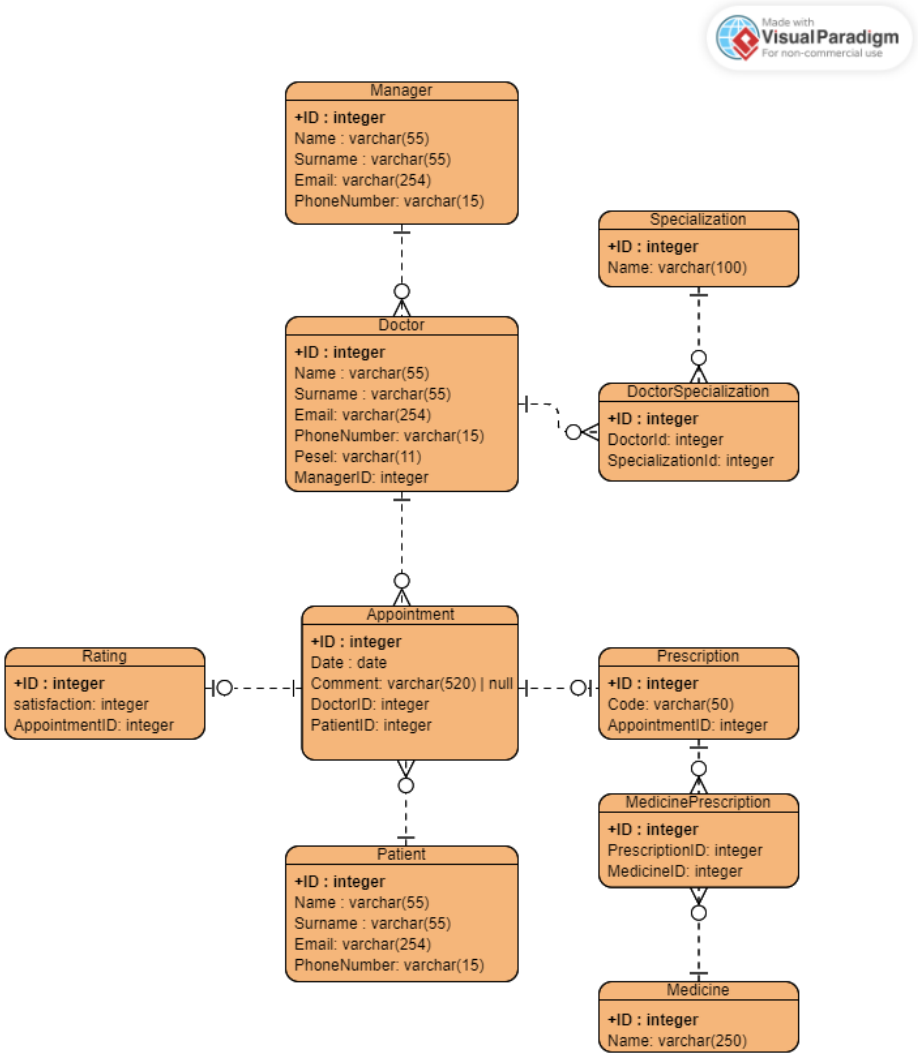
Zaawansowane Systemy Baz Danych – Etap 1 „Przychodnia”

Michał Ankiersztajn 311171

Części 1-4

W plikach znajduje się Case Study, realnie PESEL powinien znaleźć się na recepcie, jednak, aby nie złamać 3NF nie jest tam umieszczony. Będzie można się dostać do PESELu pacjenta poprzez wizytę.

Załączam tu poglądowo schemat z zaktualizowanymi wielkościami varchar:



5. Decyzje

Uwaga załączony razem z plikiem PDF jest projekt, poniższe decyzje dotyczą pliku **SQl/init.sql**, który jest odpalany przy tworzeniu bazy danych.

Zdecydowałem się zmienić wielkości varchar, dla imienia na 55, choć istnieją historycznie nazwiska, która mają 963 znaków szansa, że będzie potrzeba na ich skorzystanie jest nikła. Dodatkowo jest to skrajny procent społeczeństwa, na tyle skrajny, że nawet jeśli obetniemy ich nazwisko do 55 znaków i zostawimy imię to nadal jesteśmy ich w stanie zidentyfikować.

