27 październik 2024

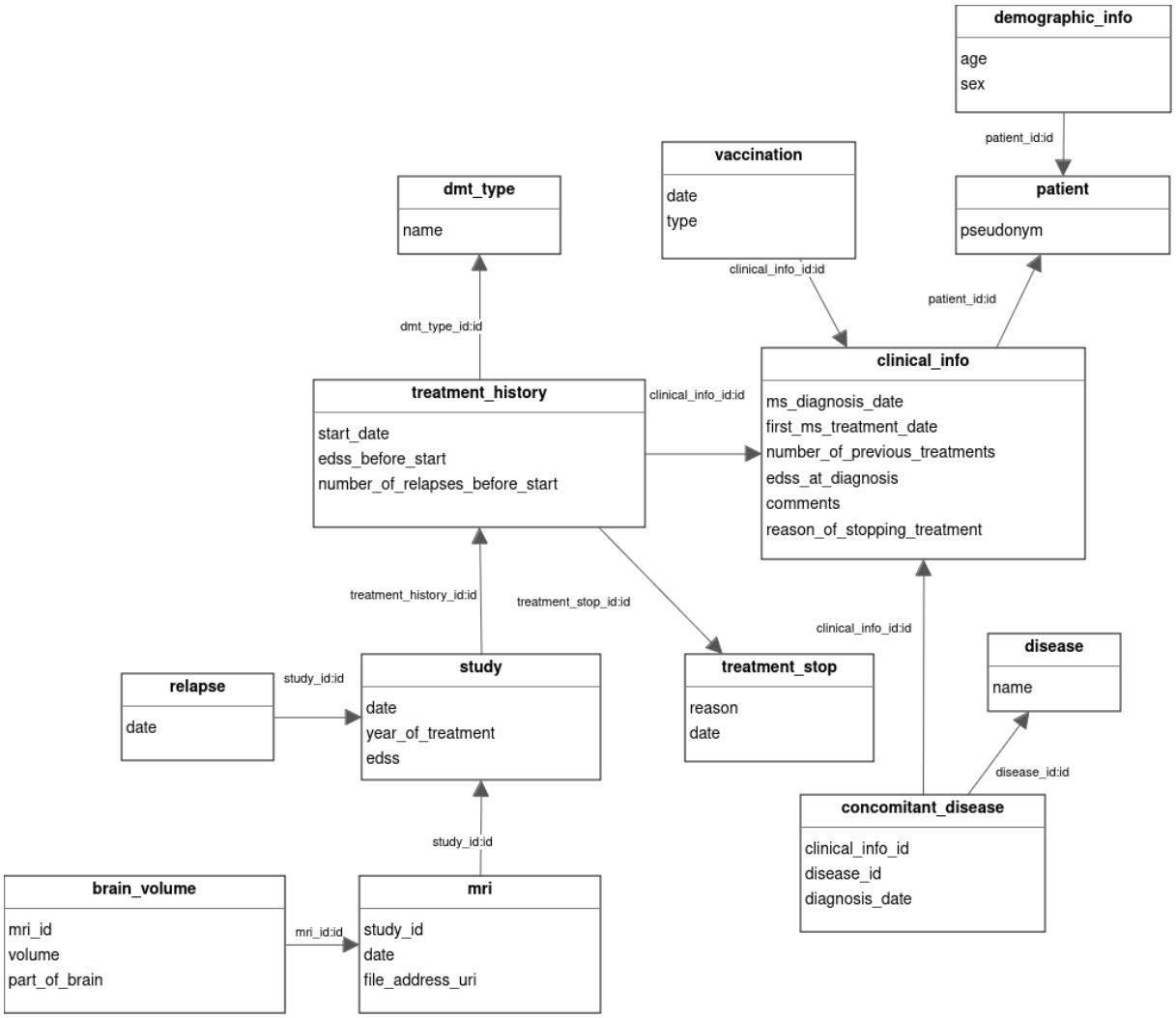
Zaawansowane systemy baz danych - Etap 1

Sebastian Górka

Spis treści

1 Schemat logiczny

Względem proponowanej w case study struktury zmieniło się to, że encja reprezentująca lekarza została usunięta. W zamian za nią powstało kilka nowych tabel. Między innymi informacje o chorobach współtowarzyszących, rzutach choroby oraz historii leczenia pacjenta danym lekiem.



