5.3.4 4. Wykorzystanie zasobów (CPU, pamięć, I/O dysku)

Wykorzystanie zasobów w środowiskach baz danych ma kluczowy wpływ na wydajność i efektywność operacji obsługi danych. Kilka kluczowych zasobów, takich jak CPU, pamięć i operacje wejścia/wyjścia na dysku, wpływa na działanie bazy danych:

- Użycie CPU: Procesor obsługuje wszystkie obliczenia związane z bazą danych, od wykonywania zapytań po zarządzanie transakcjami. Wysokie użycie procesora może wskazywać na nadmierne obciążenie bazy danych, co może prowadzić do spowolnienia operacji i długich czasów odpowiedzi. Maksymalne obciążenie procesora może również oznaczać, że zapytania nie są zoptymalizowane.
- Wykorzystanie pamięci: Pamięć przechowuje aktywnie używane dane, a jej odpowiednia alokacja ma kluczowe znaczenie dla wydajności bazy danych. Wyczerpanie pamięci RAM i poleganie na pamięci dyskowej może znacząco obniżyć wydajność, co często wynika z wycieków pamięci lub niewłaściwych ustawień.
- Operacje I/O na dysku: Operacje wejścia/wyjścia na dysku obejmują odczytywanie i zapisywanie danych, co ma istotne znaczenie dla przechowywania danych na dłuższy czas. Wysoki poziom operacji I/O na dysku może być objawem nieskutecznych strategii buforowania. Optymalne przechowywanie najczęściej używanych danych w pamięci może przyspieszyć operacje i uniknąć tworzenia wąskich gardeł związanym z dostępem do dysku.

Efektywne zarządzanie zasobami, takimi jak CPU, pamięć i operacje wejścia/wyjścia na dysku, jest kluczowe dla zapewnienia optymalnej wydajności bazy danych i uniknięcia spowolnień czy problemów z operacjami.

5.3.5 5. Zapytania N+1

Problemy z zapytaniami N+1 są powszechną nieefektywnością w aplikacjach korzystających z baz danych, szczególnie tych wykorzystujących narzędzia mapowania obiektowo-relacyjnego (ORM). Problem ten polega na nadmiernym wykonywaniu zapytań do bazy danych, co jest szczególnie zauważalne w przypadku, gdy aplikacja wykonuje dodatkowe zapytania dla każdego powiązanego obiektu po jednym początkowym zapytaniu. Przykładowo, jeśli aplikacja pobiera 10 użytkowników za pomocą jednego zapytania, a następnie wykonuje dodatkowe 10 zapytań dla pobrania profili każdego użytkownika, prowadzi to do łącznie 11 zapytań - co jest problemem zapytań N+1.

Przyczyny problemów z zapytaniami N+1 to:

- Błędna konfiguracja ORM: Narzędzia ORM mają za zadanie ułatwić interakcję z bazą danych, ale nieprawidłowa konfiguracja może prowadzić do nieefektywnych strategii ładowania danych, takich jak „leniwe ładowanie”, które powoduje nadmiarowe zapytania.
- Brak zapytań łączących: Niedostateczne wykorzystanie złączeń SQL może prowadzić do problemów z zapytaniami N+1, gdzie aplikacja pobiera dane fragmentarycznie zamiast łączyć je w jednym zapytaniu.
- Niezoptymalizowane wzorce dostępu do danych: Nieefektywne praktyki kodowania, zwłaszcza te związane z iteracyjnym dostępem do danych, mogą prowadzić do nadmiernego wykonywania zapytań do bazy danych, szczególnie w przypadku pętli, które wyzwalają nowe zapytania dla każdej iteracji.

Rozwiązanie problemów z zapytaniami N+1 wymaga odpowiedniej konfiguracji ORM, wykorzystania złączeń SQL oraz optymalizacji wzorców dostępu do danych, aby uniknąć nadmiernego obciążenia bazą danych i poprawić wydajność aplikacji.

5.3.6 6. Błędy bazy danych

Wskaźniki wydajności bazy danych obejmują również błędy, które mogą wpływać na operacje na bazie danych, od pobierania danych po ich przechowywanie. Błędy te mogą objawiać się jako komunikaty o błędach lub kody, sygnalizujące konkretne problemy w systemie bazy danych. Typowe rodzaje błędów bazy danych to:

- Błędy połączenia: Pojawiają się, gdy aplikacja nie może nawiązać połączenia z bazą danych, co może być spowodowane problemami sieciowymi, błędami w ciągach połączeń lub awarią serwera bazy danych.
- Błędy składni w zapytaniach: Występują, gdy polecenie SQL zawiera błędy składniowe, co powoduje odrzucenie go przez bazę danych, szczególnie w przypadku złożonych zapytań SQL.
- Naruszenia ograniczeń: Bazy danych mają reguły, takie jak klucze obce i unikalne ograniczenia, które mają na celu utrzymanie integralności danych. Naruszenie tych ograniczeń, na przykład próba wstawienia duplikatu w miejscu, gdzie powinny być tylko unikalne wpisy, spowoduje zgłoszenie błędu przez bazę danych.
- Błędy limitu zasobów: Pojawiają się, gdy baza danych przekracza limity dostępnych zasobów, takie jak brak miejsca na dysku, przeciążenie procesora czy brak pamięci. Te błędy mogą znacząco spowolnić lub nawet zatrzymać działanie systemu.
- Błędy uprawnień i zabezpieczeń: Próba wykonania operacji bez odpowiednich uprawnień spowoduje błędy, na przykład brak dostępu do tabeli lub wykonywanie operacji bez wymaganych uprawnień.

Rozpoznanie i rozwiązanie tych błędów jest kluczowe dla zapewnienia stabilności i wydajności bazy danych oraz uniknięcia problemów podczas operacji na danych.

