Baza danych medium społecznościowego

Łukasz Fabia 272724
Mikołaj Kubś 272662
Martyna Łopianiak 272682
Piotr Schubert 272659
Projektowanie baz danych wt 18:55

21 stycznia 2025

Spis treści

1 Etap 1: Faza konceptualna **2**
1.1 Analiza świata rzeczywistego . . . 2
1.1.1 Streszczenie - Zarys wymagań projektu . . . 2
1.1.2 Potrzeby informacyjne . . . 2
1.1.3 Czynności, wyszukiwania . . . 2
1.1.4 Cele projektu . . . 3
1.1.5 Zakres projektu . . . 3
1.2 Wymagania funkcjonalne . . . 3
1.3 ERD . . . 4

2 Etap 2: Faza logiczna **4**

3 Etap 3: Faza fizyczna **5**

4 Etap 4: Faza fizyczna 2 **6**

5 Etap 5: Faza fizyczna 3 **7**
5.1 Projekt interfejsu graficznego . . . 7
5.2 Kwerendy SQL . . . 8

6 Etap 6: Faza fizyczna 4 **9**
6.1 Różne typy indeksów i ich zastosowanie . . . 9
6.2 Obserwacje . . . 10

7 Etap 7: Faza fizyczna 5 **11**
7.1 Sprawozdanie powykonawcze . . . 11
7.1.1 Mocne strony projektu (S) . . . 11
7.1.2 Słabe strony projektu (W) . . . 12
7.1.3 Możliwości (O) . . . 12
7.1.4 Zagrożenia (T) . . . 12
7.1.5 Podsumowanie . . . 12
7.2 Wprowadzone modyfikacje . . . 13
7.2.1 Porzucone pomysły . . . 13
7.2.2 Nowe encje . . . 13
7.2.3 Rozdzielenie lokacji na 3 table . . . 13

8 Etap 8: Faza logiczna I **14**
8.1 Update ERD'a . . . 14
8.2 Update schematu relacji . . . 14
8.3 Weryfikacja i aktualizacja więzów integralności . . . 14

9 Etap 9: Faza logiczna II **17**
9.1 Przetworzenie struktury bazy danych sprzed modyfikacji do zgodnej z nowymi wymaganiami . . . 17
9.2 Przykładowe zapytania . . . 18

10 Etap 10: Nierelacyjne bazy danych - faza wstępna **19**

11 Etap 11 - Faza konceptualna i fizyczna **19**
11.1 Definicja i wdrożenie struktur przechowywania danych w wybranej technologii nierelacyjnej. . . . 19
11.2 Prezentacja przykładowych zapytań . . . 20

12 Etap 12: Faza fizyczna **20**

1 Etap 1: Faza konceptualna

1.1 Analiza świata rzeczywistego

1.1.1 Streszczenie - Zarys wymagań projektu

Celem projektu jest stworzenie bazy danych do obsługi medium społecznościowego. Ma ona przechowywać informacje o użytkownikach, ich treściach, relacjach i aktywnościach. Baza powinna być zaprojektowana w sposób wydajny i skalowalny.

1.1.2 Potrzeby informacyjne

- Rejestracja i logowanie użytkowników z różnymi poziomami dostępu.
- Publikowanie i interakcje z treściami użytkowników (posty, polubienia, komentarze).
- Zarządzanie relacjami społecznymi (znajomości).
- Przechowywanie wiadomości prywatnych i historii aktywności.
- Tworzenie konwersacji z innymi użytkownikami
- Reportowanie postów z nieodpowiednimi treśćmi
- Tworzenie i zarządzanie stronami organizacji, firm, fanpage itd.
- Tworzenie i zarządzanie wydarzeniami

1.1.3 Czynności, wyszukiwania

- Wyszukiwanie użytkowników.
- Wyszukiwanie treści po hasztagach lub słowach kluczowych.
- Filtrowanie aktywności użytkownika, np. przeglądanie polubień i komentarzy.
- Wyszukiwanie relacji (np. znajomi użytkownika, osoby obserwujące daną osobę).
- Dodawanie innych użytkowników do znajomych i interakcja z treściami - dodawanie komentarzy i reakcji
- Tworzenie postów, wydarzeń, grup
- Konwersacja grupowa, pisanie wiadomości

1.1.4 Cele projektu

S (Specific): Zaprojektowanie bazy danych dla medium społecznościowego.

M (Measurable): Baza musi być wydajna, tzn. musi być w stanie obsługiwać dużą ilość użytkowników, co najmniej 20 000.

A (Achievable): Projekt zostanie zrealizowany przy użyciu PostgreSQL. Do stworzenia struktury tabel wykorzystany zostanie mechanizm ORM (Object Relational Mapping). Na koniec baza zostanie wypełniona danymi, aby przetestować jej wydajność.

R (Relevant): Przechowywanie profili użytkowników oraz interakcje między nimi są kluczowe dla funkcjonowania medium społecznościowego.

T (Time-bound): Praca nad projektem powinna zająć 2 miesiące.

1.1.5 Zakres projektu

Multimedia: W bazie przechowywane będą wyłącznie linki do plików na zewnętrznym serwerze.

Obsługa haseł: Wszystkie hasła w bazie będą hashowane.

1.2 Wymagania funkcjonalne

Streszczenie

Użytkownikom przypisany jest jeden z tych poziomów dostępu: admin, user lub guest.

Guest(Gość)

- Może przeglądać wybrane dane.

