Perbandingan Penerapan *Relational Database* dan *Graph Database* dalam Sistem Rekomendasi Film

Jennifer Florentina[#1], Hans Christian Kurniawan[#2]

[#]*Program Studi Informatika, Institut Teknologi Harapan Bangsa*
Jl. Dipati Ukur No. 80, Bandung, Indonesia

[1]jenniferflorentina@gmail.com

[2]hans_christian@ithb.ac.id

Abstract— ***Recommendation systems are used in various applications, such as e-commerce, social media, and others in building recommendation systems that require databases as data storage. The importance of database selection on system performance has increased research on the application of various types of databases in recommendation systems, including this research. This research compares latency and memory usage between relational databases and graph databases in movie recommendation systems. The main indicators in this research are the threshold value for the similarity value limit and the recommendation system technique used. There are 3 techniques used, namely content-based filtering using Jaccard similarity, collaborative filtering using cosine similarity, and hybrid filtering which is a combination of content-based filtering and collaborative filtering. The database used is PostgreSQL for relational databases and Neo4j for graph databases. Based on testing at various threshold values, the latency and memory usage values of the two databases are compared. In the content-based filtering technique, PostgreSQL has a latency time of 120-150 seconds and memory usage of 119-120 MB, while Neo4j is 6-7 seconds and 41-43 MB. In the collaborative filtering technique, PostgreSQL has a latency time of 3-4 seconds and memory usage of 119 - 120 MB, while Neo4j is 4-5 seconds and 24 - 26 MB. In the hybrid filtering technique, PostgreSQL has a latency time of 3-4 seconds and a memory usage of 24 - 26 MB. In the hybrid filtering technique, PostgreSQL has a latency time of 125-150 seconds and memory usage of 119-120 MB, while Neo4j has 9-11 seconds and 32-34 MB.***

Keywords*— *comparison database, recommendation system, graph database, Neo4j, relational database, PostgreSQL, content based filtering, collaborative filtering, hybrid filtering.

Abstrak*— Sistem rekomendasi dipakai di berbagai aplikasi, seperti *e-commerce*, media sosial, dan yang lainnya dalam membangun sistem rekomendasi yang membutuhkan *database* sebagai penyimpanan data. Pentingnya pemilihan *database* terhadap performa sistem membuat penelitian akan penerapan berbagai jenis *database* pada sistem rekomendasi meningkat termasuk dalam penelitian ini. Penelitian ini melakukan perbandingan latensi dan penggunaan memori antara *relational database* dan *graph database* pada sistem rekomendasi film. Indikator utama dalam penelitian ini adalah nilai *threshold* untuk batas nilai *similarity* serta teknik sistem rekomendasi yang digunakan. Terdapat 3 teknik yang digunakan yaitu teknik *content-based filtering* memakai *Jaccard similarity*, *collaborative filtering* memakai *cosine similarity*, dan *hybrid filtering* yang merupakan gabungan dari *content-based filtering* dan *collaborative filtering*. *Database* yang digunakan adalah PostgreSQL untuk *relational database* dan Neo4j untuk *graph database*. Berdasarkan pengujian di berbagai nilai *threshold*, didapatkan nilai latensi dan penggunaan memori pada kedua *database* yang dibandingkan. Pada teknik *content-based filtering*, PostgreSQL memiliki latensi waktu 120-150 detik dan penggunaan memori 119 - 120 MB, sedangkan Neo4j 6-7 detik dan 41-43 MB. Pada teknik *collaborative filtering*, PostgreSQL memiliki latensi waktu 3-4 detik dan penggunaan memori 119-120 MB, sedangkan Neo4j 4-5 detik dan 24-26 MB. Pada teknik *hybrid filtering*, PostgreSQL memiliki latensi waktu 125-150 detik dan penggunaan memori 119-120 MB, sedangkan Neo4j 9-11 detik dan 32-34 MB.

Kata Kunci— **perbandingan *database*, sistem rekomendasi, *graph database*, Neo4j, *relational database*, PostgreSQL, pemfilteran berbasis konten, pemfilteran kolaboratif, pemfilteran hibrida.**

I. PENDAHULUAN

Sistem rekomendasi dapat dipakai di berbagai aplikasi, seperti aplikasi *e-commerce*, media sosial, dll. Dalam pembangunan setiap sistem rekomendasi dibutuhkan sebuah sistem penyimpanan data yang digunakan untuk menyimpan data objek rekomendasi, pengguna, dan data lainnya yang mendukung jalannya sistem rekomendasi [1]. Salah satu *database* yang banyak digunakan adalah *relational database*. Adanya kebutuhan untuk memetakan relasi dengan lebih mudah dipahami dan mengurangi komputasi untuk mendapatkan data dengan banyak relasi, maka munculah konsep *database* lain. Salah satunya adalah *graph database*. *Graph database* menjadi alternatif dari *relational database* karena grafik cukup jelas dan fleksibel yang dapat mengatasi berbagai struktur kompleks [2]. Penggunaan *graph database* pada sistem rekomendasi *memory-based* memungkinkan sistem digunakan secara *real-time*. Hal ini dilakukan untuk mewakili rekomendasi objek terbaru yang lebih cocok sesuai dengan riwayat pencarian pengguna atau merekomendasikan hal yang disukai pengguna lain [2].