2 Wygenerowanie danych

Dane do bazy zostały wygenerowane przy użyciu języka SQL. Dzięki temu, że dane medyczne nie posiadają w sobie trudnych do wygenerowania danych takich jak dane osobowe, to możliwe było wygenerowanie automatyczne dużej ilości danych, które mają sens za pomocą złożonych procedur. Przykładowa procedura widoczna jest na listingu 1.

```
DO
$$
  DECLARE
    _mri_id integer;
    _volume double precision;
    _part_of_brain patient.part_of_brain;
  BEGIN
    FOR _mri_id IN SELECT id FROM patient.mri
    LOOP
      FOR _part_of_brain IN SELECT unnest(
        enum_range(NULL::patient.part_of_brain))::patient.part_of_brain
      LOOP
        _volume := 200 + floor(random() * 800);
        INSERT INTO patient.brain_volume (mri_id, volume, part_of_brain)
        VALUES (_mri_id, _volume, _part_of_brain);
      END LOOP;
    END LOOP;
  END;
$$;
```

Program 1: Kod generujący dane o objętościach fragmentów mózgu

3 Użytkownicy

Tak jak zaznaczone zostało w case study, przewidzianych jest trzech (nie licząc super-usera `postgres`) użytkowników: analyst, importer i migration. Testy uprawnień użytkowników przeprowadzane są automatycznie, za pomocą konteneru testującego, który uruchamia się po przeprowadzeniu migracji przez Liquibase. Wycinek pliku testującego uprawnienia jednego z użytkowników widoczny jest na listingu 2.

```
PGPASSWORD=$ANALYST_PASSWORD psql -h $DB_HOSTNAME -U $ANALYST_USER -d ms -c "DELETE FROM
$schema.test_analyst_dml WHERE name = 'test2';" > /dev/null 2>&1
if [ $? -ne 0 ]; then
  echo "PASSED: $ANALYST_USER cannot delete from $schema schema"
else
  echo "FAIL: $ANALYST_USER can delete from $schema schema"
  exit 1
fi

PGPASSWORD=$ANALYST_PASSWORD psql -h $DB_HOSTNAME -U $ANALYST_USER -d ms -c "SELECT * FROM
$schema.test_analyst_dml;" > /dev/null 2>&1
if [ $? -eq 0 ]; then
  echo "PASSED: $ANALYST_USER can query from $schema schema"
else
  echo "FAIL: $ANALYST_USER cannot query from $schema schema"
  exit 1
fi
```

Przykładowy wydruk z programu testującego do konsoli można zobaczyć na listingu 3.

```
PASSED migration cannot create table in public schema
PASSED migration can create table in liquibase schema
PASSED: migration can create table in patient schema
PASSED: migration can insert into patient schema
PASSED: migration can update in patient schema
PASSED: migration can delete from patient schema
PASSED: migration can query from patient schema
PASSED: migration can drop table in patient schema
PASSED: analyst cannot create table in public schema
PASSED: analyst cannot create table in liquibase schema
PASSED: analyst cannot create table in patient schema
PASSED: analyst cannot drop table in patient schema
PASSED: analyst cannot insert into patient schema
PASSED: analyst cannot delete from patient schema
PASSED: analyst can query from patient schema
PASSED: importer cannot create table in public schema
PASSED: importer cannot create table in liquibase schema
PASSED: importer cannot create table in patient schema
PASSED: importer cannot drop table in patient schema
PASSED: importer can insert into patient schema
PASSED importer can update in patient schema
PASSED: importer cannot delete from patient schema
PASSED: importer can query from patient schema
permissions tests passed
```

```yaml
services:

database:
  extends:
    service: database
    file: compose.yaml

migration:
  extends:
    service: migration
    file: compose.yaml

test:
  build:
    context: ./test
    dockerfile: Dockerfile
  environment:
    DB_HOSTNAME: database
    MIGRATION_USER: migration
    MIGRATION_PASSWORD: $MIGRATION_PASSWORD
    ANALYST_USER: analyst
    ANALYST_PASSWORD: $ANALYST_PASSWORD
    IMPORTER_USER: importer
    IMPORTER_PASSWORD: $IMPORTER_PASSWORD
    POSTGRES_PASSWORD: $POSTGRES_PASSWORD
  depends_on:
    migration:
      condition: service_completed_successfully
```

Program 4: compose.test.yaml - plik, który rozszerza podstawowy plik compose.yaml o kontener testujący konfigurację bazy danych