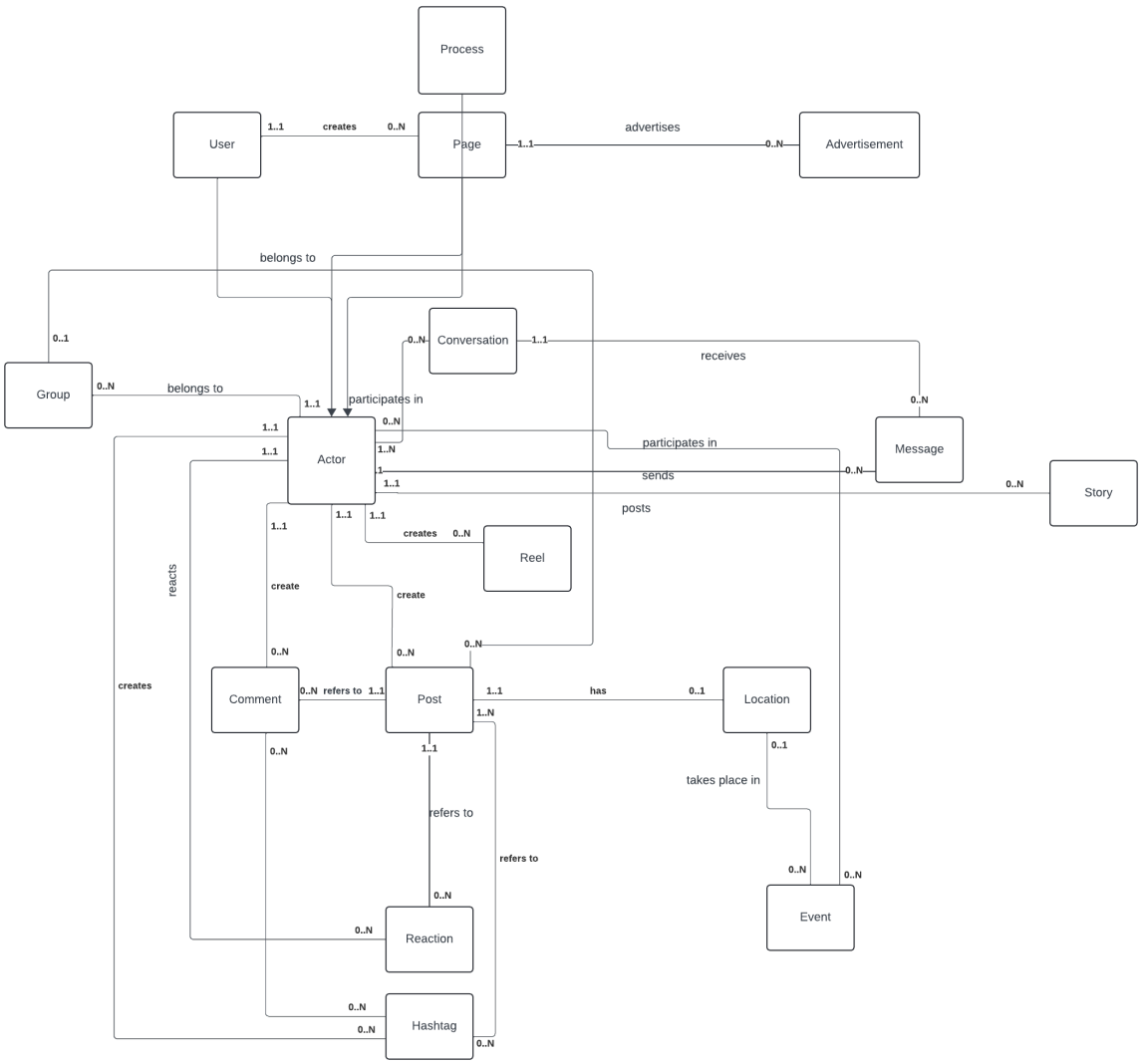
Admin

- Może przeglądać, edytować, usuwać, dodawać i przeglądać wszystkie treści, zrządza bazą i nadaje uprawnienia

User(Użytkownik)

- System umożliwia rejestrację oraz logowanie.
- Rejestracja wymaga imienia, nazwiska, daty urodzenia, hasła oraz maila.
- Logowanie wymaga maila i hasła.

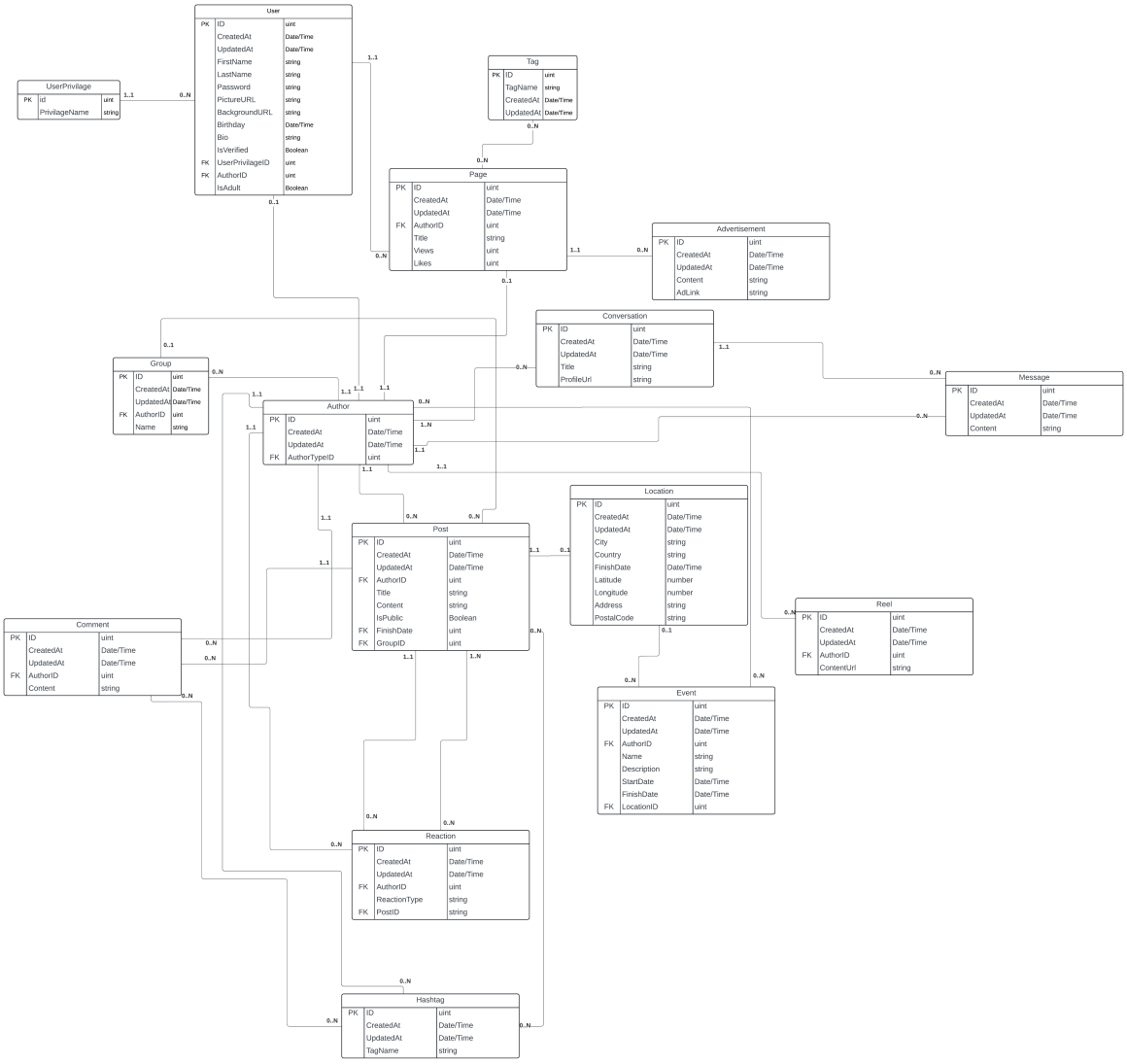
1.3 ERD



Rysunek 1: Diagram obiektowo-relacyjny

2 Etap 2: Faza logiczna

Do tabel zostały dodane atrybuty, tabele **Page** oraz **User** zostały uogólnione przez **Author**, który bierze udział w innych czynnościach. Baza została także sprowadzona do *III postaci normalniej*, przez co wydzielono klika nowych tabel.



Rysunek 2: Diagram relacji

3 Etap 3: Faza fizyczna

Encje, które uległy zmianie:

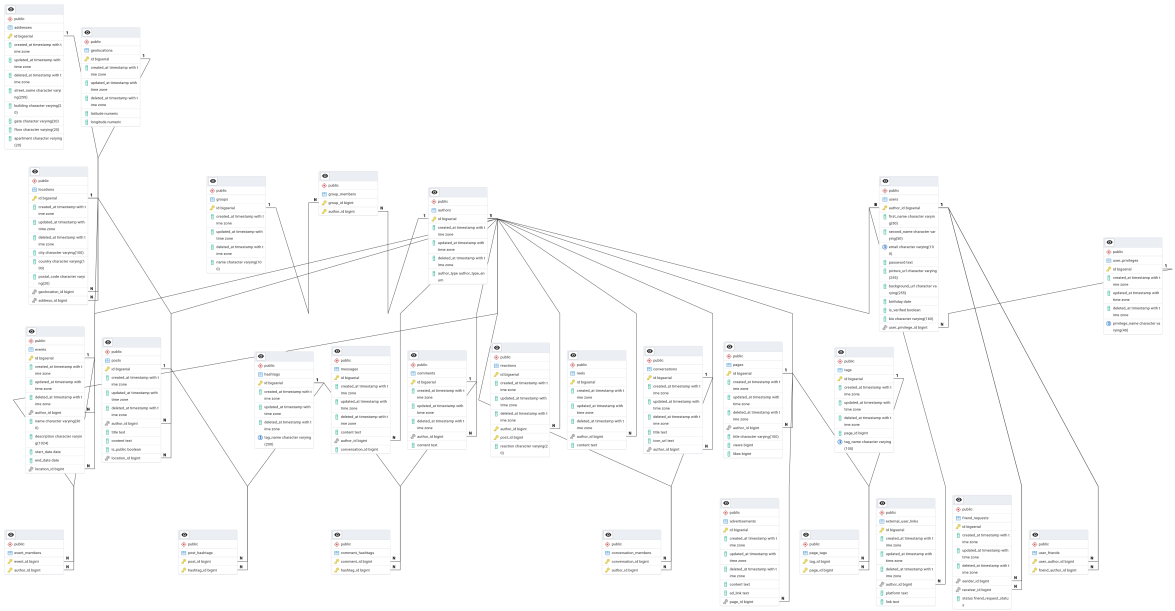
- Rozbicie lokalizacji na pomniejsze tabele
- Zamiana na enumy: typu autora oraz statusu zaproszenia do znajomych
- Dodanie do encji Conversation pole Members, które przechowuje id użytkowników biorących udział w konwersacji