Penelitian terdahulu melakukan perbandingan latensi antara *relational database* dan *graph database* pada sistem rekomendasi resep makanan dengan berbagai jumlah data yang ada pada masing-masing database mengunakan metode *content-based filtering*. Jumlah data ada pada 350.000, 700.000, 1.400.000, dan 2.100.000 data. Hasil yang didapatkan *query* dalam *relational database* dengan satu JOIN lebih cepat dibandingkan dengan *graph database*. Pada *query* yang memiliki lebih dari satu JOIN, latensi pada *relational database* lebih tinggi dibanding *graph database* dan meningkat sesuai

dengan pertambahan jumlah data. Semakin banyak *query* JOIN dan jumlah data, maka semakin besar latensi pada *relational database*. Sementara itu, *graph database* cenderung stabil di berbagai jumlah data [3].

Penelitian terdahulu melakukan perbandingan latensi dan penggunaan memori antara *relational database* dan *graph database* pada sistem rekomendasi. Tujuannya adalah untuk pengambilan keputusan kebijakan publik dengan berbagai jumlah data yang ada pada masing-masing database mengunakan metode *content-based filtering*. Hasil penelitian yang didapatkan, latensi dan penggunaan memori pada kedua *database* meningkat diiringi dengan peningkatan jumlah data. *Graph database* memiliki nilai latensi dan penggunaan memori lebih tinggi dibanding *relational database* dengan varian jumlah data 10, 100, 500, 1000, dan 10000 data [4].

Penelitian terdahulu melakukan pengujian optimalisasi sistem rekomendasi dengan *graph database* untuk jalur pengiriman barang mengunakan metode *hybrid filtering* dengan algoritme *pagerank* dan *similarity algorithm*, serta *route path optimization*. Hasil penelitian teknik *hybrid filtering* yang menggabungkan beberapa algoritme optimalisasi memberikan rekomendasi yang lebih akurat daripada algoritme tunggal [5].

Pemilihan *database* pada sebuah sistem rekomendasi mempengaruhi performa dari sistem rekomendasi [3][4][5]. Oleh karena itu, pada penelitian ini dilakukan perbandingan *relational database* dan *graph database* pada sistem rekomendasi film [6] dengan 3 teknik, yaitu: *content-based filtering*, *collaborative filtering*, dan *hybrid filtering*. Perbandingan yang dilakukan adalah berupa perhitungan latensi dan penggunaan memori *relational database* dan *graph database* pada sistem rekomendasi film. Perbandingan tiga teknik, yaitu *content-based filtering*, *collaborative filtering*, dan *hybrid filtering* dilakukan terhadap kualitas hasil rekomendasi dan *database* yang digunakan. Perbandingan didasari dengan kegunaan dan alasan penggunaan *graph database* pada berbagai sistem rekomendasi *memory-based* yang dibandingkan dengan *relational database* serta berbagai macam teknik yang dapat digunakan pada sistem rekomendasi. Penelitian ini diharapkan dapat memberikan perbandingan yang signifikan antara *relational database* dan *graph database* pada sistem rekomendasi dengan tiga teknik, yaitu *content-based filtering*, *collaborative filtering*, dan *hybrid filtering* sehingga dapat membantu pemilihan *database* dan teknik yang tepat untuk mendukung jalannya sistem rekomendasi.

II. METODOLOGI

A. Relational Database

Relational database adalah basisdata yang struktur logisnya terdiri dari kumpulan relasi antar objek yang dipetakan pada tabel berisi kolom dan baris. SQL dikenal dunia sebagai bahasa manipulasi data standar untuk *relational database*. Sebagai bahasa manipulasi data yang lengkap, SQL berisi pernyataan yang memungkinkan pengguna untuk melakukan *insert*, *modify*, *delete*, dan *retrieve* data dari *relational database* [7].

B. Graph Database

Graph database adalah basis data yang struktur logisnya terdiri dari kumpulan objek dan relasi yang dipetakan, seperti model data grafik, yaitu *node*s dan *edges*. *Cypher* adalah query language untuk *graph database* sama seperti SQL untuk *relational database* [2].

C. Sistem Rekomendasi

Sistem rekomendasi adalah sistem penyaringan informasi yang membantu menangani masalah kelebihan informasi dengan menyaring dan memisahkan informasi dan membuat grup dari sejumlah besar informasi yang dihasilkan secara dinamis sesuai dengan preferensi pengguna, minat, atau perilaku yang diamati tentang *item* atau *item* tertentu. Sistem rekomendasi memiliki kemampuan untuk memprediksi apakah pengguna tertentu akan memilih *item* atau tidak berdasarkan profil pengguna dan informasi historisnya [1].

