Sprawozdanie_z_Baz_Danych

Release 1.0

Kacper Rasztar

Jul 05, 2025

SPIS TREŚCI

Add your content using `reStructuredText` syntax. See the reStructuredText documentation for details.

CHAPTER

ONE

WPROWADZENIE

Kurs: Bazy Danych 1

Prowadzący: Piotr Czaja

Autor: Kacper Rasztar

Sprawozdanie z kursu Bazy danych stanowiące podsumowanie pracy wykonanej indywidualnie. Dokumentacja obejmuje przegląd literaturowy dotyczący baz danych oraz autorski projekt bazy danych dla wędkarskiego sklepu internetowego. Projekt został szczegółowo omówiony na trzech poziomach: modelu konceptualnym, logicznym oraz fizycznym. Raport zawiera również analizę działania bazy danych oraz opis zastosowanych metod optymalizacji zapytań.

W trakcie realizacji projektu wykorzystano dwa systemy zarządzania bazami danych: PostgreSQL oraz LiteSQL. Całość dokumentacji została przygotowana przy użyciu narzędzia Sphinx, co zapewniło spójność oraz łatwość nawigacji po wszystkich częściach raportu.

CHAPTER

TWO

MODELE BAZY DANYCH

Author: Kacper Rasztar

2.1 Wprowadzenie

W tym rozdziale przedstawiono modele bazy danych zaprojektowane i zaimplementowane w dwóch środowiskach: **SQLite** oraz **PostgreSQL**. Projekt dotyczy systemu sklepu wędkarskiego i obejmuje strukturę logiczną i fizyczną bazy danych, której celem jest zarządzanie produktami, klientami, zamówieniami oraz płatnościami.

2.2 Model Konceptualny

Baza danych składa się z 5 głównych tabel:

- **klienci** – zawiera dane użytkowników sklepu,
- **produkty** – przechowuje informacje o dostępnych towarach,
- **zamówienia** – reprezentuje zakupy klientów,
- **płatności** – powiązane z zamówieniami,
- **kategorie** – grupuje produkty według typów.

Relacje:

- Jeden klient może mieć wiele zamówień (relacja jeden-do-wielu),
- Każde zamówienie ma jedną płatność (relacja jeden-do-jednego),
- Produkty należą do jednej kategorii (relacja wiele-do-jednego).

2.3 Model Logiczny

Tabela klienci:

- *id* – liczba całkowita, klucz główny,
- *imie* – tekst,
- *nazwisko* – tekst,
- *email* – tekst (unikalny),
- *telefon* – tekst,
- *adres* – tekst.

Tabela produkty:

- *id* – liczba całkowita, klucz główny,
- *nazwa* – tekst,
- *opis* – tekst,
- *cena* – liczba zmiennoprzecinkowa,
- *stan_magazynowy* – liczba całkowita,
- *kategoria_id* – liczba całkowita, klucz obcy.

Tabela zamówienia:

- *id* – liczba całkowita, klucz główny,
- *klient_id* – liczba całkowita, klucz obcy,
- *data_zamowienia* – data,
- *status* – tekst (np. ‘nowe’, ‘w_realizacji’, ‘zrealizowane’).

Tabela płatności:

- *id* – liczba całkowita, klucz główny,
- *zamowienie_id* – liczba całkowita, klucz obcy,
- *kwota* – liczba zmiennoprzecinkowa,
- *metoda_platnosci* – tekst,
- *data_platnosci* – data.

Tabela kategorie:

- *id* – liczba całkowita, klucz główny,
- *nazwa* – tekst,
- *opis* – tekst.

2.4 Model Fizyczny

Implementacja w SQLite:

- *TEXT* używany dla danych tekstowych (np. imie, nazwisko, email),
- *INTEGER* dla identyfikatorów i wartości liczbowych,
- *REAL* dla cen i kwot,
- *DATE* dla dat (przechowywane jako tekst w formacie ISO).

Implementacja w PostgreSQL:

- *VARCHAR* dla tekstów (np. VARCHAR(100) dla nazw i emaili),
- *INTEGER* dla kluczy głównych i liczbowych pól,
- *DECIMAL(10,2)* dla cen i kwot,
- *DATE* dla dat,
- *TEXT* dla dłuższych opisów i adresów.

Relacje między tabelami zostały zaimplementowane przy użyciu kluczy obcych (FOREIGN KEY) oraz ograniczeń spójności.

2.5 Podsumowanie

Zaprojektowana baza danych jest znormalizowana i zapewnia integralność danych oraz możliwość łatwego rozszerzania funkcjonalności sklepu internetowego. Obsługuje zarówno zapisywanie danych testowych, jak i ich import/eksport w różnych formatach (CSV, JSON).

CHAPTER

THREE

ANALIZA BAZY DANYCH ORAZ OPTYMALIZACJA ZAPYTAŃ

Autor: Kacper Rasztar

3.1 Wstęp

W tym rozdziale przeprowadzono analizę wydajności zapytań SQL oraz ocenę struktury bazy danych dla dwóch środowisk: SQLite oraz PostgreSQL. Analiza ta obejmuje ocenę szybkości działania zapytań, wykorzystania indeksów oraz potencjalnych możliwości optymalizacji.

3.2 Migracja danych między SQLite a PostgreSQL

Proces migracji między bazami wymaga odpowiedniego przekształcenia typów danych oraz dostosowania ograniczeń i struktur tabel.