Wykorzystano instrukcję CHECK do potwierdzenia poprawności danych w paru encjach. Przykładowo:

- 'A' nie moze być przyjacielem 'A'
- 'A' nie moze wysłać zaproszenia do znajomych do 'A'
- Czas rozpoczęcia wydarzenia musi być wcześniejszy niż czas zakończenia wydarzenia
- Grupa musi mieć od 1 do 10000000 członków.

Do stworzenia struktury bazy wykorzystano mechanizm ORM - (gorm). Sama baza wymagała dopracowania jeśli chodzi o enumeracje oraz reguły usuwania już bezpośrednio w systemie PostgreSQL (pgadmin).

Usuwanie: Każdy model posiada pole *DeletedAt* z indeksem. Podczas usuwania danego wiersza pole *DeletedAt* jest ustawiane na znacznik czasu (timestamp) wskazujący moment, w którym dane zostały usunięte. Dzięki temu baza obsługuje soft deleting. Oznacza to, że gdy użytkownik usunie swoje konto, będzie można je przywrócić. Taką operację obsługują na przykład serwisy takie jak Facebook.



Rysunek 3: Diagram relacji z PostgreSQL

4 Etap 4: Faza fizyczna 2

Standardowo kod można zobaczyć w repozytorium na gicie: social media db.

Mając napisany ORM poprzenio, w tym etapie pozostało napisać funkcje generujące dane do bazy. Rozwiązanie można podzielić na 3 cześci (od szczegółu do ogółu):

1. Dekorator, który będzie wykonywać określny blok `count` razy.
2. Funkcja, generująca dany typ danych np. generator użytkowników.
3. Odpowienie ułożenie wywołań.

Zapełnianie bazy danych dużą ilością danych zajmuje stosunkowo dużo czasu, może to być spowodowane przez unikalność niektórych atrybutów(biblioteka "męczy" się z generowaniem unikalnych sensownych danych), ale także przez wywołania które tworzą listy autorów dla np. eventów, ostatnim podejrzeniem może być nieoptymalnie napisany kod.

Język: Go
Generowanie sztucznych danych: gofakeit
Object relational mapping (ORM): gorm

Reszta rzeczy, które zostały wykonane:

- Funkcja usuwająca dane z tabel
- Własne implementacje niektórych danych np. `Title`, `Birthday`

- Konfiguracja loggera
- Funkcja haszująca hasło

5 Etap 5: Faza fizyczna 3

W tej fazie projektu stworzyliśmy projekt interfejsu graficznego w programie Figma oraz napisaliśmy 10 nietrywialnych kwerend w SQL.

5.1 Projekt interfejsu graficznego

Zaprojektowaliśmy następujące 10 widoków do aplikacji mobilnej:

- Strona główna
- Strona komentarzy i rekacji
- Strona profilu
- Strona strony (page)
- Strona grupy
- Strona wydarzenia
- Strona konwersacji
- Strona panelu konwersacji
- Strona rolek (reel)
- Strona ze znajomymi

Podczas projektowania zauważyliśmy parę drobnych rzeczy, które można by było poprawić - np. brak ikony czy grafiki tła dla strony. Tak więc poprawiliśmy takie nieścisłości i zaktualizowaliśmy kod seeder.go, który zajmuje się generowaniem tych danych.

Pełny projekt interfejsu graficznego znajduje się w załączonym pliku "figma.pdf".

Rysunek 4: 1 z widoków interfejsu graficznego

5.2 Kwerendy SQL

Poniżej znajdują się 10 kwerend SQL, które zostały napisane w celu przeszukania i stworzenia raportów na podstawie bazy danych:

- 10 najpopularniejszych hashtagów z ostatnich 7 dni
- Konwersacje autora
- Publiczne posty autora
- Średnia liczba znajomych
- Średnia liczba poszczególnych reakcji na posty w ostatnim miesiącu
- Wydarzenia w danym mieście (np. Saint Petersburg)
- Wydarzenia, w których wezmą udział znajomi autora
- Tagi stron, posortowane po średniej liczbie polubień i wyświetleń
- Użytkownicy, z którymi użytkownik ma wspólnych znajomych
- Użytkownicy z najwiekszą średnią liczbą reakcji na posty

	user_author_id bigint	first_name character varying (50)	second_name character varying (50)	mutualfriendscount bigint
1	2	Jazmin	Donnelly	3
2	247	Tess	Gusikowski	2
3	281	Francisca	Hoppe	2
4	154	Micheal	Jewess	2
5	326	Abby	Miller	2
6	244	Jana	Spinka	2
7	460	Lisette	Bednar	1
8	328	Darrion	Blanda	1
9	107	Danny	Boehm	1
10	441	Bernhard	Bogan	1

Rysunek 5: Wynik kwerendy wyszukującej użytkowników ze wspólnymi znajomymi

	id [PK] bigint	name character varying (300)	start_date date	end_date date	numberoffriendsattending bigint
1	5507	Seminar Civis Analytics	2024-11-18	2024-11-28	3
2	7711	Summit Robinson + Yu	2024-11-18	2024-11-24	2
3	4220	Festival Embark	2024-11-18	2024-11-21	2
4	10376	Party TransUnion	2024-11-18	2024-11-28	2
5	6989	Seminar Bekins	2024-11-18	2024-11-22	2
6	10096	Party Morgan Stanley	2024-11-18	2024-11-26	2
7	7596	Summit Weather Decision Technologies	2024-11-18	2024-11-28	2
8	11376	Party iRecycle	2024-11-18	2024-11-27	2
9	8206	Summit StreetCred Software, Inc	2024-11-18	2024-11-27	2
10	5778	Summit xDayta	2024-11-18	2024-11-28	2
11	4122	Webinar KPMG	2024-11-18	2024-11-25	2
12	153	Summit SmartProcure	2024-11-18	2024-11-21	2
13	5392	Summit Ensco	2024-11-18	2024-11-24	2
14	11168	Party StreamLink Software	2024-11-18	2024-11-26	2
15	2615	Seminar CitySourced	2024-11-18	2024-11-27	2
16	9197	Festival American Red Ball Movers	2024-11-18	2024-11-22	2
17	1941	Party Recargo	2024-11-18	2024-11-28	2
18	11513	Workshop SAS	2024-11-18	2024-11-22	2

Rysunek 6: Wynik kwerendy wyszukującej wydarzenia, na które zapisali się znajomi użytkownika

6 Etap 6: Faza fizyczna 4

Przykładowy wynik działania `EXPLAIN`

```
Seq Scan on foo  (cost=0.00..155.00 rows=10000 width=4)
```

Ogólnie można powiedzieć, że komenda do dostarcza bardzo dokładne informacje w porównaniu np. do **MySQL**.

Powyższa komenda to sposób w jaki zapytanie może zostać wykonane. W tym wypadku mamy wykonanie sekwencyjne, czyli skanujemy całą tabelę w celu znalezienia wierszy spełniających warunki w kwerendzie.

