Универзитет „Св. Кирил и Методиј“ во Скопје
ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

Проект по предметот
Неструктурирани бази на податоци

Тема:

Споредба на перформанси кај неструктурирана граф база и структурирана база на податоци: Neo4j и PostgreSQL

Изработиле:
Никола Талевски 211009
Бојана Марковска 211134
Бојан Стефановски 214005
Филип Насковски 211198

Содржина

Вовед 2
Методологија 3
Инсталација 5
Импортирање на податоци 5
Прашалници 11
Добиени резултати 17
Заклучок 18
Користена литература 18

Вовед

Изборот на соодветен модел на база на податоци претставува критична одлука при дизајнирањето на апликации што користат и обработуваат големи количини податоци, особено кога се работи со податочни множества кои што содржат голем број на врски измеѓу податоците. Релационите бази на податоцу како PostgreSQL веќе подолго време се стандард за управување со табеларни податоци со добро дефинирани шеми. Сепак, со растечката потреба за анализа на меѓусебно поврзани податоци, како мрежи, хиерархии или системи за препорака, граф базите на податоци како Neo4j нудат алтернативен пристап. Овие бази ги стават врските меѓу податоците во центарот на вниманието, со фокус на нивно пребарување и препознавање, и често ги надминуваат релационите системи каде врските се клучни.

Овој проект претставува споредбена аналуза помеѓу PostgreSQL и Neo4j со цел да се оцени нивната ефикаснот и соодветност, како и нивната брзина. Со извршување на еквивалентни операции во двете бази, ја анализираме ефикасноста на пребарувањата, флексибилноста во моделирањето и леснотијата на добивање увид базиран на врските помеѓу податоците.

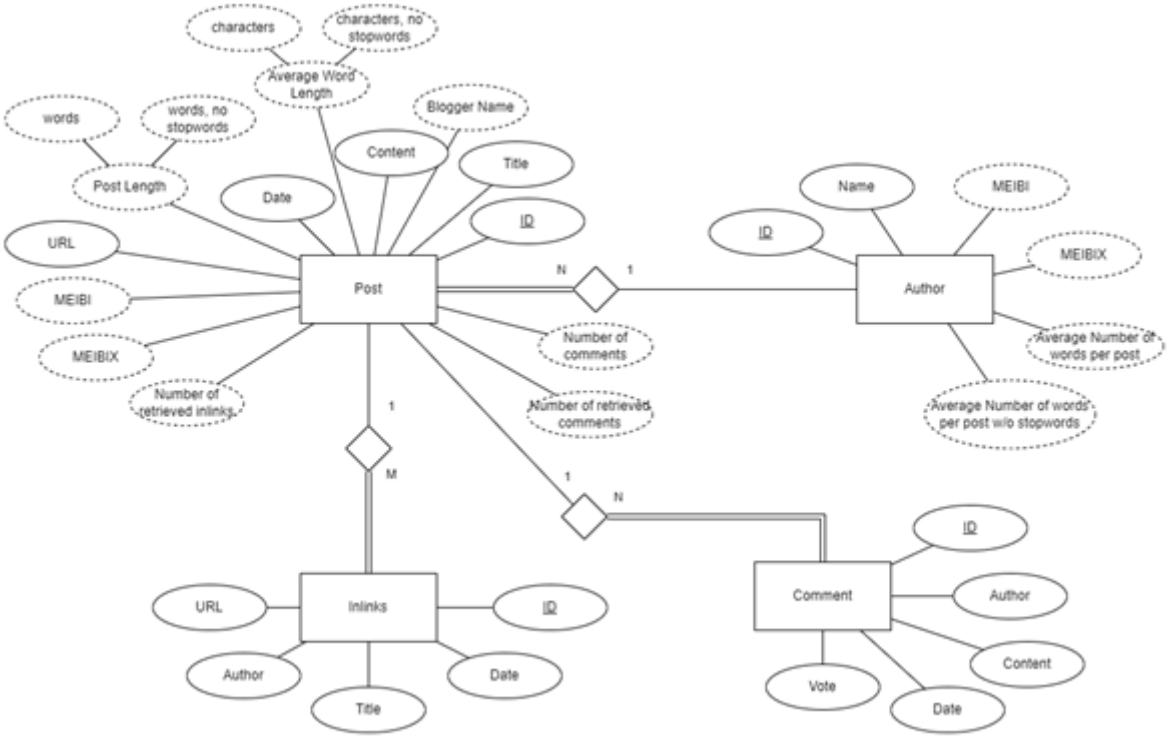
За да ја засноваме оваа споредба врз реална примена, го одбравме множеството податоци „Identifying Influential Bloggers“, кое содржи мета податоци и врски од платформата TechCrunch. Множеството вклучува блог постови, коментари, автори и референци, природно формирајќи насочен граф на интеракции. Истовремено може да се претстави и во релациска форма преку нормализирани табели, што го прави совршено множество за ова истражување.