Email został zmieniony na 254 ponieważ jest to maksymalna długość maila.

Komentarz został wyspecyfikowany na 520, ponieważ jest to zazwyczaj wystarczająca liczba znaków, aby dodać krótki i znaczący opis.

Korzystam ze składni CREATE TABLE IF NOT EXISTS, aby uniknąć błędów przy powtórnym tworzeniu bazy danych. Jeśli tabele już istnieją to nie chcemy ich nadpisywać, a zostawić tak jak były.

Korzystam ze składni SERIAL zamiast INTEGER dla PRIMARY KEY, abym nie musiał ‘ręcznie’ zarządzać ID w tabelach, co znacznie uprości proces populacji tabel.

Korzystam ze składni CONSTRAINT name FOREIGN KEY, aby w wypadku błędu wiedzieć, który CONSTRAINT go spowodował.

Wszystkie dane przechowywane są jako NOT NULL, ponieważ wszystkie dane w wypisanych tabelach są wymagane od samego początku. Jeśli dane są opcjonalne (jak np. recepta, czy ocena) to są one w relacji 0..1 - 1

Zdecydowałem się użyć UNIQUE dla:

1. Specialization(Name) – nazwa musi być unikalna, aby uniknąć duplikacji danych. Nie chcemy mieć wielu tabel dla jednej i tej samej specjalizacji.
2. Prescription(Code) – kod recepty musi być unikalny, jeśli z jakiegoś powodu wystąpił duplikat oznacza to po prostu błąd.

Tworząc bazę danych jednocześnie tworzę scheme **public** do której podstawowo dodawane są wszystkie tabele, dzięki temu jestem w stanie ją wyeksportować komendą:

docker exec -it zsbd-postgresql_database-1 pg_dump -U admin --schema-only MediPlaceDatabase > schema.sql

Plik **schema.sql** został załączony do wglądu

6. Wypełnienie danymi

Do wypełnienia danymi skorzystałem z ChatGPT. Podałem mu schemę bazy danych i poprosiłem o wypełnienie ‘mockowymi’ danymi. Lekko zmodyfikowałem wrzucone przez

niego dane, aby baza była bardziej skomplikowana: link do rozmowy: https://chatgpt.com/share/67179ff9-8688-8011-8b19-db8e45283eee

7. Stworzenie użytkowników

Baza danych będzie posiadała 4 użytkowników.

Dostęp będzie weryfikowany za pomocą komendy:

SELECT * FROM pg_tables WHERE tableowner = 'USER';

Zrzuty z tej komendy dla każdego użytkownika są załączone razem ze sprawozdaniem.

1. **Admin**

Użytkownik, który jest podstawowo tworzony razem z bazą danych i ma dostęp do wszystkiego. Jego uprawnienia wyciągam w następujący sposób:

```
PS C:\Users\Michal\Desktop\ZSBD> docker exec -it zsbd-postgresql_database-1 psql MediPlaceDatabase -U admin
psql (17.0 (Debian 17.0-1.pgdg120+1))
Type "help" for help.

MediPlaceDatabase=# \o admin_permissions.txt
MediPlaceDatabase=# SELECT * FROM information_schema.role_table_grants WHERE grantee = 'admin';
MediPlaceDatabase=# \o
MediPlaceDatabase=# \q
PS C:\Users\Michal\Desktop\ZSBD> docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS          PORTS                    NAMES
1eeb7594f2cf   postgres:latest   "docker-entrypoint.s…"   18 minutes ago   Up 18 minutes   0.0.0.0:5432->5432/tcp   zsbd-postgresql_database-1
PS C:\Users\Michal\Desktop\ZSBD> docker cp 1eeb7594f2cf:/admin_permissions.txt .
```

Uprawnienia załączone w pliku **admin_permissions.txt**

2. **Manager**

- Pełna kontrola nad doktorami, specjalizacjami i ich połączeniem
- Wgląd w wyniki doktorów poprzez wizyty + oceny