Koszt(cost) - wartość po lewej to koszt początkowy zapytania tutaj jest on równy 0.00, kolejna wartość to szacowany koszt operacji, wartość ta jest obliczana przez system bazodanowy.
Wiersze(rows) - ilość wierszy spełniająca dane kryteria.
Szerokość(width) - waga wiersza w bajtach

Zatem optymalizacja zapytań będzie polegała na minimalizacji tych "kosztów". W tym celu wykorzystuje się **indeksy**.

6.1 Różne typy indeksów i ich zastosowanie

Typ indeksu	Zastosowanie
B-Tree	Wyszukiwanie, sortowanie, zakresy, klucze główne, indeksy unikalne.
Hash	Szybkie porównania równości (=).
GiST	Dane przestrzenne, zakresy, hierarchie.
BRIN	Duże tabele z posortowanymi danymi, dane archiwalne.

6.2 Obserwacje

Indeksy nie pomogą w każdej operacji - na przykład dla kwerend, gdzie największym kosztem jest grupowanie, mogą niekoniecznie pomóc.

Hash nie przynosi znaczącej poprawy wydajności w naszych zapytaniach w porównaniu z B-tree.

Zapytania wykorzystujące funkcje agregujące są trudne do optymalizacji, a dodanie indeksów w większości przypadków nie przyniosło zauważalnych korzyści.

Wyjątek stanowi zapytanie **show_events_in_st_petersburg**, gdzie zastosowanie indeksu złożonego przyniosło wyraźną poprawę:

- Zastosowanie indeksu typu **B-tree** zmniejszyło koszt zapytania o około 10%.
- Dodanie indeksu typu **BRIN** dodatkowo obniżyło koszt o kolejne 10%.

Dla zapytań niewykorzystujących funkcji agregujących, dodanie indeksów pozwoliło na kilkukrotne zmniejszenie kosztów ich wykonania.

Rysunek 7: Przykładowe czasy zapytań bez i z indeksami.

Rysunek 8: Przykładowe czasy zapytań bez i z indeksami.

Rysunek 9: Optymalizacja zapytania z wydarzeniami w Piotrogrodzie

7 Etap 7: Faza fizyczna 5

7.1 Sprawozdanie powykonawcze

7.1.1 Mocne strony projektu (S)

Jedną z pierwszych mocnych stron projektu jest wydzielenie tabeli `authors`. Pozwoliło to na rozszerzenie możliwości tworzenia nowych tabel dla użytkowników (`user`, `page`). Dodatkowo w ten sposób pozbyto się *redundantnych* relacji, ponieważ zarówno **strona**, jak i **użytkownik** mogli w domyśle wykonywać te same czynności na innych tabelach.

Zastosowanie usuwania kaskadowego również okazało się być bardzo pomocne, ponieważ w ten sposób przy usuwaniu jakichkolwiek danych mielibyśmy pewność, że dane pozostaną spójne.

Będąc w temacie usuwania, dane z bazy nie były całkowicie usuwane(*hard delete*). To nie tylko pozwoliłoby na przywrócenie danych, na przykład konta niezdecydowanego użytkownika, ale również pozwoliłoby śledzić historię danych. Ponadto dane, które nie są widoczne dla użytkonika mogą posłużyć np. do analiz biznesowych.

Uzyskanie zgodności z przepisami, w tym z RODO w naszym projekcie wydaje się ułatwione przez dwa czynniki:

- Haszowanie haseł
- Wydzielenie danych osobowych do osobnej tabeli, dzięki czemu można łatwo zarządzać dostępem do tych danych, a także przeprowadzić *soft delete* użytkownika bez usuwania wszystkich rzeczy, które wytworzył.
- Usuwanie kaskadowe, które pozwala na szybkie usunięcie wszystkich danych związanych z użytkownikiem.
- Ograniczenie zbieranych danych do niezbędnych, co pozwala na zminimalizowanie ryzyka naruszenia RODO.

Skrypt generujący dane również jest mocną stroną projektu. Skrypt generuje tabele w bazie danych, dodając potrzebne ograniczenia do atrybutów, enumeracje oraz dość sensowne dane. Jest on solidną bazą pod część backendową (związaną z warstwą modeli np. dla *architektury warstwowej*) aplikacji. Dodatkowym atutem jest to, że generacja danych nie jest destruktywna - mając już dane w bazie danych, możemy spokojnie wywołać metodę odpowiedzialną za generowanie dodatkowych danych w określonych przez nas tabelach.

Dodatkowo można w prosty sposób zmienić system, dla którego ma być generowana baza - my korzystaliśmy z PostgreSQL, ale bez większych problemów dałoby się zmienić system na inny (na przykład SQLite, MySQL czy SQLServer) dzięki używaniu GORM'a, który jest właśnie kompatybilny z wieloma systemami.

Został napisany również skrpty, który automatycznie uruchomi bazę w kontenerze `Dockerowym`, dzięki czemu nie trzeba instalować samego PostgreSQL, PGAdmin czy innych pluginów, żeby zarządzać tą bazą - wystarczy podać parę podstawowych danych w pliku `.env` i uruchomić skrypt. Dodatkową zaletą jest to, że baza jest uruchamiana w izolowanym środowisku, co pozwala na łatwe testowanie różnych wersji bazy danych, a także na łatwe przenoszenie bazy na inne środowisko (np. z lokalnego na serwer produkcyjny).

Dodatkową automatyzacją jest integracja backendu z komendami umieszczonymi w Makefile'u. Dzięki temu można w prosty sposób uruchomić skrypt generujący dane, uruchomić bazę danych, przeczyścić bazę i inne.

Co do wybranego systemu, system PostgreSQL, okazał się on być całkiem wydajnym pod względem operacji typu *CRUD* na tabelach. Dzięki dodaniu indeksów, kwerendy zostały jeszcze bardziej zoptymalizowane.

Wydaje nam się, że dzięki uogólnieniu wszystkich czatów do grupowych (czat 1-1 nie różni się formalnie od czatu wieloosobowego w naszej bazie) zmniejszamy ryzyko błędów przez utrzymywanie dwóch wersji konwersacji. Nie powinno to sprawić nam problemów implementacyjnych w przyszłości, a dodatkowo dodanie lub usunięcie użytkownika z konwersacji będzie bardzo proste.

Nasz projekt stara się oczywiście spełniać podstawowe założenia dobrej bazy danych - jest spójny, niepowtarzalny, znormalizowany, a także ma ograniczenia integralnościowe. Żadne dane nie występują

niepotrzebnie więcej niż raz. Encje wydają się być dobrze znormalizowane. Ograniczenia integralnościowe są zaimplementowane w postaci kluczy obcych, ograniczeń CHECK oraz unikalności. Wszędzie, gdzie to miało sens, wprowadziliśmy tabele asocjacyjne, aby uniknąć problemów związanych z relacjami wieloma do wielu - udało się je wszystkie usunąć z naszego projektu.

7.1.2 Słabe strony projektu (W)

Największą słabą stroną projektu jest dość wolno działający skrypt generujący dane. Dzieje się tak, gdy chcemy dodać więcej wierszy do tabeli - przez to, że skrypt wybiera foreign keys losowo wśród istniejących już elementów, często jest duża szansa niepowodzenia przez złamanie jakiejś zasady - dałoby się to ulepszyć dzięki bardziej inteligentemu wybieraniu foreign keys. Wygenerowanie niektórych danych wymaga też skomplikowanej i zagnieżdżonej logiki tworzenia elementów. Być może problemem jest też biblioteka *faker*, która nie bardzo radziła sobie z generowaniem bardzo losowych, często unikalnych danych.

Okazuje się, że draw.io (narzędzie do modelowania diagrmów, w naszym wypadku diagramów ERD) ma wbudowaną integrację z GORM'em, dzięki której wystarczyłoby przesunąć diagram nad okienko z kodem. Ostatecznie nie oszczędziłoby nam to dużo czasu, ale z pewnością ułatwiłoby pracę i zmniejszyłoby ryzyko literówek.