Методологија

Множеството податоци „Identifying Influential Bloggers“ [1] претставува збир од четири CSV датотеки кои документираат блог-активности на платформата **TechCrunch** во текот на 2010 година. Податоците вклучуваат блогери, нивните објави, коментари од читатели и надворешни линкови кон самите објави, соодветно зачувани во четирите CSV датотетки:

- authors.csv
- posts.csv
- comments.csv
- Inlinks.csv

Овие четири датотеки соодветствуваат на четирите ентитети со кои ќе работиме во продолжение. Тие се меѓусебно поврзани преку ID вредности и временски ознаки. Структурата на податоците овозможува нивно мапирање и во релационен и во граф модел, при што врските меѓу блогерите, објавите и читателите претставуваат клучен елемент за анализата. Овие ентитети ги претставивме на ЕР дијаграмот на слика 1.



Слика 1. ЕР дијаграм на датасетот „Identifying Influential Bloggers“

Во ЕР дијаграмот ги моделиравме следниве ентитети:

- **Author** - содржи информации за авторите на објавите, претставени со уникатен id и име (Name), и изведените атрибути MEIBI (метрика за евалуација на блогери), MEIBIX (проширена верзија на метриката), просечен број на зборови по објава, и просечен број на зборови по објава без стоп-зборови
- **Post** - ги претставува објавите на блогот, преку атрибутите: ID, наслов (Title), содржина (Content), датум на објава (Date), URL, и изведените атрибути: име на автор, должина на објавата, просечна должина на збор, број на inlinks (num_inlinks), број на коментари. Од релацијата со Author, во CSV датотеката се присутни двете колони: *ID* и *Име на блогерот (Blogger Name)*. Овој ентитет служи како централна точка на множеството.
- **Comment** - ги претставува коментарите од публиката, со атрибутите: ID, Име на авторот (се однесува на автор на коментарот, а не на објавата и не е поврзано со Author ентитетот), содржина (Content), датум на објава (Date), и глас (Vote).
- **Inlink** - ги претставува линковите кои водат од други сајтови до објавата. Секој ред содржи ID, Име на авторот (повторно не поврзано со Author ентитетот, туку е име на изворот), наслов (Title), датум на објава (Date), и URL.

Во релационен тип на база на податоци, овие CSV-датотеки можат да се трансформираат во нормализирани табели со јасно дефинирани примарни и надворешни клучеви: author_id во posts.csv се поврзува со authors.csv, а post_id е врска со comments.csv и inlinks.csv. Овој модел овозможува класични SQL-операции како JOIN, агрегирања и филтрирање по време.

Во граф тип, податоците се преставуваат како јазли и релации: блогерите, објавите и читателите се јазли, додека односите како „објавува“, „коментира“, и „реферира“ се насочени врски. На пример, јазел за автор може да има повеќе WROTE врски кон објави, кои пак имаат ON_POST врски од коментари, и POINTS_TO врски од inlink јазли.

Со претставување на овие податоци во двата типови на бази на податоци, можеме да ја разгледаме нивната поврзаност и да ги оцениме перформансите на базите на податоци.

Инсталација

Neo4j

Инсталацијата на менаџмент системот за управување со граф датабази Neo4j претставува многу едноставен процес: од нивната официјална веб страна [2] потребно е да се инсталира десктоп верзијата, и со едноставно поминување преку волшебникот се извршува инсталацијата.

За креирање на нова датабаза, најпрво е потребно да се креира нов проект, и потоа во новокреираниот проект да се креира нова инстанца на датабаза. Во нашиот случај, ние работевме со локална датабаза.

Целиот процес, од инсталацијата до креирањето на празна инстанца за датабазата, не трае повеќе од 10 минути, што претставува многу едноставен и брз процес за корисниците веднаш да може да започнат со работа.