5.4 Skalowanie

Bazy danych SQL nie są tak kosztowne w rozbudowie, jak się powszechnie sądzi. Możliwe jest skalowanie ich wszerz, co przynosi wiele korzyści, zwłaszcza w kontekście analizy danych biznesowych. Firmy coraz bardziej interesują się analizą danych klientów pochodzących z różnych źródeł, co wymaga platform skalowalnych wszerz do przetwarzania dużych ilości danych w czasie rzeczywistym. Istnieje kilka opcji, takich jak bazy NoSQL, NewSQL czy platforma Hadoop, które mogą rozwiązać różne wyzwania związane z przetwarzaniem danych. Wdrożenie rozwiązania skalowalnego wszerz z odpowiednim balansem między pamięcią RAM a nośnikami flash może przynieść istotne korzyści, a nowa generacja skalowalnych baz SQL, takie jak InfiniSQL, ClustrixDB czy F1, pokazuje, że bazy SQL mogą być skalowalne wszerz.

5.4.1 Analityka czasu rzeczywistego:

Analityka czasu rzeczywistego w branży Big Data skupia się obecnie na analizie danych w czasie rzeczywistym, co pozwala firmom uzyskać przewagę konkurencyjną i korzyści biznesowe. Istotnym elementem są skalowalne bazy danych SQL, które umożliwiają przetwarzanie danych operacyjnych w czasie rzeczywistym. Wykorzystanie metod przetwarzania kwerend w pamięci operacyjnej i macierzy dyskowych opartych na nośnikach SSD pozwala osiągnąć wysoką wydajność bez konieczności stosowania specjalistycznych rozwiązań. Firmy takie jak Google czy Facebook udowodniły, że bazy danych SQL są skutecznym narzędziem do przetwarzania danych, co może przyczynić się do ograniczenia kosztów zatrudniania specjalistów. Przykładowo, Google wykorzystuje bazę F1 SQL do usługi Adwords, co ułatwia tworzenie aplikacji do zadań OLTP i OLAP. Facebook również podkreśla znaczenie relacyjnych baz danych w analityce, co przekonało wielu do promowania rozszerzeń Hadoopa umożliwiających integrację z bazami SQL.

5.4.2 Łatwa rozbudowa:

Bazy danych SQL typu scale out umożliwiają łatwą liniową skalowalność poprzez dodawanie nowych węzłów do klastra, nawet w trakcie intensywnego użytkowania. Ta operacja nie wymaga zmian w kodzie, aktualizacji bazy danych ani wymiany sprzętu obsługującego aplikację. Każdy nowy węzeł może przyjmować i przetwarzać transakcje wraz z rozszerzaniem klastra. Istotną cechą tych baz SQL jest możliwość przenoszenia kodu bazy danych do węzłów przechowujących dane, zamiast przenoszenia samych danych. Dzięki temu ogranicza się ilość danych przesyłanych wewnątrz klastra, co prowadzi do zmniejszenia nadmiernego ruchu w klastrze i umożliwia liniową skalowalność bazy danych. Ponadto zapewnia to, że tylko jeden węzeł jest odpowiedzialny za zapis danych w określonym zbiorze, co eliminuje problem konkurencyjnego dostępu do tych samych zasobów. W tradycyjnych bazach danych każde zadanie blokuje obszary danych, co przy dużej liczbie zadań konkurencyjnych prowadzi do spadku wydajności.

5.4.3 Eliminacja wąskich gardeł:

W skalowalnych bazach danych SQL rozwiązano problem logu transakcyjnego, który często stanowił wąskie gardło. W tradycyjnych bazach danych wszystkie przetwarzane rekordy są zapisywane w logu transakcyjnym przed zakończeniem kwerendy. W przypadku błędnej konfiguracji lub awarii może to spowodować nadmierny wzrost logu transakcyjnego, który może przekroczyć rozmiar samej bazy danych. To z kolei prowadzi do spowolnienia operacji zapisu w bazie, nawet w przypadku użycia nośników SSD.

5.4.4 Wysoka dostępność w chmurze:

Organizacje oczekują, że ich aplikacje produkcyjne będą zawsze dostępne, co zapewni ciągłość procesów biznesowych. W przypadku awarii chmury, która może się zdarzyć, istotne jest, aby firma mogła szybko przywrócić działanie bazy danych bez utraty danych. Skalowalne bazy danych SQL posiadają wbudowane funkcje wysokiej dostępności, które zapewniają przechowywanie kilku kopii danych, co minimalizuje ryzyko ich utraty.

5.5 Replikacja

Replikacja danych to metoda duplikowania informacji pomiędzy różnymi serwerami baz danych. Dzięki replikacji możemy: - Zwiększyć skalowalność – obciążenie można rozdzielić między wiele serwerów. Operacje takie jak zapisywanie i aktualizowanie rekordów są wykonywane na jednym serwerze, podczas gdy pobieranie i przeszukiwanie danych są realizowane na innym. - Poprawić bezpieczeństwo – poprzez replikację tworzymy kopię istniejącej bazy danych produkcyjnej. Choć nie zabezpieczy nas to przed operacjami typu DROP TABLE, może to być pomocne w przypadku awarii sprzętowej głównego serwera. - Ułatwić analizę – złożone operacje analityczne, różne przeliczenia i analizy statystyczne mogą być przeprowadzane na dedykowanym serwerze, bez obciążania głównej bazy danych. - Zapewnić separację – możemy udostępnić kopię bazy danych produkcyjnej dla programistów lub testerów, umożliwiając im pracę na kopii bazy.