7.1.3 Możliwości (O)

Tak jak już wcześniej wspominaliśmy, baza jest dobrą podstawą do stworzenia systemu na wzór **Facebooka** czy **Twittera/X** poprzez zbudowanie jakiegoś prostego serwisu REST'owego. Dzięki temu, że serwis byłby faktycznie używany na większą skalę, zauważylibyśmy więcej problemów optymalizacyjnych. Pozwoliłoby to na dopracowanie bazy danych w miejscach, gdzie brakowałoby wydajności.

Jeśli chodzi o możliwości rozbudowy bazy to można by dodać np. lepszą obsługę mediów (wydzielenie osobnej tabeli). Można być też poprawić tabele związane z lokalizacjami i zamiast przechowywania danych koordynatów przechowywać typ `Point` do obsługi współrzędnych geograficznych wykorzystując plugin **PostGIS**. Można by wtedy ulepszyć sposób wyszukiwania wydarzeń użytkownikowi, poprzez szukanie ich w promieniu n m/km od wybranego punktu.

7.1.4 Zagrożenia (T)

Problemem baz relacyjnych jest trudność w horyzontalnym skalowaniu, ponieważ zapewnienie spójności danych i transakcyjności (ACID) w rozproszonej architekturze jest skomplikowane. Tradycyjnie łatwiej jest skalować je pionowo, czyli ulepszać pojedynczą maszynę, dodając więcej zasobów, takich jak pamięć RAM, mocniejszy procesor czy szybsze dyski. Jednak przy zastosowaniu odpowiednich technik, takich jak sharding czy replikacja, możliwe jest również skalowanie horyzontalne baz relacyjnych. Tabela, która może być problematyczna w utrzymaniu to, na przykład, tabela z wiadomościami, które użytkownicy wysyłają sobie wzajemnie. W przypadku tabeli z wiadomościami może to być problem, ponieważ może ona gwałtownie rosnąć. Dlatego lepszym rozwiązaniem byłoby podejście hybrydowe - przeniesienie niektórych tabel do baz NOSQL, dzięki czemu można by rozproszyć obliczenia, a nasz system nie miałby problemów z powolnym przeszukiwaniem bazy.

Baza wymagałaby też dużo optymalizacji, by wspierać miliony użytkowników stabilnie - aktualnie ciężko powiedzieć, jak dobrze by sobie radziła w takiej sytuacji - można by wprowadzać więcej indeksów które najlepiej radzą sobie z konkretną sytuacją, ale ciężko powiedzieć, czy to by wystarczyło.

7.1.5 Podsumowanie

Próbując wprowadzić produkt z taką bazą danych w życie, mielibyśmy wielki problem z konkurencją - Facebook, Twitter, Instagram, TikTok, LinkedIn, Pinterest, Reddit, Snapchat, Tumblr, WhatsApp, YouTube, czyli największe platformy społecznościowe, mają już swoje miejsce na rynku. Nasz produkt musiałby być bardzo innowacyjny, aby przyciągnąć użytkowników.

Jednakże, nasz projekt jest dobrym punktem wyjścia do stworzenia takiego produktu. Baza danych jest solidna, choć wymagałaby jeszcze sporo optymalizacji. Skrypt generujący dane jest bardzo pomocny do debuggowania i byłby krytycznie potrzebny tworząc MVP, ale wymagałby jeszcze sporo pracy, aby działał szybciej i bardziej efektywnie.

Czym mógłby wyróżniać się potencjalny produkt oparty na naszej bazie danych? Dobrym punktem wyjścia jest to, że nasza baza jest dość generyczna - większość wcześniej wymienionych serwisów ma już podobne encje. Tak więc można by rozwinąć tą bazę w jednym z tych 4 przykładowych kierunków:

- **Baza danych dla GoLocal** - aplikacja do tworzenia i głosowania na wydarzenia dla samorządów i zwykłych ludzi, aplikacja w formie serwisu społecznościowego. Dodanie obsługi ankiet, opinii użytkowników o wydarzeniu, rozwinięcie tabel z lokalizacjami oraz wydarzeniami, obsługa użytkowników *premium*. Usprawnienie obsługi komunikacji między użytkownikami, połączenie serwera z bazą NOSQL do zarządzania wiadomościami. W dalekiej przyszłości możnaby zastanowić się nad dedykowaną bazą do obsługi systemu rekomendacji - hurtowanie danych historycznych (usuniętych profili, rekacji, wydarzeń, postów, komentarzy).
- **Baza danych dla serwisu społecznościowego dla programistów** - dodanie tabeli związanej z projektami, repozytoriami, commitami, pull requestami, itd. Można by też dodać tabelę związane z technologiami, językami programowania, frameworkami, itd.
- **Baza danych dla serwisu społecznościowego dla naukowców** - dodanie tabeli związanej z publikacjami, konferencjami, grantami, itd. Można by też dodać tabelę związane z dziedzinami nauki, konferencjami, itd.
- **Lepszy Twitter** - przez ostatnie zamieszanie w serwisie Twitter/X, można by stworzyć serwis, który byłby bardziej przejrzysty, miałby lepsze algorytmy rekomendacji, byłby bardziej stabilny i bardziej moderowany, itd.

7.2 Wprowadzone modyfikacje

7.2.1 Porzucone pomysły

Już w początkowej fazie tworzenia projektu doszliśmy do wniosku że opcja zgłaszania postów/treści przez użytkowników wymaga nieadekwatnego nakładu pracy w stosunku do korzyści jakie nasz projekt na tym etapie mógłby z tego zyskać. Może być to dobra funkcjonalność do dodania w ramach rozwoju aplikacji, jednak na początkowych etapach stwierdziliśmy, że postawimy na jakość, a nie ilość. Dlatego zrezygnowaliśmy z tego pomysłu, aby w pełni skupić się na pozostałych funkcjonalnościach. Podobnie stało się z encją "Story". Jako że jej atrybuty praktycznie nie różniłyby się od obecnych już encji (Post, Reel) zdecydowaliśmy się na jej pominięcie w późniejszych etapach projektu. Jednak tak jak w przypadku zgłaszania postów jest to jak najbardziej funkcjonalność możliwa do dodania w przyszłości, a dzięki podobieństwu do obecnych już encji wdrożenie Story do naszej bazy danych powinno być nawet prostsze niż dodanie opcji zgłaszania postów.

7.2.2 Nowe encje

Na pierwszym diagramie ERD naszego projektu nie uwzględniliśmy encji "Tag", która została dodana w następnym etapie. Encja ta określa typ prowadzonej strony. Nie uwzględniliśmy również encji "UserPrivilege", która została dodana dopiero w fazie logicznej. Zdecydowaliśmy się wyodrębnić User Privilege do osobnej encji zamiast używać enuma, aby ułatwić modyfikację czy rozbudowę bazy danych oraz zmniejszyć ryzyko niespójności. Pojawiła się też encja "ExternalAuthorLinks" która pozwala przechowywać linki do innych social mediów użytkownika.

7.2.3 Rozdzielenie lokacji na 3 table

Encja związana z lokacją została rozdzielona na 3 osobne encje: Location, Geolocation, Address. Taka struktura danych niweluje powielanie informacji, ponieważ miasta, kody pocztowe czy nawet współrzędne geograficzne mogą być takie same dla wielu adresów. Dodatkowo możemy teraz w bardzo precyzyjny sposób określić adres, do jakiego się odwołujemy - mamy możliwość sprecyzowania ulicy, budynku, klatki schodowej, piętra i mieszkania. Dzięki temu, niezależnie od konwencji zapisu adresu (które mogą się różnić w zależności od kraju) użytkownik będzie w stanie wprowadzić go do naszej bazy danych. Poza tym rozdzielenie lokacji na osobne encje zmniejsza ryzyko niespójności, ułatwia modyfikacje i rozszerzanie bazy danych oraz sprawia, że zapytania na tabeli Location są bardziej wydajne. Jako że atrybuty tej encji zostały ograniczone do minimum, a bardziej złożone dane przechowywane są

w osobnych tabelach, podczas wykonywania na przykład zapytań dotyczących wydarzeń w danym mieście nie potrzebujemy importować danych ze wszystkich tabel - wystarczy nam sama tabela Location co zwiększa wydajność zapytań.