4 Zapytania

Przykładowe zapytania, które można wykonać na utworzonej bazie są wyliczone poniżej.

```sql
WITH latest_study AS (
    SELECT
        treatment_history_id,
        MAX(date) AS latest_date
    FROM patient.study
    GROUP BY treatment_history_id
),
study_with_rank AS (
    SELECT
        s.treatment_history_id,
        s.date,
        s.edss,
        ROW_NUMBER() OVER (PARTITION BY s.treatment_history_id ORDER BY s.date DESC) AS rnk
    FROM patient.study s
),
average_edss AS (
    SELECT
        s.treatment_history_id,
        AVG(s.edss) AS avg_edss_last_3_years
    FROM patient.study s
    WHERE s.date >= CURRENT_DATE - INTERVAL '3 years'
    GROUP BY s.treatment_history_id
)
SELECT
    p.id AS patient_id,
    p.pseudonym,
    d.age,
    d.sex,
    c.ms_diagnosis_date,
    c.first_ms_treatment_date,
    c.number_of_previous_treatments,
    c.edss_at_diagnosis,
    ls.latest_date AS latest_study_date,
    swr.edss AS latest_edss,
    ae.avg_edss_last_3_years
FROM patient.patient p
JOIN patient.demographic_info d ON p.id = d.patient_id
JOIN patient.clinical_info c ON p.id = c.patient_id
LEFT JOIN latest_study ls ON c.patient_id = ls.treatment_history_id
LEFT JOIN study_with_rank swr ON ls.treatment_history_id = swr.treatment_history_id AND swr.rnk
= 1
LEFT JOIN average_edss ae ON c.patient_id = ae.treatment_history_id
WHERE d.age > 30
ORDER BY p.id;
```

Program 5: Pobranie podstawowych informacji oraz ostatniego badania pacjentów starszych niż 30 lat

```sql
WITH avg_edss AS (
    SELECT
        treatment_history_id,
        AVG(edss) AS avg_edss
    FROM patient.study
    GROUP BY treatment_history_id
)
SELECT
    p.id AS patient_id,
    p.pseudonym,
    d.age,
    d.sex,
    a.avg_edss
FROM patient.patient p
JOIN patient.demographic_info d ON p.id = d.patient_id
JOIN avg_edss a ON p.id = a.treatment_history_id
JOIN patient.clinical_info c ON p.id = c.patient_id
JOIN patient.treatment_history t ON c.id = t.clinical_info_id
GROUP BY p.id, d.age, d.age, p.pseudonym, p.id, d.sex, a.avg_edss
HAVING COUNT(t.id) > 2
```

Program 6: Zapytanie o pacjentów, którzy leczyli się co najmniej dwoma różnymi rodzajami leków na stwardnienie rozsiane