5.5.1 Mechanizmy replikacji

Replikacja w bazach danych odnosi się do procesu kopiowania i dystrybucji danych i obiektów z jednej bazy danych do innej, a następnie synchronizacji obu baz danych w celu utrzymania ich spójności. Proces ten jest dość prosty. Główny serwer (master) prowadzi dziennik operacji, wykorzystując do tego pliki binarne zwane bin-logami, które zawierają instrukcje wykonane przez mastera. Te pliki są następnie odczytywane przez serwer zapasowy (slave), który wykonuje zapytania zawarte w bin-logach, co skutkuje dodawaniem nowych rekordów do bazy danych. W efekcie powstają dwie identyczne bazy danych. Po ustawieniu replikacji na serwerze master, uruchamiany jest dodatkowy wątek, który ma za zadanie wysyłać bin-logi do serwerów slave. Serwer zapasowy z kolei uruchamia dwa wątki: jeden do odczytu bin-logów i drugi do ich wykonania. - Wątek I/O (Input/Output) - zajmuje się odbieraniem dziennika od serwera głównego i zapisywaniem go w plikach tymczasowych relay-log. - Wątek SQL - parsuje te pliki i wykonuje zapytania do bazy

danych. W skrócie, mechanizm replikacji MySQL polega na tym, że serwer główny rejestruje swoje działania, a serwer zapasowy odtwarza te działania, tworząc kopię bazy danych.

5.5.2 Oprogramowanie i zaimplementowane mechanizmy replikacji

- Replikacja oparta na zapisie (Write-Ahead Logging): Ten mechanizm jest powszechnie stosowany w systemach baz danych, takich jak PostgreSQL. Polega na rejestrowaniu transakcji w dzienniku zapisu przed ich zastosowaniem, a następnie replikacji dziennika na serwery repliki.
- Replikacja oparta na zrzutach (Snapshot-Based Replication): Systemy, takie jak Apache Cassandra, wykorzystują replikację opartą na zrzutach. Polega to na tworzeniu zrzutów stanu bazy danych w określonych odstępach czasu i replikacji ich na serwery repliki.
- Replikacja oparta na transakcjach (Transaction-Based Replication): W tym modelu każda transakcja jest replikowana na serwery repliki, co jest przydatne w systemach wymagających silnej spójności, np. Google Spanner.
- Replikacja asynchroniczna i synchroniczna: W replikacji asynchronicznej dane są najpierw zapisywane do głównej bazy danych, a następnie replikowane na serwery repliki. W replikacji synchronicznej dane są zapisywane jednocześnie do głównej bazy danych i serwerów repliki.
- Replikacja dwukierunkowa (Bi-Directional Replication): Pozwala na wprowadzanie zmian na dowolnym serwerze repliki, które są następnie replikowane na pozostałe serwery, co jest przydatne w przypadku wysokiej dostępności i tolerancji na awarie.

PostgreSQL oferuje różne typy replikacji, w tym replikację opartą na zapisie (Write-Ahead Logging), replikację asynchroniczną i synchroniczną oraz replikację logiczną. Replikacja oparta na zapisie (WAL) gwarantuje odporność na awarie poprzez zapisywanie zmian w bazie danych do dziennika zapisu przed ich zastosowaniem, który jest replikowany na serwery repliki. PostgreSQL obsługuje zarówno replikację asynchroniczną, gdzie dane są najpierw zapisywane do głównej bazy danych, a następnie replikowane, jak i replikację synchroniczną, gdzie dane są zapisywane jednocześnie do głównej bazy danych i serwerów repliki. Dodatkowo, replikacja logiczna pozwala na replikację wybranych tabel lub baz danych zamiast całego klastra, co jest przydatne zwłaszcza w przypadku dużych baz danych, gdzie replikacja całego klastra byłaby nieefektywna.

5.5.3 Plusy i minusy replikacji

Plusy:

- Poprawa wydajności i dostępności: Replikacja danych pozwala na dystrybucję obciążenia zapytań pomiędzy wiele serwerów, co zwiększa wydajność systemu. Użytkownicy mogą wysyłać zapytania do najbliższego serwera repliki, co skraca czas odpowiedzi. Ponadto, jeśli jeden serwer ulegnie awarii, inne serwery repliki mogą nadal obsługiwać zapytania, co zwiększa dostępność systemu.
- Bezpieczeństwo danych: Replikacja jest również kluczowym elementem strategii tworzenia kopii zapasowych i odzyskiwania danych. Jeśli główna baza danych ulegnie awarii, replika może zostać użyta do przywrócenia danych.
- Dystrybucja danych: Replikacja umożliwia dystrybucję danych do oddzielnych lokalizacji geograficznych. Na przykład, globalna firma może chcieć replikować dane między swoimi lokalizacjami w różnych krajach, aby lokalni użytkownicy mogli szybko i łatwo uzyskać dostęp do potrzebnych informacji.
- Analiza i raportowanie: Repliki danych mogą być używane do celów analitycznych i raportowych. Dzięki temu operacje te nie wpływają na wydajność głównej bazy danych obsługującej transakcje.

Minusy:

- Nie daje nam pewności, że po przeprowadzeniu operacji, dane z serwera głównego będą poprawnie przeniesione na serwer zapasowy

- Nie zapewnia ochrony przed operacjami niosącymi poważne konsekwencje - takimi jak DROP TABLE