8 Etap 8: Faza logiczna I

8.1 Update ERD'a

Diagram ERD 10 został zaktualizowany - odzwierciedla schemat relacji w bazie oraz jest czytelniejszy względem poprzedniego. Zdecydowaliśmy się nie umieszczać szczątkowych artybutów dla encji, ponieważ wprowadziło by to zbędne zamieszanie i wpłynęło by to na czytelność.

Zmieniono:

- Dodano `User Privilege`
- Rozbito `Location`, na pomniejsze encje (Address, Geolocation)
- Wyodrębniono nową encje `Tag`
- Usunięto encje `Story`
- Pokazaliśmy na diagramie relacje użytkownika z innymi użytkownikami (asocjacja *likes (as buddy)*) oraz podobnie pokazaliśmy wysyłanie zaproszeń do znajomych (asocjacji *receives*, *sends*).
- No i usuneliśmy encje `Process`, dodaną przypadkiem.

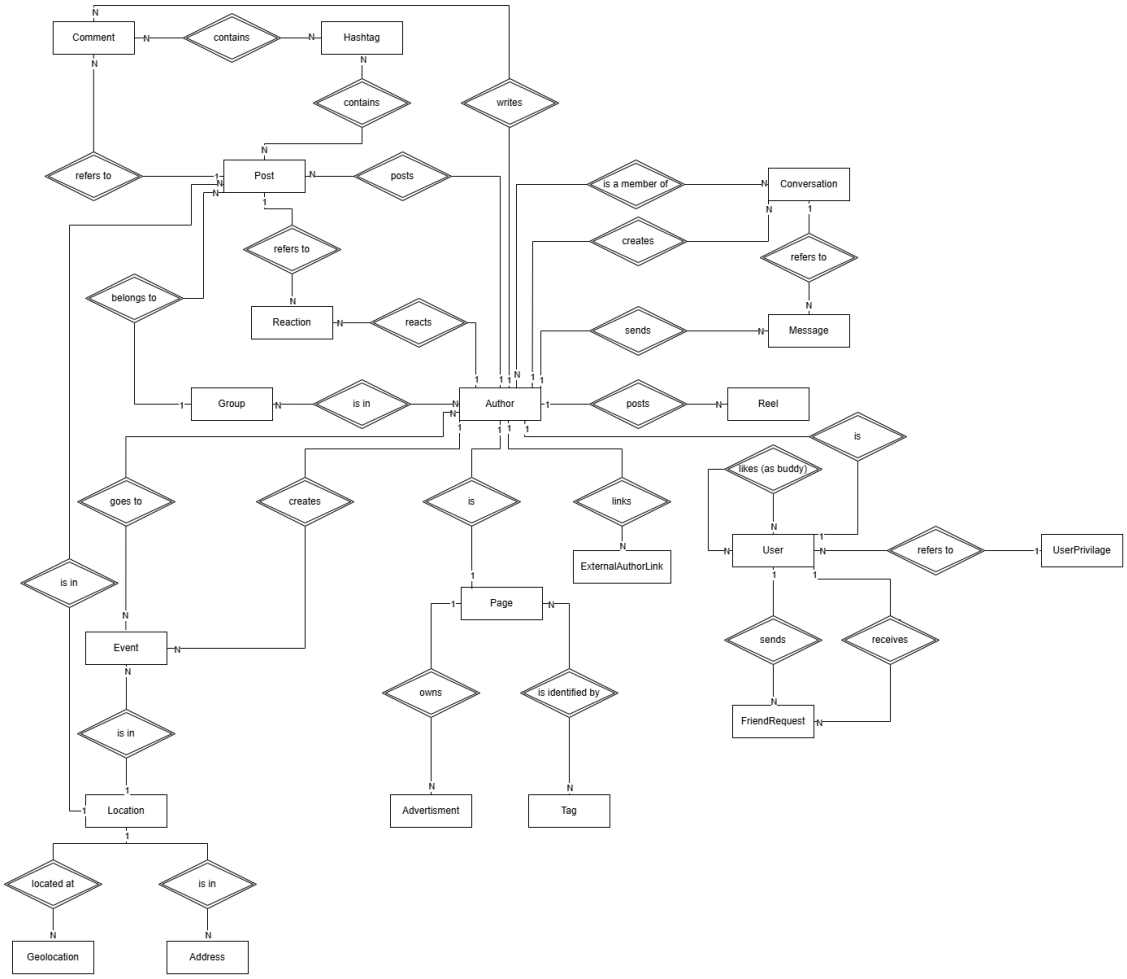
8.2 Update schematu relacji

Na podstawie nowego ERD'a 11 stary schemat relacji został uaktualniony, zostały usunięte błędy z poprzedniego oraz zwiększona czytelność.

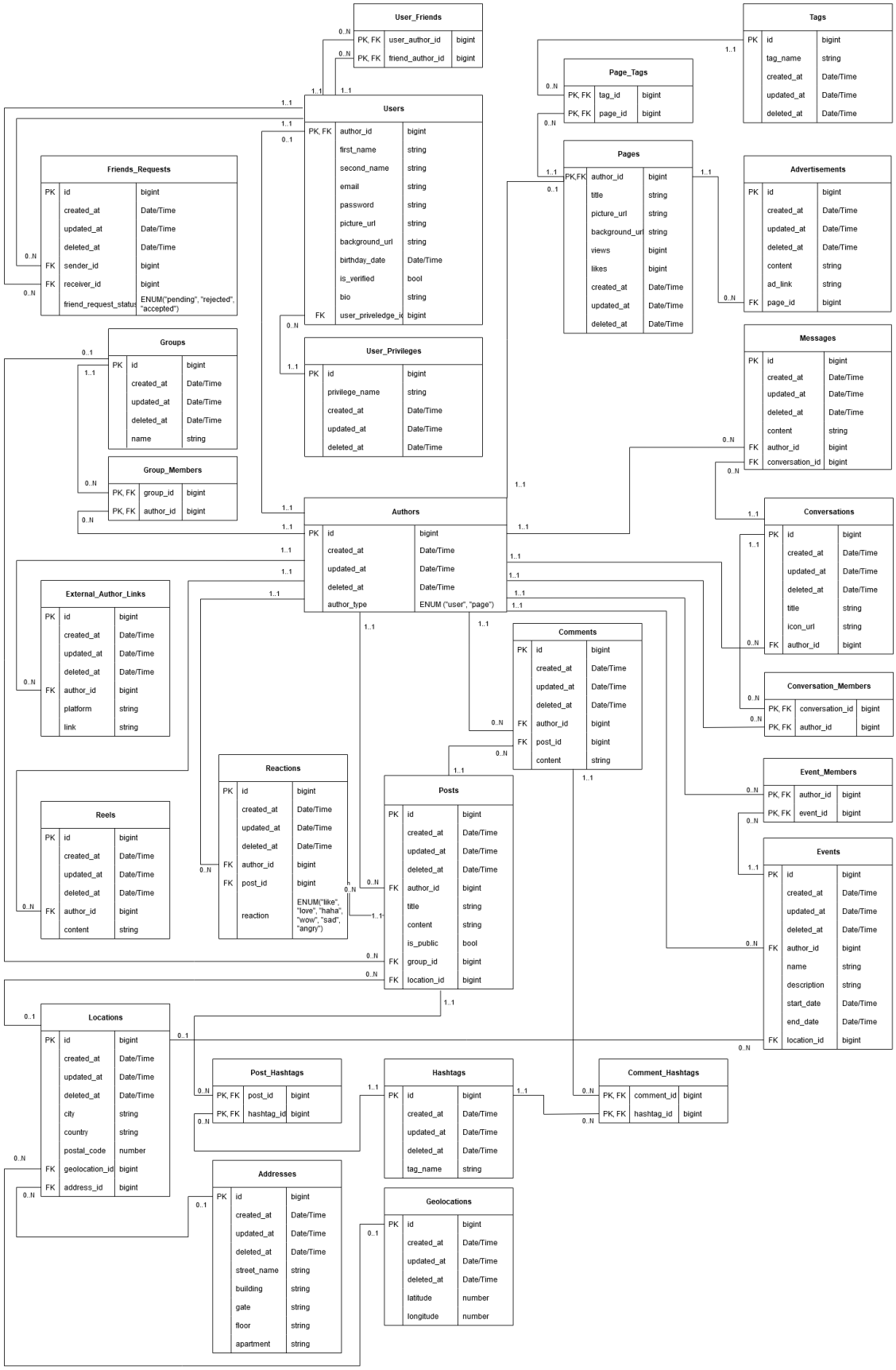
8.3 Weryfikacja i aktualizacja więzów integralności