PostgreSQL

За работа со PostgreSQL база на податоци, потребно е да се инсталира локална инстанца на PostgreSQL серверот, која е достапна за преземање од официјалната веб-страница. Пристапот до базата може да се оствари преку командна линија (терминал) или преку графички алатки за управување со бази на податоци, како што се **pgAdmin** или **DataGrip**.

Во рамки на овој проект користевме локално хостирана инстанца од PostgreSQL, што ни овозможи целосна контрола врз податочниот модел и самите податоци. На локалниот сервер креиравме нова база на податоци и ги дефиниравме потребните табели, а поради релативно малиот број на табели, ја користевме стандардната public шема.

Процесот на инсталација и иницијално конфигурирање е брз и интуитивен, што го прави PostgreSQL сигурен, стабилен и пристапен избор за релациско моделирање и управување со структурирани податоци.

Импортирање на податоци

Податочното множество, прикачено на Kaggle [3], достапно е да се преземе во CSV формат. Откако го презедовме множеството, потребно беше истото да го импортираме во соодветно креираните датабази.

Neo4j

Самиот Neo4j нуди интеграција со csv фајлови, така што најпрво е потребно фајловите да се стават во Project Files. Откако ќе се стават фајловите во соодветниот директориум, со доста едноставни и брзо-извршувачки прашалници се креираат соодветните јазли во графот.

1. За креирање на авторите го искористивме следниот прашалник:

```
LOAD CSV FROM 'file:///authors.csv' AS row
CREATE (:Author {
   id: toInteger(row[0]),
   name: row[1],
   meibi_score: toFloat(row[2]),
   meibix_score: toFloat(row[3]),
   avg_words: toFloat(row[4]),
   avg_words_no_stopwords: toFloat(row[5])
});
```

- Прашалникот се изврши за време од 78ms.

2. За креирање на постовите го искористивме следниот прашалник:

```
LOAD CSV FROM 'file:///posts.csv' AS row
MATCH (a:Author {id: toInteger(row[3])})
CREATE (p:Post {
   id: toInteger(row[0]),
   title: row[1],
   content: row[5],
   url: row[6],
   date: row[7],
   num_comments: toInteger(row[4]),
   num_retrieved_inlinks: toInteger(row[8]),
   num_retrieved_comments: toInteger(row[9]),
   len_words: toInteger(row[10]),
   len_words_no_stopwords: toInteger(row[11]),
   avg_word_len: toFloat(row[12]),
   avg_word_len_no_stopwords: toFloat(row[13]),
   meibi_score: toFloat(row[14]),
   meibix_score: toFloat(row[15])
})
CREATE (a)-[:WROTE]->(p);
```

- Прашалникот се изврши за време од 1146ms.

3. Потоа, со цел побрзо извршување на слендите прашалници, креиравме индекси на јазлите за Автор и Пост:

```
CREATE INDEX author_id_index IF NOT EXISTS FOR (a:Author) ON (a.id);
```

```
CREATE INDEX post_id_index IF NOT EXISTS FOR (p:Post) ON (p.id);
```

4. За креирањето на референците го искористивме следниот прашалник:

```
LOAD CSV FROM 'file:///inlinks.csv' AS row
MATCH (p:Post {id: toInteger(row[1])})
CREATE (i:Inlink {
  id: toInteger(row[0]),
  title: row[2],
  author_name: row[3],
  date: row[4],
  url: row[5]
})
CREATE (i)-[:POINTS_TO]->(p);
```

- Прашалникот се изврши за време од 2684ms.

За креирањето на коментарите наидовме на неколку проблеми. Првиот проблем беше тоа што имаше неколку незатворени наводници во comments.csv, или барем Neo4j ги гледаше така. Имено стануваше збор за коментари од типот "... :-\" или "... :\", односно коментари кои завршуваа со коса црта назад. Neo4j ја гледаше последната коса црта пред наводниците како излез од наводниците.