5.6 Limity systemu oraz ograniczanie dostępu użytkowników

Limity systemu w zarządzaniu bazami danych odnoszą się do maksymalnej liczby zasobów, które system może obsłużyć. Te limity są zwykle określane przez system zarządzania bazą danych (DBMS) i są zdefiniowane na podstawie dostępnych zasobów sprzętowych i ustawień konfiguracyjnych. Na przykład, w Azure SQL Database istnieją specyficzne limity zasobów dla różnych poziomów cenowych dla pojedynczych baz danych. W MySQL, efektywny maksymalny rozmiar tabeli dla baz danych MySQL jest zazwyczaj określany przez ograniczenia systemu operacyjnego na rozmiary plików, a nie przez wewnętrzne limity MySQL. Ograniczanie dostępu użytkowników w DBMS odnosi się do mechanizmu, który umożliwia lub zabrania użytkownikom dostęp do danych. Składa się z dwóch głównych komponentów: uwierzytelniania i autoryzacji. Uwierzytelnianie to sposób potwierdzenia tożsamości osoby, która próbuje uzyskać dostęp do bazy danych. Autoryzacja natomiast określa, czy poziom dostępu użytkownika jest odpowiedni. Istnieją różne modele kontroli dostępu, takie jak Kontrola Dostępu Uzależniona (DAC), Kontrola Dostępu Obowiązkowa (MAC), Kontrola Dostępu na Podstawie Roli (RBAC) i Kontrola Dostępu na Podstawie Atrybutów (ABAC). W PostgreSQL również istnieją mechanizmy do zarządzania limitami systemu oraz ograniczania dostępu użytkowników. PostgreSQL umożliwia administratorom określenie różnych parametrów konfiguracyjnych, takich jak maksymalna ilość połączeń, pamięć dostępna dla zapytań, maksymalny rozmiar pliku danych, czy maksymalny rozmiar tabeli. Te limity mogą być dostosowywane do potrzeb konkretnego środowiska i obciążenia. W kwestii ograniczania dostępu użytkowników, PostgreSQL oferuje zaawansowane mechanizmy uwierzytelniania i autoryzacji. Można definiować różne role użytkowników, nadawać im odpowiednie uprawnienia do baz danych, schematów, tabel czy nawet poszczególnych kolumn. PostgreSQL obsługuje zarówno uwierzytelnianie oparte na hasłach, jak i uwierzytelnianie oparte na certyfikatach SSL. Dzięki tym funkcjom, administratorzy baz danych mogą skutecznie kontrolować dostęp do danych, zapewniając bezpieczeństwo i poufność informacji przechowywanych w PostgreSQL.

5.7 Testy wydajności sprzętu (pamięć, procesor, dyski) na poziomie systemu operacyjnego

Testy wydajności sprzętu na poziomie systemu operacyjnego są kluczowe dla optymalizacji wydajności baz danych. Obejmują one testy pamięci (RAM), procesora (CPU) oraz dysków (HDD/SSD), które są kluczowymi komponentami sprzętowymi wpływającymi na wydajność systemu.

Testy pamięci (RAM) oceniają szybkość i efektywność pamięci RAM komputera, co ma bezpośredni wpływ na wydajność bazy danych. Narzędzia takie jak MemTest86 mogą być używane do przeprowadzania tych testów.

Testy procesora (CPU) oceniają wydajność jednostki centralnej procesora, która jest kluczowa dla szybkości przetwarzania zapytań do bazy danych. Narzędzia takie jak Cinebench R23 mogą być używane do przeprowadzania tych testów.

Testy dysków (HDD/SSD) oceniają szybkość odczytu i zapisu na dyskach, co ma istotne znaczenie dla wydajności bazy danych, ponieważ dane są przechowywane na dyskach. Narzędzia takie jak CrystalDiskMark 8 i Acronis Drive Monitor mogą być używane do przeprowadzania tych testów.

Wyniki tych testów na poziomie systemu operacyjnego mogą pomóc zidentyfikować obszary, które wymagają ulepszeń sprzętowych, aby zwiększyć wydajność bazy danych, niezależnie od konkretnego oprogramowania bazy danych.

CHAPTER 6

Bezpieczeństwo

Authors

- Dawid Łapiński
- Leon Woźniak

6.1 pg_hba.conf - Sterowanie Dostępem

Plik `pg_hba.conf` (PostgreSQL Host-Based Authentication) jest kluczowym elementem zarządzania bezpieczeństwem w PostgreSQL. Umożliwia on kontrolę dostępu do serwera bazy danych na podstawie adresów IP, typów połączeń oraz mechanizmów uwierzytelniania.

6.1.1 Struktura pliku pg_hba.conf

Plik `pg_hba.conf` składa się z linii, z których każda definiuje regułę dostępu. Każda linia zawiera następujące pola:

- **typ połączenia** - określa, czy połączenie jest lokalne (`local`), czy zdalne (`host`).
- **baza danych** - nazwa bazy danych, do której dostęp jest kontrolowany.
- **użytkownik** - nazwa użytkownika bazy danych.
- **adres** - adres IP klienta (dla połączeń zdalnych).
- **metoda uwierzytelniania** - określa mechanizm uwierzytelniania, który ma być używany.

Przykład reguły w pliku `pg_hba.conf`:

```
# Typ    Baza danych    Użytkownik    Adres              Metoda
host     all            all           192.168.1.0/24     md5
```

6.1.2 Mechanizmy Dostępu

Plik `pg_hba.conf` definiuje kilka mechanizmów uwierzytelniania, takich jak:

- **trust** - pozwala na dostęp bez uwierzytelniania. Jest to opcja najmniej bezpieczna i powinna być używana tylko w wyjątkowych przypadkach.
- **md5** - wykorzystuje hasła zaszyfrowane algorytmem MD5. Jest to standardowy mechanizm uwierzytelniania.
- **password** - wymaga podania hasła w postaci nieszyfrowanej. Jest to mniej bezpieczne niż `md5`.
- **peer** - umożliwia uwierzytelnianie na podstawie identyfikatora systemowego użytkownika.
- **cert** - wykorzystuje certyfikaty SSL do uwierzytelniania.
- **scram-sha-256** - nowoczesny i bezpieczny mechanizm uwierzytelniania, który wykorzystuje algorytm SCRAM-SHA-256.

6.1.3 Konsekwencje Wyboru Mechanizmu Dostępu

Wybór odpowiedniego mechanizmu uwierzytelniania ma bezpośredni wpływ na bezpieczeństwo systemu. Mechanizmy takie jak `trust` mogą znacznie obniżyć poziom bezpieczeństwa, podczas gdy `cert` w połączeniu z SSL zapewnia wysoki poziom ochrony danych. Warto dokładnie analizować potrzeby i ryzyka związane z każdym mechanizmem uwierzytelniania.

Na przykład, stosowanie mechanizmu `trust` może być akceptowalne w przypadku baz danych używanych wyłącznie w środowisku testowym, gdzie bezpieczeństwo nie jest priorytetem. Natomiast w środowisku produkcyjnym, gdzie przetwarzane są dane wrażliwe, konieczne jest użycie bardziej zaawansowanych mechanizmów, takich jak `md5`, `scram-sha-256` lub `cert`.