Aby zapewnić integralność więzłów upewniliśmy się, że:

- Każda encja ma unikalny identyfikator `PK`, który jest unikalny i nie może być pusty
- Dla encji asocjacyjnych kluczem głównym jest para kluczy obcych, których dotyczy ta encja (np. `Group Members` identyfikowana jest prze `Group ID` i `Author ID`)
- W przypadku relacji `1-N` i `1-1` uwzględniamy `FK` jednej z encji w atrybutach drugiej z nich
- Zastosowanie `usuwania kaskadowego` sprawia, że rekordy podrzędne usuną się automatycznie, gdy usunięty zostanie rekord nadrzędny (np. gdy zostanie usunięty `Author` usunięte zostaną też jego `Posty`, a co za tym idzie też `Komentarze` i `Reakcje` dotyczące tych postów)



Rysunek 10: Zaktualizowany ERD



Rysunek 11: Poprawiony schemat relacji bazy danych serwisu społecznościowego

9 Etap 9: Faza logiczna II

9.1 Przetworzenie struktury bazy danych sprzed modyfikacji do zgodnej z nowymi wymaganiami

Naszą bazę ulepszyliśmy w taki sposób:

- dodaliśmy user_followers - tabela asocjacyjna, która przechowuje informacje o tym, kto obserwuje kogo
- article, w tym section
- przenieśliśmy niektóre dane z tabeli Location do Address

Dodaliśmy możliwość śledzenia użytkowników, którzy obserwują danego użytkownika. Dzięki temu użytkownik może śledzić swoich ulubionych autorów.

Nasi użytkownicy mogą tworzyć długie artykuły składające się z wielu sekcji (które mają nagłówek oraz treść).

Oraz teraz geoloakalizacja jest dodatkowo wyrażana przez Point z PostGIS, pozwoli to nam na wykonanie zapytania o np. wydarzenia w promieniu n metrów. Przenieśliśmy również część danych z Location do Address, aby ułatwić zarządzanie danymi.

Opracowaliśmy więc skrypt DDL przetwarzający strukturę starej bazy danych do nowej.

```
    BEGIN;

DROP TABLE IF EXISTS public.user_followed CASCADE;

DROP TABLE IF EXISTS public.article_hashtags CASCADE;

DROP TABLE IF EXISTS public.articles CASCADE;

DROP TABLE IF EXISTS public.sections CASCADE;

ALTER TABLE
    IF EXISTS public.addresses DROP COLUMN IF EXISTS city;

ALTER TABLE
    IF EXISTS public.addresses DROP COLUMN IF EXISTS country;

ALTER TABLE
    IF EXISTS public.addresses DROP COLUMN IF EXISTS postal_code;

ALTER TABLE
    IF EXISTS public.geolocations DROP COLUMN IF EXISTS geom;

ALTER TABLE
    IF EXISTS public.locations
ADD
    COLUMN city character varying(100) COLLATE pg_catalog."default";

ALTER TABLE
    IF EXISTS public.locations
ADD
    COLUMN country character varying(100) COLLATE pg_catalog."default";

ALTER TABLE
    IF EXISTS public.locations
ADD
    COLUMN postal_code character varying(20) COLLATE pg_catalog."default";

DROP SEQUENCE IF EXISTS public.sections_id_seq;
```

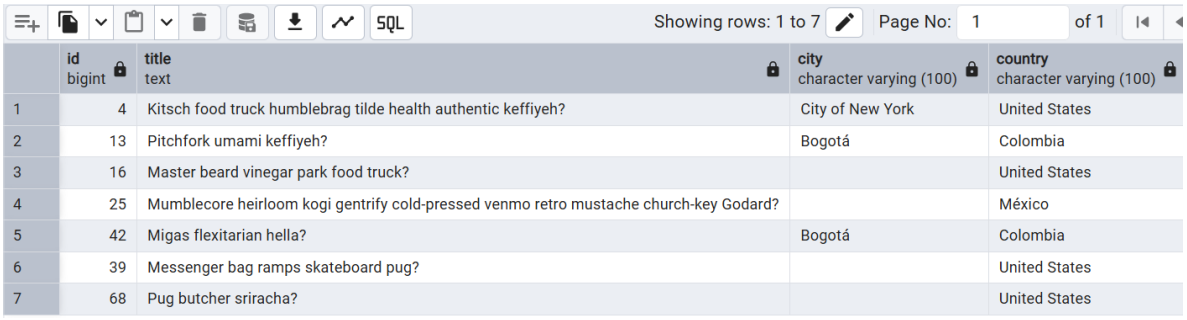
```
DROP SEQUENCE IF EXISTS public.articles_id_seq;

END;
```

9.2 Przykładowe zapytania

- 10 najdłuższych artykułów z ostatnich 30dni
- Artykuły najbardziej popularnego autora
- Wydarzenia w promieniu 100km
- Autorzy, którzy napisali najwięcej artykułów
- Posty z 5 najbliższych lokalizacji
- Użytkownik z największą liczbą obserwujących

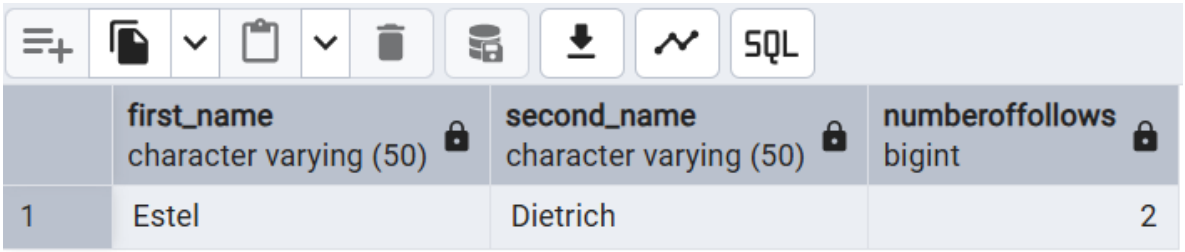
	title character varying (255)	numberofsections bigint
1	Literally neutra slow-carb gentrify mixtape skateboard?	10
2	Pinterest try-hard lo-fi?	10
3	Pug fingerstache truffaut bitters?	10
4	Whatever blog vice butcher ugh cray lo-fi humblebrag paleo tousled?	10
5	Artisan franzen direct trade 3 wolf moon?	10
6	Viral bespoke kogi cray vinyl tofu drinking you probably haven't heard of them kinfolk chicharrones?	10
7	Ethical cold-pressed quinoa sartorial you probably haven't heard of them kogi kale chips letterpress williamsburg bushwick?	10
8	Roof offal locavore kickstarter blue bottle?	10
9	Vice messenger bag helvetica knausgaard viral Thundercats kogi?	10
10	Irony readymade distillery art party single-origin coffee?	10