At D:\.Neo4jDesktop\relate-data\dbmss\dbms-825c4f82-aa55-43c4-86e2-2d52d439dcfb\import\comments.csv @ position 1893390 - there's a field starting with a quote and whereas it ends that quote there seems to be characters in that field after that ending quote. That isn't supported. This is what I read: 'Ha, I just tried going to m.google.com/voice on my G1 and got an error saying my device was not supported. Nice :-","AA'

Откако го согледавме тој проблем, креиравме кратка скрипта во Python која го реши проблемот и со тоа ги претпроцесиравме податоците доволно за да бидат читливи за Neo4j.

```
def clean_comments():
  input_path = "../../data/comments.csv"
  output_path = "../../data/comments_cleaned.csv"

  with open(input_path, "r", encoding="utf-8", errors="replace") as infile, \
       open(output_path, "w", encoding="utf-8", newline="") as outfile:
    for line in infile:
      fixed_line = line.replace('\\"', '\\\\"')
      outfile.write(fixed_line)

  print(f"Fixed file saved to: {output_path}")
```

Со оваа функција ги заменивме сите коси црти наназад со двојни коси црти наназад, за да се прави разлика измеѓу напуштена наводница и реална коса црта.

Вториот проблем се појави поради големината на фајлот. Тој содржи 756561 редици и алоцираната максимална меморија за датабазата не беше доволна тој да се импортира во една трансакција.

The allocation of an extra 2,0 MiB would use more than the limit 716,8 MiB. Currently using 715,0 MiB. dbms.memory.transaction.total.max threshold reached

Меморијата за една трансакција беше поставена системски на 70% од целосната heap меморија на базата на податоци, која изнесуваше 1GB. Со промената на поставките на базата на податоци, ја променивме heap меморијата да биди 5GB, и со тоа го решивме проблемот.

Со следниот прашалник ги креиравме и коментарите:

```
LOAD CSV FROM 'file:///comments_cleaned.csv' AS row
MATCH (p:Post {id: toInteger(row[1])})
CREATE (c:Comment {
   id: toInteger(row[0]),
   content: row[2],
   user_name: row[3],
   date: row[4],
   vote: toInteger(row[5])
})
CREATE (c)-[:ON_POST]->(p);
```

- Прашалникот се изврши за време од 6541ms.

PostgreSQL

Најпрвин напишавме DDL изрази за да ги креираме табелите за секој од CSV фајловите.

```
CREATE TABLE authors (
             id INTEGER PRIMARY KEY,
             name VARCHAR(255),
             MEIBI DOUBLE PRECISION,
             MEIBIX DOUBLE PRECISION,
             average_number_of_words_in_posts DOUBLE PRECISION,
             average_number_of_words_in_posts_wo_stopwords DOUBLE PRECISION
);
```

DDL израз за Author ентитетот

```
CREATE TABLE posts (
                      id INTEGER PRIMARY KEY,
                      title TEXT,
                      blogger_name TEXT,
                      blogger_id INTEGER REFERENCES authors(id),
                      num_comments INTEGER,
                      content TEXT,
                      url TEXT,
                      date TIMESTAMP,
                      num_retrieved_inlinks INTEGER,
                      num_retrieved_comments INTEGER,
                      post_length INTEGER,
                      post_length_nostopwords INTEGER,
                      average_word_length DECIMAL,
                      average_word_length_nostopwords DECIMAL,
                      MEIBI_score DECIMAL,
                      MEIBIX_score DECIMAL
);
```

DDL израз за Post ентитетот

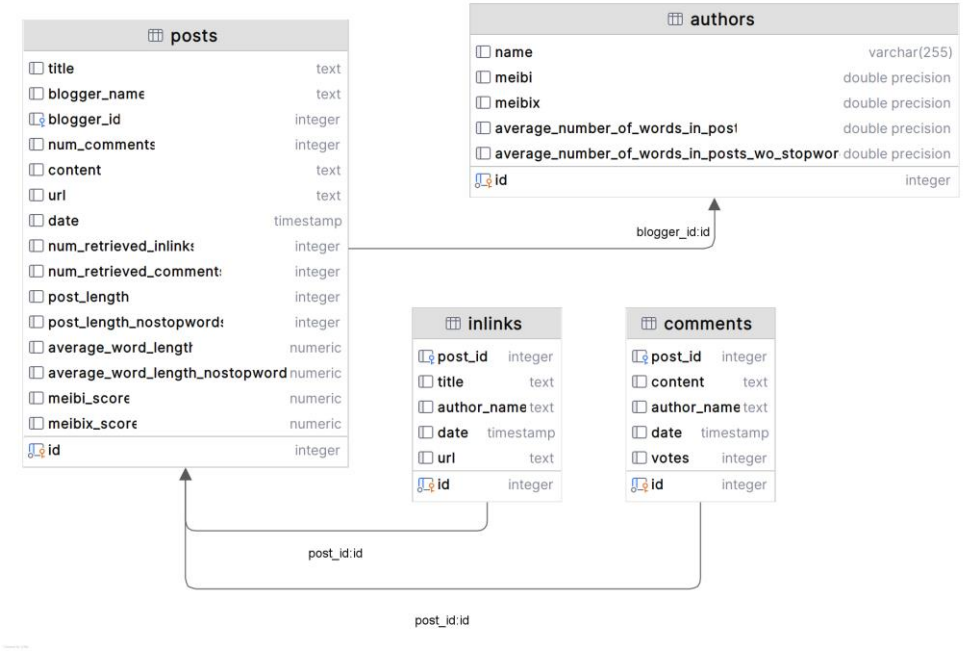
```
CREATE TABLE comments (
                         id INTEGER PRIMARY KEY,
                         post_id INTEGER REFERENCES posts(id),
                         content TEXT,
                         author_name TEXT,
                         date TIMESTAMP,
                         votes INTEGER
);
```