6.1.4 Przykłady Konfiguracji

Przykładowe konfiguracje pliku `pg_hba.conf` mogą obejmować różne scenariusze dostępu:

```
Pozwól na połączenie dowolnego użytkownika z hostem 192.168.12.10 do bazy danych
↪"postgres", jeśli hasło
użytkownika jest poprawnie podane
# Typ   Baza danych    Użytkownik    Adres              Metoda
host    postgres       all           192.168.12.10/32   scram-sha-256

Pozwól dowolnemu użytkownikowi na lokalnym systemie łączyć się z dowolną bazą danych
używając dowolnej nazwy użytkownika bazy danych za pomocą gniazd Unix (domyślnie dla␣
↪połączeń lokalnych)
# Typ   Baza danych    Użytkownik    Adres              Metoda
local   all            all                              trust

Pozwól na połączenie używając wyrażenia regularnego dla DATABASE, które pozwala na␣
↪połączenie z bazą danych db1, db2
oraz dowolnymi bazami danych o nazwie zaczynającej się od "db" i kończącej się liczbą␣
↪składającą się z dwóch do
czterech cyfr (np. "db1234" lub "db12").
# Typ   Baza danych               Użytkownik    Adres          Metoda
local   db1,"/^db\d{2,4}$",db2    all           localhost      trust
```

6.2 Uprawnienia Użytkownika

Uprawnienia użytkownika w PostgreSQL są zarządzane na kilku poziomach: systemu zarządzania bazą danych (DBMS), poszczególnych baz danych oraz tabel.

6.2.1 Poziom DBMS

Na poziomie DBMS uprawnienia mogą obejmować możliwość tworzenia nowych baz danych, zarządzanie użytkownikami oraz konfigurację systemu. Przykładowe polecenia to:

```
GRANT CREATE ON DATABASE dbname TO username;
REVOKE CREATE ON DATABASE dbname FROM username;
```

Administratorzy bazy danych (DBA) mają pełne uprawnienia na poziomie DBMS, co pozwala im na zarządzanie wszystkimi aspektami działania systemu PostgreSQL. Uprawnienia te mogą obejmować:

- Tworzenie i usuwanie baz danych.
- Tworzenie i zarządzanie użytkownikami oraz rolami.
- Konfigurację parametrów systemowych i optymalizację działania bazy danych.

6.2.2 Poziom Bazy Danych

Na poziomie bazy danych uprawnienia mogą obejmować dostęp do danych, modyfikację struktur oraz wykonywanie operacji administracyjnych. Polecenia zarządzające uprawnieniami to:

```
GRANT ALL PRIVILEGES ON DATABASE dbname TO username;
REVOKE CONNECT ON DATABASE dbname FROM username;
```

Uprawnienia na poziomie bazy danych mogą być szczegółowo dostosowane do potrzeb poszczególnych użytkowników lub grup użytkowników (ról). Na przykład, można przyznać uprawnienia do:

- Łączenia się z bazą danych (`CONNECT`).
- Tworzenia nowych schematów (`CREATE`).
- Wykonywania zapytań (`SELECT`) i modyfikacji danych (`INSERT`, `UPDATE`, `DELETE`).
- Zarządzania tabelami i innymi obiektami bazy danych (`ALTER`, `DROP`).

6.2.3 Poziom Tabeli

Na poziomie tabeli uprawnienia mogą obejmować selekcję, wstawianie, aktualizację oraz usuwanie danych. Przykładowe polecenia to:

```
GRANT SELECT, INSERT ON TABLE tablename TO username;
REVOKE UPDATE ON TABLE tablename FROM username;
```

Precyzyjne zarządzanie uprawnieniami na poziomie tabeli pozwala na ochronę danych przed nieautoryzowanym dostępem oraz modyfikacją. Przykłady uprawnień obejmują:

- `SELECT` - możliwość odczytu danych z tabeli.
- `INSERT` - możliwość dodawania nowych rekordów do tabeli.

- `UPDATE` - możliwość modyfikowania istniejących rekordów.
- `DELETE` - możliwość usuwania rekordów.

6.2.4 Role i Grupy Użytkowników

PostgreSQL umożliwia tworzenie ról i grup użytkowników, co upraszcza zarządzanie uprawnieniami. Role mogą mieć przypisane uprawnienia, które są dziedziczone przez użytkowników przypisanych do tych ról. Przykładowe polecenia:

```
CREATE ROLE read_only;
GRANT SELECT ON ALL TABLES IN SCHEMA public TO read_only;
GRANT read_only TO username;
```

Stosowanie ról i grup użytkowników pozwala na bardziej elastyczne i skalowalne zarządzanie uprawnieniami. Na przykład, można stworzyć rolę `read_only`, która ma tylko uprawnienia do odczytu danych, a następnie przypisać tę rolę wielu użytkownikom, co znacznie upraszcza administrację.

6.3 Zarządzanie Użytkownikami a Dane Wprowadzone

Zarządzanie użytkownikami w PostgreSQL obejmuje tworzenie, modyfikowanie i usuwanie użytkowników oraz ról. Ważnym aspektem jest zarządzanie danymi wprowadzonymi przez użytkowników, szczególnie w kontekście usuwania użytkowników.

6.3.1 Tworzenie i Modyfikowanie Użytkowników

Tworzenie nowych użytkowników w PostgreSQL odbywa się za pomocą polecenia `CREATE USER`. Przykład:

```
CREATE USER username WITH PASSWORD 'password';
```

Modyfikowanie istniejących użytkowników można przeprowadzać za pomocą polecenia `ALTER USER`:

```
ALTER USER username WITH PASSWORD 'new_password';
```

6.3.2 Usuwanie Użytkowników

Usuwanie użytkowników w PostgreSQL odbywa się za pomocą polecenia `DROP USER`. Przykład:

```
DROP USER username;
```

Jednakże usunięcie użytkownika nie powoduje automatycznego usunięcia danych, które zostały przez niego wprowadzone. Dane te pozostają w bazie danych i mogą być dalej dostępne dla innych użytkowników z odpowiednimi uprawnieniami.