Stworzenie użytkownika + nadanie uprawnień:

```
MediPlaceDatabase=# \du
                             List of roles
 Role name |                         Attributes
-----------+------------------------------------------------------------
 admin     | Superuser, Create role, Create DB, Replication, Bypass RLS

MediPlaceDatabase=# CREATE USER manager;
CREATE ROLE
MediPlaceDatabase=# \du
                             List of roles
 Role name |                         Attributes
-----------+------------------------------------------------------------
 admin     | Superuser, Create role, Create DB, Replication, Bypass RLS
 manager   |

MediPlaceDatabase=# GRANT SELECT, INSERT, UPDATE, DELETE ON Doctor to manager;
GRANT
MediPlaceDatabase=# GRANT SELECT, INSERT, UPDATE, DELETE ON Specialization to manager;
GRANT
MediPlaceDatabase=# GRANT SELECT, INSERT, UPDATE, DELETE ON DoctorSpecialization to manager;
GRANT
MediPlaceDatabase=# GRANT SELECT ON Appointment to manager;
GRANT
MediPlaceDatabase=# GRANT SELECT ON Rating to manager;
GRANT
```

Zrzut uprawnień:

```
PS C:\Users\Michal\Desktop\ZSBD> docker exec -it zsbd-postgresql_database-1 psql MediPlaceDatabase -U manager
psql (17.0 (Debian 17.0-1.pgdg120+1))
Type "help" for help.

MediPlaceDatabase=> \o manager_permissions.txt
MediPlaceDatabase=> SELECT * FROM information_schema.role_table_grants WHERE grantee = 'manager';
MediPlaceDatabase=> \o
MediPlaceDatabase=> \q
PS C:\Users\Michal\Desktop\ZSBD> docker cp 1eeb7594f2cf:/manager_permissions.txt .
```

Uprawnienia załączone w pliku **manager_permissions.txt**

3. **Pacjent**

- Kontroluje ocenę wizyty
- Ma wgląd w większość tabel – nie widzi innych pacjentów i managerów

Stworzenie użytkownika:

```
MediPlaceDatabase=# \du
                             List of roles
 Role name |                         Attributes
-----------+------------------------------------------------------------
 admin     | Superuser, Create role, Create DB, Replication, Bypass RLS
 manager   |

MediPlaceDatabase=# CREATE USER patient;
CREATE ROLE
MediPlaceDatabase=# \du
                             List of roles
 Role name |                         Attributes
-----------+------------------------------------------------------------
 admin     | Superuser, Create role, Create DB, Replication, Bypass RLS
 manager   |
 patient   |
```

Nadanie uprawnień:

```
GRANT
MediPlaceDatabase=# GRANT SELECT ON Doctor, Specialization, DoctorSpecialization, Patient, Appointment, Prescription, Medicine, MedicinePrescription to patient;
GRANT
MediPlaceDatabase=# GRANT SELECT, INSERT, UPDATE, DELETE ON Rating to patient;
GRANT
```

Zrzut uprawnień analogicznie jak wcześniej, załączony w pliku **patient_permissions.txt**

4. **Doktor**

- Pełna kontrola nad wizytami
- Może dodawać leki
- Może czytać, dodawać i usuwać recepty, ale nie je modyfikować, aby zapobiec sytuacją, gdzie doktor próbuje zmienić kod recepty po tym jak pacjent już jedną odebrał. Chcemy, aby pacjent najpierw skonsultował się z doktorem za pomocą wizyty zanim otrzyma kolejną receptę.
- Może dodawać leki do recepty.
- Wgląd w większość tabel – jedynie nie widzi managerów

Stworzenie użytkownika:

```
MediPlaceDatabase=# \du
                             List of roles
 Role name |                         Attributes
-----------+------------------------------------------------------------
 admin     | Superuser, Create role, Create DB, Replication, Bypass RLS
 manager   |
 patient   |

MediPlaceDatabase=# CREATE USER doctor;
CREATE ROLE
MediPlaceDatabase=# \du
                             List of roles
 Role name |                         Attributes
-----------+------------------------------------------------------------
 admin     | Superuser, Create role, Create DB, Replication, Bypass RLS
 doctor    |
 manager   |
 patient   |
```