DDL израз за Comment ентитетот

```
CREATE TABLE inlinks (
                        id INTEGER PRIMARY KEY,
                        post_id INTEGER REFERENCES posts(id),
                        title TEXT,
                        author_name TEXT,
                        date TIMESTAMP,
                        url TEXT
);
```

DDL израз за Inlink ентитетот

Со извршување на овие DDL изрази ги креиравме табелите, и го добиваме дијаграмот на базата на податоци претставен на слика 2.



Слика 2. Експортиран дијаграм на реалциона база на податоци

Откако ги креиравме табелите, следниот чекор беше да ги импортираме податоците од CSV датотеките во базата. Но, потребно беше прво да ги парсираме податоците.

CSV датотеките се енкодирани во CP1252 (WIN1252), но бидејќи се одлучивме за PostgreSQL потребно е да бидат во UTF-8. Следно, датумите во CSV датотеките се во формат %Y-%m-%d (пр. 2025-05-01), а во DDL табелите полињата за датум ги поставивме да бидат од тип TIMESTAMP, што значи дека датумот треба да биде во ISO формат. Овие два проблеми ги решивме со скрипта за парсирање на податоци со која ги усогласивме користејќи ги следниве фунцкии.

```
def parse_date(value):
    try:
        return datetime.strptime(value.strip(), '%Y-%m-%d').isoformat()
    except:
        return ''
```

Python функција која ја користиме за парсирање на времето во соодветен формат

```
def clean_text(value):
    if not isinstance(value, str):
        return value
    return value.encode('cp1252', errors='ignore').decode('utf-8',
errors='ignore').strip()
```

Python функција која ја користиме за енкодирање во UTF-8

По креирањето на новите CSV датотеки, со извршување на наредбите дадени подолу ги импортиравме податоците во базата.

```
\COPY authors FROM 'authors_utf8.csv' DELIMITER ',' CSV;
\COPY posts FROM 'posts_utf8.csv' DELIMITER ',' CSV;
\COPY comments FROM 'comments_utf8.csv' DELIMITER ',' CSV;
\COPY inlinks FROM 'inlinks_utf8.csv' DELIMITER ',' CSV;
```

Важно е овие \COPY наредби да се извршат во PSQL терминал, не е можно да се користи COPY наредбата од query console поради недоволните пермисии со кои располага PostgreSQL серверот (без дополнителни конфигурации) за читање на податоци на компјутерот.

Прашалници

Neo4j

1. ***Филтрирања***

1.1 Најдете ги сите објави со “Digital” во насловот:

```
MATCH (p:Post)
WHERE p.title CONTAINS "Digital"
RETURN p.title, p.date
```

- Започна да стримува 85 записи по 12 ms и заврши по 176 ms.

1.2 Најдете коментари од корисник по име “Seika”:

```
MATCH (c:Comment)
WHERE c.user_name = "Seika"
RETURN c
```

- Започна да стримува 6 записи по 12 ms и заврши по 462 ms.

1.3 Добијте ги сите автори со MEIBI > 10:

```
MATCH (a:Author)
WHERE a.meibi_score > 10
RETURN a.name, a.meibi_score
```

- Започна да стримува 21 записи по 17 ms и заврши по 19 ms.