6.3.3 Zachowanie Danych po Usunięciu Użytkownika

Dane wprowadzone przez usuniętego użytkownika pozostają w bazie danych, co jest ważne dla zapewnienia integralności i ciągłości danych. W praktyce oznacza to, że:

- Rekordy w tabelach nadal istnieją i są dostępne dla innych użytkowników z odpowiednimi uprawnieniami.
- Metadane, takie jak informacje o autorze danych, mogą być zachowane w celach audytowych.

Przykłady scenariuszy, w których zachowanie danych po usunięciu użytkownika jest istotne:

- **Zmiany kadrowe** - gdy pracownik odchodzi z firmy, jego dane powinny pozostać w systemie.
- **Reorganizacja projektów** - dane wprowadzone przez użytkownika mogą być ważne dla trwających projektów.
- **Naruszenia bezpieczeństwa** - w przypadku konieczności szybkiego usunięcia użytkownika, dane pozostają nienaruszone.

6.3.4 Polityki Retencji Danych

Organizacje mogą wdrażać polityki retencji danych, które określają, jak długo dane wprowadzone przez użytkowników są przechowywane oraz w jakich warunkach mogą być usuwane. Polityki te mogą obejmować:

- Automatyczne usuwanie danych po określonym czasie.
- Przeglądy i audyty danych w celu określenia ich dalszej przydatności.
- Mechanizmy archiwizacji danych w celu ich późniejszego odzyskania, jeśli zajdzie taka potrzeba.

6.4 Zabezpieczenie Połączenia przez SSL/TLS

SSL (Secure Sockets Layer) oraz TLS (Transport Layer Security) są standardowymi technologiami zabezpieczającymi połączenia sieciowe, w tym również połączenia z bazą danych PostgreSQL.

6.4.1 Konfiguracja SSL/TLS

Aby włączyć SSL/TLS w PostgreSQL, należy skonfigurować plik `postgresql.conf` oraz odpowiednio dostosować plik `pg_hba.conf`. Przykład konfiguracji:

```
# postgresql.conf
ssl = on
ssl_cert_file = 'server.crt'
ssl_key_file = 'server.key'
```

Dodatkowo, w pliku `pg_hba.conf` należy zdefiniować reguły uwierzytelniania z użyciem certyfikatów SSL:

```
# pg_hba.conf
hostssl all all 0.0.0.0/0 cert
```

6.4.2 Tworzenie i Zarządzanie Certyfikatami

Do korzystania z SSL/TLS konieczne jest posiadanie certyfikatu serwera oraz klucza prywatnego. Certyfikaty te mogą być wydawane przez zaufane urzędy certyfikacji (CA) lub generowane samodzielnie (self-signed). Przykładowe polecenia do generowania własnych certyfikatów:

```
openssl genrsa -des3 -out server.key 2048
openssl req -new -key server.key -out server.csr
openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt
```

6.4.3 Korzyści z SSL/TLS

SSL/TLS zapewnia szyfrowanie danych przesyłanych między klientem a serwerem, co chroni przed podsłuchiwaniem oraz modyfikowaniem danych podczas transmisji. Zapewnia również uwierzytelnienie serwera oraz, opcjonalnie, klienta, co zwiększa bezpieczeństwo całego systemu.

Korzyści z używania SSL/TLS obejmują:

- Ochronę danych wrażliwych podczas transmisji przez sieć.
- Zapobieganie atakom typu man-in-the-middle, które polegają na przechwytywaniu i modyfikacji danych.
- Uwierzytelnianie serwera, co pozwala klientom na weryfikację, że łączą się z właściwym serwerem.

6.4.4 Monitorowanie i Audyt Połączeń SSL/TLS

Ważnym aspektem korzystania z SSL/TLS jest monitorowanie i audyt połączeń zabezpieczonych. PostgreSQL oferuje mechanizmy logowania, które mogą rejestrować informacje o połączeniach SSL/TLS, co pozwala na:

- Identyfikację prób nieautoryzowanego dostępu.
- Analizę i diagnostykę problemów z połączeniami.
- Zapewnienie zgodności z politykami bezpieczeństwa organizacji.

6.5 Szyfrowanie Danych

Szyfrowanie danych w PostgreSQL może odbywać się zarówno na poziomie transmisji danych, jak i na poziomie przechowywania danych.

6.5.1 Szyfrowanie w Transmisji

Jak wspomniano wcześniej, SSL/TLS umożliwia szyfrowanie danych podczas transmisji między klientem a serwerem, co zapobiega nieautoryzowanemu dostępowi do danych w trakcie ich przesyłania.

6.5.2 Szyfrowanie na Poziomie Dysku

PostgreSQL nie posiada natywnego wsparcia dla szyfrowania danych na poziomie tabel lub baz danych, jednak możliwe jest wykorzystanie zewnętrznych narzędzi i systemów plików szyfrujących. Przykładem może być system plików z szyfrowaniem (np. LUKS w systemach Linux) lub szyfrowanie oferowane przez rozwiązania chmurowe (np. Amazon RDS).

Przykładowa konfiguracja szyfrowania dysku na systemie Linux z użyciem LUKS:

```
sudo cryptsetup luksFormat /dev/sdX
sudo cryptsetup luksOpen /dev/sdX encrypted_disk
sudo mkfs.ext4 /dev/mapper/encrypted_disk
sudo mount /dev/mapper/encrypted_disk /mnt/encrypted
```

6.5.3 Szyfrowanie na Poziomie Aplikacji

Innym podejściem do szyfrowania danych jest szyfrowanie na poziomie aplikacji, gdzie dane są szyfrowane przed zapisaniem do bazy danych i odszyfrowywane po ich odczytaniu. Takie podejście zapewnia pełną kontrolę nad procesem szyfrowania, jednak wymaga dodatkowej implementacji w kodzie aplikacji.