5 Perspektywy

W tej chwili najbardziej potrzebną perspektywą jest ta ukazująca wszystkie dane wolumetryczne i kliniczne pacjenta. Pobiera ona informacje na temat objętości wszystkich części mózgu i dla każdego pacjenta, terapii i roku terapii zwraca jeden wiersz widoczna jest na listingu 7.

```sql
DROP VIEW IF EXISTS patient.patient_volumes;
CREATE VIEW patient.patient_volumes AS
    SELECT
        clinical.patient_id,
        clinical.pseudonym,
        clinical.treatment_id,
        clinical.year_of_treatment,
        clinical.edss,
        clinical.relapses,
        clinical.dmt_type,
        MAX(CASE WHEN bv.part_of_brain = 'cerebrum' THEN bv.volume END) AS cerebrum,
        MAX(CASE WHEN bv.part_of_brain = 'frontal_lobe' THEN bv.volume END) AS frontal_lobe,
        MAX(CASE WHEN bv.part_of_brain = 'parietal_lobe' THEN bv.volume END) AS parietal_lobe,
        MAX(CASE WHEN bv.part_of_brain = 'temporal_lobe' THEN bv.volume END) AS temporal_lobe,
        MAX(CASE WHEN bv.part_of_brain = 'occipital_lobe' THEN bv.volume END) AS occipital_lobe,
        MAX(CASE WHEN bv.part_of_brain = 'cerebellum' THEN bv.volume END) AS cerebellum,
        MAX(CASE WHEN bv.part_of_brain = 'brain_stem' THEN bv.volume END) AS brain_stem,
        MAX(CASE WHEN bv.part_of_brain = 'midbrain' THEN bv.volume END) AS midbrain,
        MAX(CASE WHEN bv.part_of_brain = 'pons' THEN bv.volume END) AS pons,
             MAX(CASE WHEN bv.part_of_brain = 'medulla_oblongata' THEN bv.volume END) AS
medulla_oblongata,
        MAX(CASE WHEN bv.part_of_brain = 'thalamus' THEN bv.volume END) AS thalamus,
        MAX(CASE WHEN bv.part_of_brain = 'hypothalamus' THEN bv.volume END) AS hypothalamus,
        MAX(CASE WHEN bv.part_of_brain = 'amygdala' THEN bv.volume END) AS amygdala,
        MAX(CASE WHEN bv.part_of_brain = 'hippocampus' THEN bv.volume END) AS hippocampus,
        MAX(CASE WHEN bv.part_of_brain = 'basal_ganglia' THEN bv.volume END) AS basal_ganglia,
        MAX(CASE WHEN bv.part_of_brain = 'corpus_callosum' THEN bv.volume END) AS corpus_callosum,
        MAX(CASE WHEN bv.part_of_brain = 'meninges' THEN bv.volume END) AS meninges,
            MAX(CASE WHEN bv.part_of_brain = 'cerebrospinal_fluid' THEN bv.volume END) AS
cerebrospinal_fluid,
            MAX(CASE WHEN bv.part_of_brain = 'blood_brain_barrier' THEN bv.volume END) AS
blood_brain_barrier,
        MAX(CASE WHEN bv.part_of_brain = 'neurons' THEN bv.volume END) AS neurons,
        MAX(CASE WHEN bv.part_of_brain = 'glial_cells' THEN bv.volume END) AS glial_cells
    FROM patient.brain_volume bv
    JOIN LATERAL (
        SELECT
            p.id AS patient_id,
            p.pseudonym,
            th.id AS treatment_id,
            s.year_of_treatment,
            s.edss,
            count(r.id) AS relapses,
            m.id AS mri_id,
            dmt.name AS dmt_type
        FROM patient.patient p
        JOIN patient.clinical_info ci ON p.id = ci.patient_id
        JOIN patient.treatment_history th ON ci.id = th.clinical_info_id
        JOIN patient.study s ON th.id = s.treatment_history_id
        JOIN patient.mri m ON s.id = m.study_id
        JOIN patient.dmt_type dmt ON th.dmt_type_id = dmt.id
        LEFT JOIN patient.relapse r ON s.id = r.study_id
        GROUP BY p.id, p.pseudonym, th.id, s.year_of_treatment, s.edss, m.id, dmt.name
    ) clinical ON bv.mri_id = clinical.mri_id
            GROUP BY clinical.patient_id, clinical.pseudonym, clinical.treatment_id,
clinical.year_of_treatment, clinical.edss, clinical.relapses, clinical.dmt_type;
```

Program 7: Perspektywa patient.patient_volumes

Kolejna stworzona perspektywa ukazuje zagregowane dane dotyczące konkretnych leków DMT widać ją na listingu 8.

```sql
DROP VIEW IF EXISTS patient.dmt_edss_increment;
CREATE VIEW patient.dmt_edss_increment AS
    SELECT
        AVG(last_study.edss - first_study.edss) AS edss_increment,
        dmt.name AS dmt_name
    FROM patient.treatment_history th
    JOIN LATERAL (
        SELECT edss
        FROM patient.study s
        WHERE s.treatment_history_id = th.id AND s.year_of_treatment = 0
    ) first_study ON true
    JOIN LATERAL (
        SELECT edss
        FROM patient.study s
        WHERE s.treatment_history_id = th.id
        ORDER BY s.year_of_treatment DESC
        LIMIT 1
    ) last_study ON true
    JOIN patient.dmt_type dmt ON th.dmt_type_id = dmt.id
    GROUP BY dmt.name
```

Program 8: Perspektywa patient.dmt_edss_increment

W związku z tym, że perspektywa `patient_volumes` zwraca prawie wszystkie dostępne w tej chwili dane, to nie ma potrzeby tworzyć kolejnych perspektyw, ponieważ analityka interesują jedynie serie czasowe przebiegu choroby i różnice przy przebiegach w zależności od stosowanych DMTs.

6 Indeksy

Indeksy zostały pozakładane głównie na kolumnach, które często są używane w warunkach łączenia pomiędzy tabelami (`idx_mri_study_id`, `idx_brain_volume_mri_id`), jak również na kolumnach, po których często się sortuje. Na przykład PostgreSQL udostępnia specjalny indeks B-tree, który świetnie się nadaje do zakładania na kolumny, które się w pewien sposób porządkuje.

Najbardziej popularne są cztery typy indeksów:

- główne – zakładane są na kluczach głównych
- unikalne – wymuszają unikalność na kolumnach, lub zbiorach kolumn, jeśli przy okazji są to indeksy kompozytowe
- sklastrowane – wymuszają fizyczne uporządkowanie danych na dysku – może być tylko jeden w tabeli
- niesklastrowane – może być ich wiele, ponieważ nie mają wpływu na fizyczną organizację danych na dyski, przechowują jedynie referencje do wartości przechowywanych w kolumnach