2. ***Спојување на податоци од повеќе инстанци***

2.1 Земете ги објавите со нивните автори и број на коментари:

```
MATCH (a:Author)-[:WROTE]->(p:Post)
OPTIONAL MATCH (p)<-[:ON_POST]-(c:Comment)
RETURN a.name, p.title, count(c) AS num_comments
ORDER BY num_comments DESC
```

- Започна да стримува 19282 записи по 17 ms и заврши по 430 ms.

2.2 Најдете врски што укажуваат на објави од одреден автор:

```
MATCH (a:Author {name: "MG Siegler"})-[:WROTE]->(p:Post)<-[:POINTS_TO]-(i:Inlink)
RETURN p.title, i.title, i.date
```

- Започна да стримува 15227 записи по 15 ms и заврши по 110 ms.

2.3 Добијте објави со повеќе од 5 линкови и најмалку 10 коментари:

```
MATCH (p:Post)
OPTIONAL MATCH (p)<-[:POINTS_TO]-(i:Inlink)
WITH p, count(i) AS inlink_count
WHERE inlink_count > 5
OPTIONAL MATCH (p)<-[:ON_POST]-(c:Comment)
WITH p, inlink_count, count(c) AS comment_count
WHERE comment_count >= 10
RETURN p.title, inlink_count, comment_count
```

- Започна да стримува 9216 записи по 20 ms и заврши по 476 ms.

2.4 Наведете ги авторите и просечниот број на коментари по објава што тие го напишале:

```
MATCH (a:Author)-[:WROTE]->(p:Post)
OPTIONAL MATCH (p)<-[:ON_POST]-(c:Comment)
WITH a, p, count(c) AS comment_count
WITH a, avg(comment_count) AS avg_comments
RETURN a.name, avg_comments
ORDER BY avg_comments DESC
```

- Започна да стримува 107 записи по 14 ms и заврши по 333 ms.

3. *Агрегирани извештаи*

3.1 Добијте ги сите објави во 2009 година со барем еден коментар во кој се споменува “Google”:

```
MATCH (p:Post)<-[:ON_POST]-(c:Comment)
WHERE c.content CONTAINS "Google" AND p.date STARTS WITH "2009"
RETURN DISTINCT p.title, p.date
```

- Започна да стримува 3337 записи по 11 ms и заврши по 502 ms.

3.2 Топ 5 највлијателни автори според просечниот MEIBI и вкупните коментари на нивните објави:

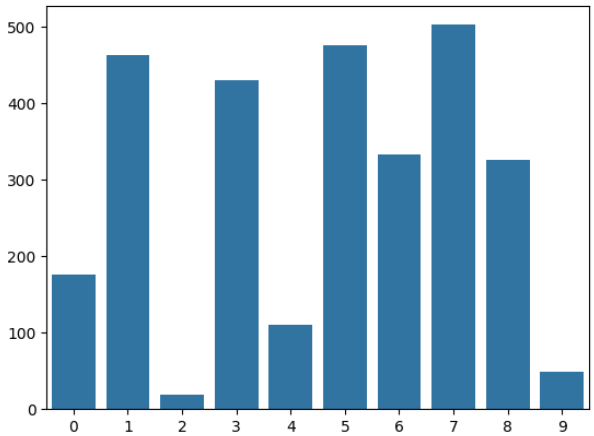
```
MATCH (a:Author)-[:WROTE]->(p:Post)
OPTIONAL MATCH (p)<-[:ON_POST]-(c:Comment)
WITH a, avg(p.meibi_score) AS avg_meibi, count(c) AS total_comments
RETURN a.name, avg_meibi, total_comments
ORDER BY avg_meibi DESC, total_comments DESC
LIMIT 5
```

- Започна да стримува 5 записи по 19 ms и заврши по 325 ms.

3.3 Месечен број на објави и просечна должина на објава (со и без stopwords):

```
MATCH (p:Post)
WITH substring(p.date, 0, 7) AS month,
    avg(p.len_words) AS avg_length,
    avg(p.len_words_no_stopwords) AS avg_length_nostop,
    count(*) AS total_posts
RETURN month, total_posts, round(avg_length, 1) AS avg_length, round(avg_length_nostop,
1) AS avg_length_nostop
ORDER BY month
```

- Започна да стримува 59 записи по 13 ms и заврши по 49 ms.



Слика 3. Времетраење на Neo4j прашалниците изразено во милисекунди

PostgreSQL

1. ***Филтрирања***

1.1 Најдете ги сите објави со “Digital” во насловот:

```
SELECT p.title, p.date
FROM   posts p
WHERE  title LIKE '%Digital%';
```

- Прашалникот се изврши за време од 219 ms. Обработени се 85 редови.