Przykładowe biblioteki do szyfrowania danych na poziomie aplikacji:

- **Python** - `cryptography`, `pycryptodome`.
- **Java** - `javax.crypto`, `Bouncy Castle`.
- **JavaScript** - `crypto`, `sjcl`.

6.5.4 Zarządzanie Kluczami Szyfrującymi

Kluczowym elementem skutecznego szyfrowania danych jest zarządzanie kluczami szyfrującymi. Klucze muszą być bezpiecznie przechowywane i zarządzane, aby zapobiec ich utracie lub kradzieży. Przykładowe narzędzia do zarządzania kluczami:

- **HashiCorp Vault** - bezpieczne przechowywanie i zarządzanie tajemnicami oraz kluczami szyfrującymi.
- **AWS Key Management Service (KMS)** - zarządzanie kluczami w środowisku chmurowym Amazon Web Services.
- **GCP Cloud KMS** - zarządzanie kluczami w środowisku Google Cloud Platform.

CHAPTER 7

Aplikacja na koniec laboratorium

Author

- Wincenty Wensker
- Michał Pawlica

7.1 Wprowadzenie

Celem projektu było zaprojektowanie i implementacja bazy danych w systemie SQLite oraz PostgreSQL, która przechowuje informacje o zawodnikach oraz kategoriach sportowych, do których należą. Projekt obejmował stworzenie odpowiednich tabel, zdefiniowanie relacji między nimi oraz implementację podstawowych operacji na danych, takich jak dodawanie, modyfikowanie, usuwanie i odczytywanie rekordów.

7.2 Założenia co do bazy danych

Baza danych ma służyć jako archiwum wybranych sportowców do małego magazynu z rankingiem sportowców. Pozwala na zapis i modyfikacje sportowca oraz wszystkich kategorii w których sportowiec brał udział, liczbę zdobytych medali wraz z datą rozpoczęcia i jeżeli zawodnik jest już na emeryturze końca kariery. Baza dodatkowo pozwala na usuwanie sportowców i kategorii choć ta operacja jako że działamy na archiwum wedle założeń nie powinna nigdy być przeprowadzona. Baza pozwala na zapis wszystkich danych w plikach csv oraz stosuje symbol zastępczy dla braku daty emerytury w postaci daty 2222-10-31. Oprócz bazy powstały dwie aplikacje. Interfejs służący do wprowadzania zebranych danych oraz analizator to podglądania z wykresami oraz generowania wykresów porównujące poszczególne kraje. Dodatkowo do bazy danych powstała seria funkcji w języku python pozwalających na szybkie tworzenie skryptów wykonujących operacje na bazie danych.

7.3 Projekt Modelu Koncepcyjnego

Model koncepcyjny bazy danych jest abstrakcyjnym przedstawieniem danych oraz związku między nimi. W naszym przypadku baza danych składa się z dwóch tabel: Zawodnicy oraz Kategorie.

- **Zawodnicy:**
  - key: unikalny identyfikator zawodnika.
  - Imie: imię zawodnika.
  - Nazwisko: nazwisko zawodnika.
  - Pseudonil: pseudonim zawodnika.
  - Plec: płeć zawodnika.
  - Narodowsc: narodowość zawodnika.
- **Kategorie:**
  - key: unikalny identyfikator kategorii.
  - NazwaKategorii: nazwa kategorii sportowej.
  - NumerZawodnika: numer zawodnika, klucz obcy odnoszący się do klucza w tabeli Zawodnicy.
  - LiczbaZlotychMedali: liczba złotych medali zdobytych przez zawodnika.
  - LiczbaSrebnychMedali: liczba srebrnych medali zdobytych przez zawodnika.
  - LiczbaBrazowychMedali: liczba brązowych medali zdobytych przez zawodnika.
  - CzyAktynwy: informacja, czy zawodnik jest aktywny sportowo.
  - DataEmerytura: (Jeżeli zawodnik nieaktywny)data przejścia na emeryturę.
  - DataStart: data rozpoczęcia kariery sportowej.

Związek między tymi dwoma tabelami jest taki, że każda kategoria zawiera się w zawodniku, co jest reprezentowane przez klucz obcy NumerZawodnika w tabeli Kategorie.

7.4 Projekt Modelu Fizycznego

Model fizyczny bazy danych zawiera szczegółowe odwzorowanie tabel i relacji między nimi, z uwzględnieniem specyfiki systemu zarządzania bazą danych SQLite. Oto szczegółowy opis tabel:

- **Tabela Zawodnicy (SQLite)**

```
CREATE TABLE IF NOT EXISTS Zawodnicy(
key INT AUTO_INCREMENT PRIMARY KEY,
Imie VARCHAR(15),
Nazwisko VARCHAR(15),
Pseudonil VARCHAR(15),
Plec BOOL,
Narodowsc VARCHAR(15));
```

- **Tabela Kategorie (SQLite)**

```
CREATE TABLE IF NOT EXISTS Kategorie(
key INT AUTO_INCREMENT PRIMARY KEY,
NazwaKategorii VARCHAR(15),
NumerZawodnika VARCHAR(15),
LiczbaZlotychMedali INT,
LiczbaSrebnychMedali INT,
LiczbaBrazowychMedali INT,
CzyAktynwy BOOL,
DataEmerytura DATE,
DataStart DATE);
```

- **Tabela Zawodnicy (PostgreSQL)**

```
CREATE TABLE IF NOT EXISTS Zawodnicy(
key INT PRIMARY KEY,
Imie VARCHAR(15),
Nazwisko VARCHAR(15),
Pseudonil VARCHAR(15),
Plec BOOL,
Narodowsc VARCHAR(15))
```

- **Tabela Kategorie (PostgreSQL)**

```
CREATE TABLE IF NOT EXISTS Kategorie(
key INT PRIMARY KEY,
NazwaKategorii VARCHAR(30),
NumerZawodnika VARCHAR(15),
LiczbaZlotychMedali INT,
LiczbaSrebnychMedali INT,
LiczbaBrazowychMedali INT,
CzyAktynwy BOOL,
DataEmerytura DATE,
DataStart DATE)
```

W powyższych tabelach zdefiniowano odpowiednie typy danych oraz relacje między tabelami. Klucz obcy NumerZawodnika w tabeli Kategorie odnosi się do klucza głównego w tabeli Zawodnicy.