Nadanie uprawnień:

```
MediPlaceDatabase=# GRANT SELECT, INSERT, UPDATE, DELETE ON Appointment to Doctor;
GRANT
MediPlaceDatabase=# GRANT SELECT, INSERT ON Medicine;
ERROR:  syntax error at or near ";"
LINE 1: GRANT SELECT, INSERT ON Medicine;
                                        ^
MediPlaceDatabase=# GRANT SELECT, INSERT ON Medicine to Doctor;
GRANT
MediPlaceDatabase=# GRANT SELECT, INSERT, DELETE ON MedicinePrescription to Doctor;
GRANT
MediPlaceDatabase=# GRANT SELECT, INSERT ON MedicinePrescription to Doctor;
GRANT
MediPlaceDatabase=# GRANT SELECT ON Doctor, Specialization, DoctorSpecialization, Patient, Rating;
ERROR:  syntax error at or near ";"
LINE 1: ...ctor, Specialization, DoctorSpecialization, Patient, Rating;
MediPlaceDatabase=# GRANT SELECT ON Doctor, Specialization, DoctorSpecialization, Patient, Rating to doctor;
GRANT
```

Zrzut uprawnień analogicznie jak wcześniej, załączony w pliku **doctor_permissions.txt**

8. Dwa zapytania

1. Zdecydowałem się stworzyć zapytanie, które zwraca 5 doktorów z najwyższą średnią satysfakcją na swoich wizytach. Jest to potrzebne ze strony biznesu, aby manager był w stanie zidentyfikować dobrze prosperujących doktorów i dać im np. podwyżkę, analogicznie można sprawdzić najniższą ocenę i takich doktorów zwolnić.

Liczą się tylko doktorzy, którzy mają co najmniej 30 wizyt i 5 ocen:

```
MediPlaceDatabase=# SELECT d.name, d.surname, stats.average_rating, stats.visit_count, stats.rating_count
FROM doctor d
JOIN (
    SELECT a.doctorid, AVG(r.satisfaction) AS average_rating, COUNT(a.id) AS visit_count, COUNT(r.id) AS rating_count
    FROM appointment a
    JOIN rating r ON a.id = r.appointmentid
    GROUP BY a.doctorid
    HAVING COUNT(a.id) >= 10 AND COUNT(r.id) >= 10
) AS stats ON d.id = stats.doctorid
ORDER BY stats.average_rating DESC
LIMIT 5;
```

Moja baza ma zbyt mało danych, jeśli usunąć linijkę HAVING COUNT... to uzyskamy:

```
MediPlaceDatabase=# SELECT d.name, d.surname, stats.average_rating, stats.visit_count, stats.rating_count
FROM doctor d
JOIN (
    SELECT a.doctorid, AVG(r.satisfaction) AS average_rating, COUNT(a.id) AS visit_count, COUNT(r.id) AS rating_count
    FROM appointment a
    JOIN rating r ON a.id = r.appointmentid
    GROUP BY a.doctorid
) AS stats ON d.id = stats.doctorid
ORDER BY stats.average_rating DESC
LIMIT 5;
 name  | surname |   average_rating   | visit_count | rating_count
-------+---------+--------------------+-------------+--------------
 Bob   | Green   | 4.0000000000000000 |           1 |            1
 Alice | Brown   | 3.0000000000000000 |           2 |            2
(2 rows)

MediPlaceDatabase=#
```

Jeden z doktorów nie otrzymał jeszcze żadnej oceny, dlatego się nie wyświetla – jest to pożądane działanie.

1. Sprawdzenie przez pacjenta doktorów którzy są dostępni w zakresie dat i zależnie od specjalizacji:

```
MediPlaceDatabase=# SELECT DISTINCT d.name, d.surname FROM doctor d
JOIN (
SELECT ds.doctorID
    FROM doctorspecialization ds
    JOIN specialization s ON ds.specializationID = s.ID
    WHERE s.name IN ('Cardiology', 'Dermatology')  -- Tutaj wrzucamy liste specjalizacji
    GROUP BY ds.doctorid
    HAVING COUNT(DISTINCT s.name) >= 2 -- Tutaj wrzucamy ile specjalizacji musi się pokrywać
) AS specializedDoctors on d.id = specializedDoctors.doctorId
JOIN (
SELECT a.doctorID
    FROM Appointment a
    WHERE NOT a.date = '2024-10-22' -- Tutaj wrzucamy datę którą chcemy sprawdzić
) AS availableDoctors on d.id = availableDoctors.doctorId;
 name  | surname
-------+---------
 Alice | Brown
(1 row)

MediPlaceDatabase=#
```

Wyświetla się tylko Alice Brown jest ona jedynym doktorem z obydwoma specjalizacjami, który jest dostępny danego dnia.