1.2 Најдете коментари од корисник по име “Seika”:

```
SELECT * FROM   comments
WHERE  author_name = 'Seika';
```

- Прашалникот се изврши за време од 385 ms. Обработени се 6 редови.

1.3 Добијте ги сите автори со MEIBI > 10:

```
SELECT NAME, meibi FROM   authors
WHERE  meibi > 10;
```

- Прашалникот се изврши за време од 177 ms. Обработени се 21 редови.

2. ***Спојување на податоци од повеќе инстанци***

2.1 Земете ги објавите со нивните автори и број на коментари:

```
SELECT a.NAME, p.title, Count(c.id) AS num_comments FROM   authors a
JOIN posts p ON a.id = p.blogger_id
LEFT JOIN comments c
ON p.id = c.post_id
GROUP  BY a.NAME,p.title
ORDER  BY num_comments DESC;
```

- Прашалникот се изврши за време од 842 ms. Обработени се 19282 редови.

2.2 Најдете врски што укажуваат на објави од одреден автор(пример ‘MG Siegler’):

```
SELECT p.title AS post_title, i.title AS inlink_title, i.date
FROM   authors a
JOIN posts p ON a.id = p.blogger_id
JOIN inlinks i ON i.post_id = p.id
WHERE  a.NAME = 'MG Siegler';
```

- Прашалникот се изврши за време од 339 ms. Обработени се 15227 редови.

2.3 Добијте објави со повеќе од 5 линкови и најмалку 10 коментари:

```
SELECT p.title,inlink_counts.inlink_count,comment_counts.comment_count
FROM   posts p
JOIN (SELECT post_id, Count(*) AS inlink_count
             FROM   inlinks
             GROUP  BY post_id
             HAVING Count(*) > 5) AS inlink_counts
         ON p.id = inlink_counts.post_id
JOIN (SELECT post_id,Count(*) AS comment_count
             FROM   comments
             GROUP  BY post_id
             HAVING Count(*) >= 10) AS comment_counts
         ON p.id = comment_counts.post_id;
```

- Прашалникот се изврши за време од 508 ms. Обработени се 9216 редови.

2.4 Наведете ги авторите и просечниот број на коментари по објава што тие го напишале:

```
SELECT a.NAME,Avg(comment_stats.comment_count) AS avg_comments
FROM   authors a JOIN posts p ON a.id = p.blogger_id
       JOIN (SELECT post_id, Count(*) AS comment_count
             FROM   comments GROUP  BY post_id) AS comment_stats
             ON p.id = comment_stats.post_id
GROUP  BY a.NAME
ORDER  BY avg_comments DESC;
```

- Прашалникот се изврши за време од 424 ms. Обработени се 102 редови.

3. Агрегирани извештаи

3.1 Добијте ги сите објави во 2009 година со барем еден коментар во кој се споменува “Google”:

```
SELECT DISTINCT p.title,p.date
FROM  posts p JOIN comments c ON p.id = c.post_id
WHERE c.content LIKE '%Google%' AND Extract(year FROM p.date) = 2009;
```

- Прашалникот се изврши за време од 547 ms. Обработени се 3337 редови.

3.2 Топ 5 највлијателни автори според просечниот MEIBI и вкупните коментари на нивните објави:

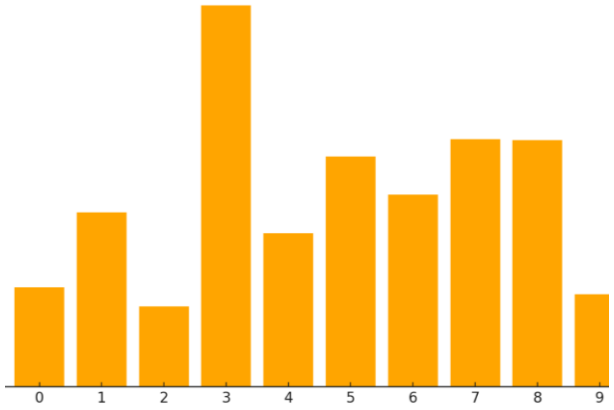
```
SELECT a.name,
       Avg(p.meibi_score) AS avg_meibi,
       Count(c.id)        AS total_comments
FROM   authors a JOIN posts p ON a.id = p.blogger_id
       LEFT JOIN comments c ON p.id = c.post_id
GROUP  BY a.id
ORDER  BY avg_meibi DESC, total_comments DESC
LIMIT  5;
```

- Прашалникот се изврши за време од 545 ms. Обработени се 5 редови.