Z SQLite do PostgreSQL:

- TEXT → VARCHAR, INTEGER → SERIAL, REAL → DECIMAL
- Eksport danych do plików CSV
- Import danych do PostgreSQL za pomocą *COPY* lub pgloader
- Wprowadzenie kluczy obcych i ograniczeń

Z PostgreSQL do SQLite:

- SERIAL → INTEGER PRIMARY KEY AUTOINCREMENT
- VARCHAR → TEXT, DECIMAL → REAL
- Eksport do CSV i import poprzez *.import*
- Upraszczanie schematu (mniej restrykcyjne klucze)

Wnioski:

- PostgreSQL zapewnia większą kontrolę nad typami i relacjami
- SQLite pozwala na szybsze prototypowanie, ale wymaga dodatkowej walidacji

3.3 Analiza zapytania SQLuser_price

Poniżej zaprezentowano zapytanie SQL, które oblicza sumę zakupów dla każdego użytkownika:

```
SELECT
    u.imie || ' ' || u.nazwisko AS nazwa_uzytkownika,
    SUM(p.cena * pz.ilosc) AS suma_zakupow
FROM Uzytkownicy u
JOIN Zamowienia z ON u.id_uzytkownika = z.id_uzytkownika
JOIN PozycjeZamowienia pz ON z.id_zamowienia = pz.id_zamowienia
JOIN Produkty p ON pz.id_produktu = p.id_produktu
GROUP BY u.id_uzytkownika
ORDER BY suma_zakupow DESC;
```

Opis działania:

- Łączy tabele użytkowników, zamówień, pozycji zamówień i produktów
- Wylicza sumę wydatków każdego użytkownika
- Sortuje wyniki malejąco

Wnioski z analizy EXPLAIN ANALYZE:

- Zapytanie może działać wolno na dużych zbiorach danych bez indeksów
- Zalecane utworzenie indeksów na kolumnach: *id_uzytkownika*, *id_zamowienia*, *id_produktu*

Przykład indeksów (PostgreSQL):

```
CREATE INDEX idx_zamowienia_id_uzytkownika ON Zamowienia(id_uzytkownika);
CREATE INDEX idx_pozycje_id_zamowienia ON PozycjeZamowienia(id_zamowienia);
CREATE INDEX idx_pozycje_id_produktu ON PozycjeZamowienia(id_produktu);
```

Wydajność zapytania można porównać przy użyciu *EXPLAIN ANALYZE*:

```
EXPLAIN ANALYZE SELECT * FROM Produkty WHERE cena > 100;

EXPLAIN ANALYZE SELECT * FROM Produkty WHERE nazwa LIKE 'A%';
```

3.4 Monitorowanie i optymalizacja

- Regularne stosowanie *EXPLAIN* i *EXPLAIN ANALYZE* umożliwia analizę planów zapytań
- Indeksy na kolumnach wykorzystywanych w filtrach (*WHERE*, *JOIN*, *ORDER BY*) znacząco poprawiają wydajność
- W PostgreSQL zaleca się również wykonywanie *ANALYZE* i *VACUUM* w celu utrzymania aktualnych statystyk

Przykład użycia EXPLAIN:

```
EXPLAIN SELECT * FROM Zamowienia WHERE status = 'zrealizowane';

EXPLAIN ANALYZE SELECT * FROM PozycjeZamowienia WHERE ilosc > 10;
```

3.5 Podsumowanie

Analiza pokazała, że baza danych może być skutecznie przenoszona pomiędzy środowiskami SQLite i PostgreSQL z zachowaniem spójności danych. Kluczowym elementem zapewnienia wydajności jest stosowanie indeksów oraz testowanie zapytań przy pomocy narzędzi takich jak *EXPLAIN*. Dobrze zaprojektowane zapytania oraz utrzymana struktura bazy danych pozwalają uniknąć opóźnień i przeciążeń w działaniu systemu sklepu internetowego.

CHAPTER

FOUR

SPIS WYKORZYSTANYCH REPOZYTORIÓW

4.1 Główne repozytorium dokumentacji

https://github.com/krasztar/Sprawozdanie

4.2 Moje repozytoria

Przegląd literatury : https://github.com/krasztar/Literatura

Repozytorium ze skryptami i implementacją baz danych: https://github.com/krasztar/Modele_fizyczne

4.3 Pozostałe repozytoria z przeglądem literatury

1. Sprzęt dla bazy danych https://github.com/oszczeda/Sprzet-dla-bazy-danych
2. Konfiguracja bazy danych https://github.com/Chaiolites/Konfiguracja_baz_danych
3. Kontrola i konserwacja https://github.com/Pi0trM/Kontrola_i_konserwacja
4. Monitorowanie i diagnostyka https://github.com/GrzegorzSzczepanek/repo-wspolne
5. Wydajność, skalowanie i replikacja https://github.com/Broksonn/Wydajnosc_Skalowanie_i_Replikacja
6. Partycjonowanie danych https://github.com/BartekHen/Partycjonowanie-danych
7. Kopie zapasowe i odzyskiwanie danych https://github.com/m-smieja/Kopie_zapasowe_i_odzyskiwanie_danych
8. Bezpieczeństwo https://github.com/BlazejUl/bezpieczenstwo