9. Perspektywy

W systemie przychodni ze względu na prywatność większość z widoków nie powinien być używany wprost a przez funkcję lub query, która filtruje po ID danego doktora, managera, czy pacjenta.

1. Dla Managera:

- **doctorsPerformanceView** – wszyscy doktorów wraz z danymi ilościowymi, ile odbyli wizyt, ile wizyt dostało oceny, jaka jest średnia ocen ich wizyt

2. Dla Doktora i Pacjenta:

- **appointmentDetailsView** – szczegóły na temat wizyty wraz z oceną, kodem recepty i listą leków

```
MediPlaceDatabase=# CREATE VIEW appointmentDetailsView AS
MediPlaceDatabase-# SELECT
MediPlaceDatabase-#     a.id,
MediPlaceDatabase-#     a.date,
MediPlaceDatabase-#     a.comment,
MediPlaceDatabase-#     r.satisfaction,
MediPlaceDatabase-#     p.code AS prescription_code,
MediPlaceDatabase-#     STRING_AGG(m.name, ', ') AS medicines
MediPlaceDatabase-# FROM appointment a
MediPlaceDatabase-# LEFT JOIN rating r ON a.id = r.appointmentid
MediPlaceDatabase-# LEFT JOIN prescription p ON a.id = p.appointmentid
MediPlaceDatabase-# LEFT JOIN medicineprescription mp ON p.id = mp.prescriptionid
MediPlaceDatabase-# LEFT JOIN medicine m ON mp.medicineid = m.id
MediPlaceDatabase-# GROUP BY a.id, a.date, a.comment, a.doctorid, a.patientid, r.satisfaction, p.code;
CREATE VIEW
MediPlaceDatabase=# SELECT * FROM appointmentDetailsView;
 id |    date    |      comment       | satisfaction | prescription_code | medicines
----+------------+--------------------+--------------+-------------------+-----------
  1 | 2024-10-15 | Regular check-up   |            5 | RX12345           | Aspirin
  2 | 2024-10-18 | Skin rash          |            4 | RX67890           | Ibuprofen
  3 | 2024-10-20 | Child consultation |              |                   |
  4 | 2024-10-16 | no comment         |            1 |                   |
(4 rows)
```

Jeśli dana wizyta nie ma oceny, kodu lub leków to nie zostaną one wyświetlone.

3. Dla wszystkich:

- **doctorDetailsView** – szczegóły na temat doktora wraz z listą jego specjalizacji, liczbą odbytych wizyt i średnią oceną

```
MediPlaceDatabase=# CREATE VIEW doctorDetailsView AS
SELECT
    d.id,
    d.name,
    d.surname,
    d.email,
    d.phonenumber,
    STRING_AGG(s.name, ', ') AS specializations
FROM doctor d
LEFT JOIN doctorspecialization ds ON d.id = ds.doctorid
LEFT JOIN specialization s ON ds.specializationid = s.id
GROUP BY d.id, d.name, d.surname, d.email, d.phonenumber, d.pesel, d.managerid;
ERROR:  relation "doctordetailsview" already exists
MediPlaceDatabase=# SELECT * FROM doctorDetailsView;
 id |  name   | surname |          email           | phonenumber |     specializations
----+---------+---------+--------------------------+-------------+-------------------------
  2 | Bob     | Green   | bob.green@hospital.com   | 321321321   | Dermatology
  3 | Charlie | White   | charlie.white@hospital.com | 456456456   | Pediatrics
  1 | Alice   | Brown   | alice.brown@hospital.com   | 123123123   | Cardiology, Dermatology
(3 rows)
```