3.3 Месечен број на објави и просечна должина на објава (со и без запирки (stopwords)):

```
SELECT To_char(date, 'YYYY-MM')                          AS month,
       Count(*)                                          AS total_posts,
       Round(Avg(average_word_length), 1)                AS avg_length,
       Round(Avg(average_word_length_nostopwords), 1) AS
avg_length_nostop
FROM   posts
GROUP  BY To_char(date, 'YYYY-MM')
ORDER  BY month;
```

- Прашалникот се изврши за време од 204 ms. Обработени се 59 редови.



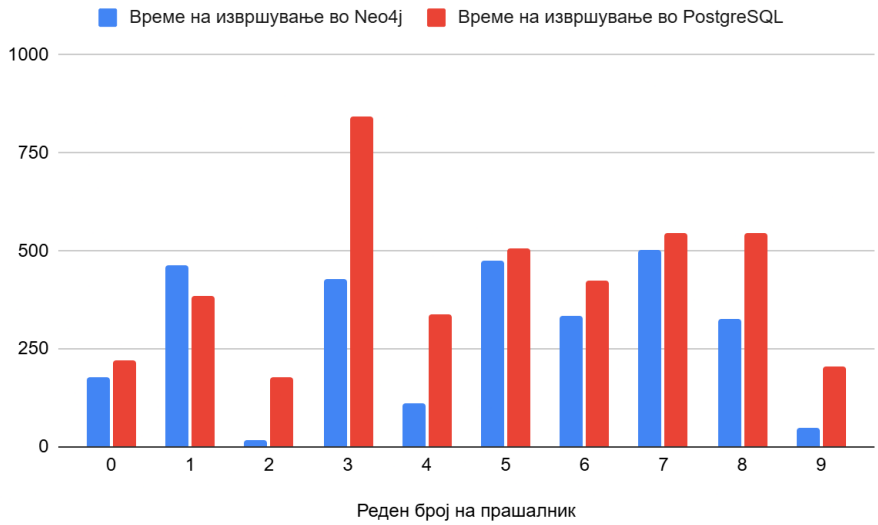
Слика 4. Времетраење на PostgreSQL прашалниците изразено во милисекунди

Добиени резултати

При извршувањето на прашалниците во двете бази на податоци, дојдовме до овие резултати за времето на извршување. Прашалниците се во истиот редослед како што се напишани погоре со ист реден број, почнувајќи од 0. Времето е изразено во милисекунди (ms).

Реден број на прашалник	Време на извршување во Neo4j	Време на извршување во PostgreSQL
0	176	219
1	462	385
2	19	177
3	430	842
4	110	339
5	476	508
6	333	424
7	502	547
8	325	545
9	49	204

Табела 1. Времетраење на прашалниците изразено во милисекунди



Слика 5. Времетраење на прашалниците изразено во милисекунди

Заклучок

Според резултатите добиени во извршените мерења, дојдовме до заклучокот дека за ова податочно множество, подобра опција е граф база на податоци, односно Neo4j во нашиот случај. Сите прашалници, со исклучок само на прашалникот со реден број 1, се извршија за пократок временски интервал.

За разлика од релационите бази на податоци, кои се потпираат на скапи "JOIN" операции низ повеќе табели за да ги следат врските, Neo4j ги складира податоците како јазли и врски со директни физички покажувачи. Тоа значи дека преминувањето од еден до друг поврзан јазол, бара минимални ресурси. Напротив, PostgreSQL мора да направи повеќе "JOIN" операции и пребарувања низ индекси, што станува сè понеефикасно со зголемување на длабочината и сложеноста на релациите.

Користена литература

1. Akritidis, L., Katsaros, D., & Bozanis, P. (2011). Identifying the Productive and Influential Bloggers in a Community. *IEEE Transactions on Systems, Man, and Cybernetics - Part C: Applications and Reviews*, *41*(5), 759–764. https://ieeexplore.ieee.org/abstract/document/5686949/
2. https://neo4j.com/download/
3. https://www.kaggle.com/datasets/lakritidis/identifying-influential-bloggers-techcrunch