Rysunek 12: Wynik kwerendy wyszukującej 10 najdłuższych artykułów z ostatnich 30dni



	id bigint	title text	city character varying (100)	country character varying (100)
1	4	Kitsch food truck humblebrag tilde health authentic keffiyeh?	City of New York	United States
2	13	Pitchfork umami keffiyeh?	Bogotá	Colombia
3	16	Master beard vinegar park food truck?		United States
4	25	Mumblecore heirloom kogi gentrify cold-pressed venmo retro mustache church-key Godard?		México
5	42	Migas flexitarian hella?	Bogotá	Colombia
6	39	Messenger bag ramps skateboard pug?		United States
7	68	Pug butcher sriracha?		United States

Rysunek 13: Wynik kwerendy wyszukującej posty z najbliższych 5 lokalizacji



Rysunek 14: Wynik kwerendy wyszukującej użytkownika z największą ilością obserwujących

10 Etap 10: Nierelacyjne bazy danych - faza wstępna

Nasz wybór padł na **MongoDB**, które według nas najlepiej sprawdzi się przy okazji tworzenia bazy dla serwisu społecznościowego. Do komunikacji z bazą wykorzystamy *Pythona*.

Najbardziej optymalnym rozwiązaniem byłoby rozdzielenie danych między różne typy baz. Niektóre dane najlepiej przechowywać w bazie relacyjnej, inne w nierelacyjnej - na przykład dokumentowej lub grafowej - w zależności od ich struktury i zastosowania.

Zalety	Wady
Skalowanie horyzontalne	Brak zdefiniowanego schematu jaki jest w SQL
Wysoka wydajność odczytu i zapisu	Denormalizacja niektórych tabel
Obsługa ACID	Duży ruch = duże koszty
Dobre wsparcie dla technologii np. Python	

Tabela 1: Plusy i minusy MongoDB

Założenia (podobnie jak w przypadku relacyjnej bazy): chcemy aby baza mogła być wdrożona w jakiś serwis społecznościowy oraz aby miała dobrą wydajność, ponieważ wpływa to na doświadczenie użytkownika.

Ograniczenia: naszym zdaniem, tak jak poprzednio, ogranicza nas to, że baza musi być w całości w NoSQL. Tymczasem - jak wcześniej wspomniano - najlepiej stworzyć bazy dedykowane dla danych tabel. Problematyczne może być tworzenie `constraints`, ponieważ bazy NoSQL nie wymagają od nas tworzenia schematów danych. W tym wypadku będziemy mieli **kolekcje** zamiast tabel i **dokumenty** zamiast wierszy. Oznacza to, że będziemy musieli zadbać sami o **integralność danych**.

11 Etap 11 - Faza konceptualna i fizyczna

11.1 Definicja i wdrożenie struktur przechowywania danych w wybranej technologii nierelacyjnej.

Wdrożyliśmy odpowiedniki struktur SQL w MongoDB (NoSQL). W bazie danych MongoDB nie ma tabel, a dokumenty, które są przechowywane w kolekcjach. Dokumenty są przechowywane w formacie JSON, co pozwala na przechowywanie zagnieżdżonych obiektów. Ten bardziej elastyczny sposób przechowywania danych pozwolił nam między innymi na przechowywanie User'a i Page'a w bardziej wygodny sposób.

Definiować związki można przez dodanie listy powiązanych `id` z obiektem lub przez denormalizację. Wybór jest zależny od tego, jak będą pobierane dane. Na przykład, kiedy mamy komentarz, lepiej będzie dodać autora komentarza jako mały obiekt z jego najważniejszymi danymi, takimi jak:

- username lub imie, nazwisko albo email
- zdjęcie

Umieszczanie listy interesujących nas `id` w obiekcie przyda się, jeśli dane będą pobierane z serwera. Na przykład mamy zakładke z requestami do znajomych, wchodzimy w nią i przed renderem widoku pobieramy dane z API, wybierając kokretne obiekty z bazy. Warunkiem jest to, że user jest jakoś przechowywany.

Wykorzystane mechanizmy zapewnienia spójności:

- spójność ostateczna: w międzyczasie mogą wystąpić niespójności, ale w niedalekiej przyszłości będą one spójne
- spójność natychmiastowa: wszystkie zmiany są od razu aplikowane na wszystkich serwerach
- spójność przez większość: jeśli większość serwerów powie, że dane są w porządku, to wtedy dane są spójne

Z tych mechanizmów wynika problem rozproszenia, tutaj pojawia się twierdzenie CAP, mówiące, że w systemie rozproszonym można spełnić maksymalnie 2 z 3 warunków:

- C - consistent (spójność)
- A - available (dostępność)
- P - partition (tolerancja na partycjonowanie)

Nie ma jednoznaczenej odpowiedzi, które najlepiej wybrać z tych dwóch, ponieważ jest to zależne od przeznaczenia bazy.

> "Systemy baz danych zaprojektowane z tradycyjnymi gwarancjami ACID, takimi jak RDBMS, wybierają spójność nad dostępność, natomiast systemy zaprojektowane wokół filozofii BASE, wspólne w ruchu NoSQL, na przykład, wybierają dostępność zamiast spójności [**?**]." *CAP_theorem, en.wikipedia.org*

ACID (Atomicity, Consistency, Isolation, Durability) to dobrze znany z tradycyjnych baz danych model, który zapewnia pełną spójność danych. W przypadku NoSQL, zamiast ACID, mówimy o BASE (Basically Available, Soft state, Eventually consistent), który skupia się bardziej na dostępności danych.

Cechy	**ACID**	**BASE**
Spójność danych	Natychmiastowa	Ostateczna
Dostępność	Ograniczona w razie problemów	Zawsze dostępne
Wydajność	Mniejsza, szczególnie w skali	Wyższa, dzięki kompromisom
Zastosowanie	Bankowość, finanse, systemy ERP	Media społecznościowe, big data

Tabela 2: Porównanie paradygmatów ACID i BASE

Model Base zdecydowanie bardziej pasuje do dynamicznej struktury serwisów społecznościowych, gdzie dostępność danych jest kluczowa, a spójność danych nie jest tak ważna.

11.2 Prezentacja przykładowych zapytań

Poniżej podaliśmy odpowiedniki komend SQL w MongoDB (NoSQL).

- SELECT * FROM users → db.users.find()
- SELECT * FROM users WHERE age < 18 → db.users.find(age: $lt: 18)
- INSERT INTO users (krotka) VALUES (dane) → db.users.insertOne(dane: "w json")
- CREATE INDEX idx_deleted_at ON users (deleted_at) → db.users.createIndex(deleted_at: 1)

12 Etap 12: Faza fizyczna

Standardowo kod można zobaczyć w repozytorium na gicie: social media db. Skrypt łączący się z bazą danych przechowywaną w chmurze i generujący rekordy do kolekcji to main.py. Wykorzystuje on funkcje napisane w seeder.py.

Mając wcześniej napisaną strukturę modeli przy użyciu pydantic należało napisać funkcje generujące dane do bazy. Można je było podzielić na dwa rodzaje:

1. Funkcje generujące rekordy do kolekcji.
2. Funkcje generujące rekordy dla wszystkich rekordów w danej kolekcji, przykładowo dodawanie konwersacji dla każdego użytkownika.

Zapełnianie bazy danych dużą ilością danych zajmuje stosunkowo dużo czasu, może to być spowodowane przez wywołania które tworzą listy użytkowników dla np. eventów, innym podejrzeniem może być nieoptymalnie napisany kod.

Język: Python
Generowanie sztucznych danych: Faker

Literatura