10. Indeksy

Sprawdzam, które kolumny domyślnie mają ustawione indeksy:

```
MediPlaceDatabase=# \di
                          List of relations
 Schema |            Name             | Type  | Owner |        Table
--------+-----------------------------+-------+-------+----------------------
 public | appointment_pkey            | index | admin | appointment
 public | doctor_pkey                 | index | admin | doctor
 public | doctorspecialization_pkey   | index | admin | doctorspecialization
 public | manager_pkey                | index | admin | manager
 public | medicine_pkey               | index | admin | medicine
 public | medicineprescription_pkey   | index | admin | medicineprescription
 public | patient_pkey                | index | admin | patient
 public | prescription_pkey           | index | admin | prescription
 public | rating_pkey                 | index | admin | rating
 public | specialization_name_key     | index | admin | specialization
 public | specialization_pkey         | index | admin | specialization
(11 rows)

MediPlaceDatabase=# SELECT indexname, tablename, indexdef FROM pg_indexes;
```

Przy okazji sprawdziłem też definicję indexów, wszystkie z nich korzystają z algorytmu btree. Oznacza to, że PostgreSQL domyślnie tworzy indeksy dla kluczy głównych oraz kolumn typu UNIQUE. Dodatkowo są to specjalne indeksy typu UNIQUE, które nie pozwalają na powtórki danych.

Indeksy, które warto założyć to również te na klucze obce, ponieważ są one często wykorzystywane w przeszukiwaniu(przykład dla paru kolumn):

```
MediPlaceDatabase=# CREATE INDEX fk_doctorspecialization_doctor ON doctorspecialization(DoctorID);
CREATE INDEX
MediPlaceDatabase=# CREATE INDEX fk_doctorspecialization_specialization ON doctorspecialization(SpecializationID);
CREATE INDEX
MediPlaceDatabase=# CREATE INDEX fk_appointment_doctor ON appointment(DoctorID);
CREATE INDEX
MediPlaceDatabase=# CREATE INDEX fk_appointment_patient ON appointment(PatientID);
CREATE INDEX
MediPlaceDatabase=# CREATE  INDEX fk_rating_appointment ON rating(AppointmentID);
CREATE INDEX
MediPlaceDatabase=# CREATE  INDEX fk_prescription_appointment ON prescription(AppointmentID);
CREATE INDEX
MediPlaceDatabase=# CREATE INDEX fk_medicineprescription_medicine ON medicineprescription(MedicineID);
CREATE INDEX
MediPlaceDatabase=# CREATE INDEX fk_medicineprescription_prescription ON medicineprescription(PrescriptionID);
CREATE INDEX
MediPlaceDatabase=# SELECT indexname, tablename, indexdef FROM pg_indexes;
MediPlaceDatabase=#
```

Wszystkie te indeksy korzystają z algorytmu btree.

Kolejną wartościową kolumną z indeksami są daty, w tym przypadku występuje tylko data wizyty.

```
MediPlaceDatabase=# CREATE INDEX appointment_date ON appointment(Date);
CREATE INDEX
MediPlaceDatabase=# _
```

Teoretycznie moglibyśmy założyć indeksy na imię i nazwisko pacjenta oraz doktora, bo są to typowo często przeszukiwane kolumny dla systemów. **Jednak z założenia funkcjonalności tego systemu nie ma to większego sensu** (nie ma planowanego dostępu do wyszukiwarki doktorów/pacjentów, a jedynie dostęp do nich poprzez **id wizyty lub specjalizacje**).

W aplikacji nie ma też miejsca na indeksy kompozytowe, żadna z tabel nie ma być docelowo przeszukiwana po 2 lub więcej kolumnach na raz.

Test efektywności indeksów, przetestuję 3 rodzaje indeksów. Najpierw wrzucę po 15-30 tyś. Mockowych rekordów to tabel z których będę korzystał, przykład z pacjentem:

```
MediPlaceDatabase=# INSERT INTO Patient (Name, Surname, Email, PhoneNumber)
SELECT
  'Patient_' || i,
  'Surname_' || i,
  'patient' || i || '@hospital.com',
  '876543210' || i % 10
FROM generate_series(1, 15000) AS i;
INSERT 0 15000
```

Do testów będę urachamiał SELECT 1_000_000 razy z odpalonym timingiem, aby uzyskać dokładniejsze dane statystyczne:

```
FROM generate_series(1, 1000000) AS iteration,
(SELECT DISTINCT d.name, d.surname FROM doctor d
JOIN (
SELECT ds.doctorID
FROM doctorspecialization ds
JOIN specialization s ON ds.specializationID = s.ID
WHERE s.name IN ('Cardiology', 'Dermatology')
GROUP BY ds.doctorID
) AS specializedDoctors on d.id = specializedDoctors.doctorID
JOIN (
SELECT a.doctorID
FROM appointment a
WHERE NOT a.date = '2024-10-22'
) AS availableDoctors on d.id = availableDoctors.doctorId)
AS query;
```