7.5 Implementacja Operacji na Bazie Danych

7.5.1 W projekcie zaimplementowano następujące funkcje do obsługi bazy danych SQLite:

- *dbcreater()*

Tworzy połączenie z bazą danych SQLite bazasqliteWiM.db i dwie tabele: Zawodnicy i Kategorie, jeśli nie istnieją.

- *READERzawodnicyfromCSV()*

Odczytuje dane z pliku CSV dataZaw.csv i wyświetla je w konsoli.

- *READERkategoriefromCSV()*

Odczytuje dane z pliku CSV dataZ.csv i wyświetla je w konsoli.

- *DodajZawodnika(g, im, na, ps, plec, nar)*

Dodaje nowego zawodnika do tabeli Zawodnicy

- *ZmienZawodnika(g, im, na, ps, plec, nar)*

Aktualizuje dane zawodnika w tabeli Zawodnicy.

- *UsunZawodnika(g)*

Usuwa zawodnika z tabeli Zawodnicy na podstawie klucza podstawowego.

- *DodajKategorie(g, Ka, Nz, Lz, Ls, Lb, Za, De, DPz)*

Funkcja służy do dodawania nowego rekordu do tabeli Kategorie.

- *ZmienKategorie(g, Ka, Nz, Lz, Ls, Lb, Za, De, DPz)*

Aktualizuje dane rekordu w tabeli Kategorie.

- *UsunKategorie(g)*

Usuwa rekord z tabeli Zawodnicy na podstawie klucza podstawowego.

- *DodajKategorie(g, Ka, Nz, Lz, Ls, Lb, Za, De, DPz)*

Funkcja służy do dodawania nowego rekordu do tabeli Kategorie.

- *ZmienKategorie(g, Ka, Nz, Lz, Ls, Lb, Za, De, DPz)*

Aktualizuje dane rekordu w tabeli Kategorie.

- *UsunKategorie(g)*

Usuwa rekord z tabeli Zawodnicy na podstawie klucza podstawowego.

- *ZRZUTZAW()*

Zapisuje/nadpisuje aktualny stan tabeli Zawodnicy do pliku csv.

- *ZRZUTZKAT()*

Zapisuje/nadpisuje aktualny stan tabeli Kategorie do pliku csv.

- *ZRZUTZALL()*

Zapisuje/nadpisuje aktualny stan obu tabel do pliku csv.

- *dbclean()*

Usuwa tablice i zastępuje je nowymi.

- *GenerujDanePierwszaTablica()*

Generuje przykładowe dane do tablicy Zawodnicy.

- *GenerujDaneDrugaTablica(int i)*

Generuje przykładowe dane do tablicy Zawodnicy.

- *dblosowe()*

Korzysta z GenerujDaneDrugaTablica i GenerujDanePierwszaTablica do generowania wielu danych naraz.

7.5.2 W projekcie zaimplementowano następujące funkcje do obsługi bazy danych PostgreSQL:

- *def DBcreate(plikJSON)*

Tworzy połączenie z bazą danych postgres i dwie tabele: Zawodnicy i Kategorie, jeśli nie istnieją.

- *CCSVtoDB (dataZ, dataZaw, plikJSON)*

Przerzuca dane z plików csv do bazy danych.

- *addZawodnicy (plikJSON,g,im,na,ps,plec,nar)*

Dodaje nowego zawodnika do tabeli Zawodnicy.

- *updateZawodnicy (plikJSON,g,im,na,ps,plec,nar)*

Aktualizuje dane zawodnika o kluczu równym g w tabeli Zawodnicy.

- *deleteZawodnicy (plikJSON,g)*

Usuwa zawodnika z tabeli Zawodnicy na podstawie klucza podstawowego.

- *addKategorie (plikJSON,g,Ka,Nz,Lz,Ls,Lb,Za,De,DPz)*

Funkcja służy do dodawania nowego rekordu do tabeli Kategorie.

- *updateKategorie (plikJSON,g,Ka,Nz,Lz,Ls,Lb,Za,De,DPz)*

Aktualizuje dane rekordu o kluczu g w tabeli Kategorie.

- *deleteKategorie (plikJSON,g)*

Usuwa rekord z tabeli Zawodnicy na podstawie klucza podstawowego.

- *CleanDB(plikJSON)*

Usuwa tablice i zastępuje je nowymi.

7.6 Użyte biblioteki

- csv
- sqlite3
- matplotlib
- numpy
- psycopg
- simplejson

7.7 Wnioski

W trakcie pracy podczas zajęć laboratoryjnych przećwiczyliśmy i zapoznaliśmy się jak przydatnym narzędziem do komunikacji z bazą danych jest język python który stanowi sprawną alternatywę dla języka PHP, mogliśmy poznać podstawowe problemy towarzyszące podczas implementacji aplikacji działającej na bazie danych (jak np. zastępowanie braku danych w dacie emerytury kiedy użytkownik wprowadza że sportowiec dalej jest aktywny zawodowo), poznać podstawowe pułapki które czekają podczas projektowania baz danych (jak na przykład niezastosowanie drugiej postaci normalizacji podczas planowania i obowiązek implementacji kodu który próbuje chronić bazę przed anomaliami) oraz lepiej poznać naturę baz danych licznie stosowanych w informatyce.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search