Jest wiele różnych typów indeksów, jednak ja zdecydowałem się przetestować to na tych 3:

Brak indeksów:

Tu jestem w stanie usunąć indeks na klucze obce oraz datę ze względu na ustawione constrainty.

```
Time: 730.251 ms
MediPlaceDatabase=# SELECT query.*
FROM generate_series(1, 1000000) AS iteration,
(SELECT DISTINCT d.name, d.surname FROM doctor d
JOIN (
SELECT ds.doctorID
FROM doctorspecialization ds
JOIN specialization s ON ds.specializationID = s.ID
WHERE s.name IN ('Cardiology', 'Dermatology')
GROUP BY ds.doctorID
) AS specializedDoctors on d.id = specializedDoctors.doctorID
JOIN (
SELECT a.doctorID
FROM appointment a
WHERE NOT a.date = '2024-10-22'
) AS availableDoctors on d.id = availableDoctors.doctorId)
AS query;
Time: 736.448 ms
```

Btree:

```
MediPlaceDatabase=# SELECT query.*
FROM generate_series(1, 1000000) AS iteration,
(SELECT DISTINCT d.name, d.surname FROM doctor d
JOIN (
SELECT ds.doctorID
FROM doctorspecialization ds
JOIN specialization s ON ds.specializationID = s.ID
WHERE s.name IN ('Cardiology', 'Dermatology')
GROUP BY ds.doctorID
) AS specializedDoctors on d.id = specializedDoctors.doctorID
JOIN (
SELECT a.doctorID
FROM appointment a
WHERE NOT a.date = '2024-10-22'
) AS availableDoctors on d.id = availableDoctors.doctorId)
AS query;
Time: 728.662 ms
```

Btree w tym wypadku jest niewiele szybsze od braku indeksów.

HASH:

```
MediPlaceDatabase=# SELECT query.*
FROM generate_series(1, 1000000) AS iteration,
(SELECT DISTINCT d.name, d.surname FROM doctor d
JOIN (
SELECT ds.doctorID
FROM doctorspecialization ds
JOIN specialization s ON ds.specializationID = s.ID
WHERE s.name IN ('Cardiology', 'Dermatology')
GROUP BY ds.doctorID
) AS specializedDoctors on d.id = specializedDoctors.doctorID
JOIN (
SELECT a.doctorID
FROM appointment a
WHERE NOT a.date = '2024-10-22'
) AS availableDoctors on d.id = availableDoctors.doctorId)
AS query;
Time: 728.065 ms
```

HASH jest bardzo podobne do btree. Dlaczego? Wynika to z tego na jakich operatorach logicznych działają dane algorytmy. Btree działa na (<, <=, =, >=, >), a Hash na (=). Dla

danego zapytania korzystamy jedynie z (=), co oznacza, że algorytmy Hash i Btree powinny uzyskać bliźniaczy wynik.

Dla danego zapytania najlepiej działa

Zrzut wszystkich indeksów w bazie danych wraz ze szczegółami znajduje się w pliku **indexes.txt**

Bibliografia:

https://www.postgresqltutorial.com/

https://stackoverflow.com/questions/14486241/postgresql-export-the-schema-of-a-database

https://stackoverflow.com/questions/5331320/how-to-save-psql-output-to-a-file

https://stackoverflow.com/questions/22049212/copying-files-from-docker-container-to-host

https://stackoverflow.com/questions/760210/how-do-you-create-a-read-only-user-in-postgresql

https://chatgpt.com/share/67179ff9-8688-8011-8b19-db8e45283eee

https://en.wikipedia.org/wiki/Hubert_Blaine_Wolfeschlegelsteinhausenbergerdorff_Sr.

https://stackoverflow.com/questions/1199190/what-is-the-optimal-length-for-an-email-address-in-a-database

https://simplebackups.com/blog/postgresql-how-to-list-indexes/

https://www.postgresql.org/docs/current/sql-createindex.html

https://www.postgresql.org/docs/current/indexes-types.html