Berdasarkan teknik *filtering* yang digunakan, sistem rekomendasi biasanya dibagi menjadi [8]:

1. *Content-based filtering*
2. *Collaborative filtering*
3. *Hybrid systems*

1) Content-based filtering

Content-based filtering adalah teknik yang didasarkan pada fitur *item* atau objek rekomendasi dan profil pengguna. Teknik ini mencoba menebak respon pengguna terhadap suatu objek berdasarkan respon positif pengguna pada objek yang serupa [8].

2) Collaborative filtering

Collaborative filtering adalah teknik yang didasarkan relasi *item* atau objek rekomendasi dengan pengguna, fitur pada objek tidak diperhitungkan pada teknik ini. Teknik ini mencoba menebak respon pengguna terhadap suatu objek berdasarkan respon positif pengguna lain pada objek tersebut. *Collaborative filtering* mengumpulkan respon pengguna pada *item* yang berbeda dan menggunakannya untuk rekomendasi [8].

3) Hybrid Filtering

Sistem hybrid adalah sistem yang menggabungkan teknik *content-based filtering* dan *collaborative filtering*. Salah satu metode *hybrid filtering* yang dapat digunakan adalah metode *mixed*. Metode *mixed* membuat sistem rekomendasi dengan teknik *content-based filtering* dan *collaborative filtering* kemudian menggabungkan hasil rekomendasi keduanya. Metode *mixed* tidak melakukan banyak pencampuran tujuannya, hanya mengembalikan penyatuan dari hasil semua teknik. Seringkali hasil rekomendasi yang paling dipersonalisasi hanya menghasilkan satu atau dua rekomendasi. Hasil rekomendasi berikutnya menghasilkan beberapa rekomendasi lagi. Dengan cara ini, sistem akan selalu memiliki jumlah rekomendasi yang baik tetapi dengan kualitas sebanyak mungkin [9].

D. Similarity Functions

Dalam sistem rekomendasi, *similarity* atau kemiripan merupakan istilah yang harus diketahui. Pada dasarnya sistem rekomendasi mencari *item* atau objek yang mirip dengan yang

telah pengguna sukai, atau mencari pengguna lain yang mirip dengan target pengguna. Untuk mencari kemiripan antar objek dapat menggunakan *similarity function*. Berikut akan dibahas beberapa *similarity function*s yang biasa dan dapat digunakan dalam sistem rekomendasi [8].

1) Jaccard Similarity

Jaccard similarity digunakan untuk mencari kesamaan antar himpunan. *Jaccard similarity* mengukur banyaknya elemen dalam dua himpunan atau objek yang beririsan (*cardinality of the intersection*). Pada sistem rekomendasi *Jaccard similarity* mengabaikan berapa nilai yang diberikan pengguna pada *item* sehingga cocok untuk diterapkan pada *content-based filtering*. Objek atau himpunan yang dibandingkan pada *Jaccard similarity* merupakan data kategorikal [8].

2) Cosine Similarity

Cosine similarity menganggap kedua objek yang akan dicari nilai kemiripannya sebagai vektor dan sudut antara dua vektor inilah yang dipakai untuk menghitung *similarity*. Pada sistem rekomendasi, *cosine similarity* memerlukan nilai yang diberikan pengguna pada *item* sehingga cocok untuk diterapkan pada *collaborative filtering*. *Cosine similarity* lebih cocok untuk *dataset sparse* yang menghasilkan nilai yang akurat dan bervariasi dari -1 sampai +1. Dari *range* nilai -1 sampai 1, dua vektor dengan orientasi yang sama dan kemiripan yang tinggi akan menghasilkan 1, dua vektor pada 90° memiliki nilai 0, dan dua vektor yang berlawanan secara diametris dan kemiripan yang tinggi akan menghasilkan -1. Objek atau himpunan yang dibandingkan pada *cosine similarity* merupakan data numerik [8].

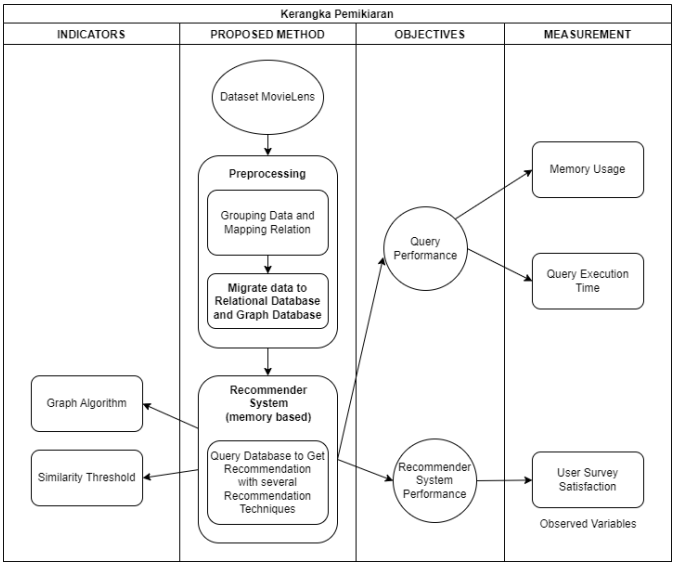
E. Perancangan Sistem

1) Kerangka Pemikiran

Terdapat kerangka pemikiran dari metode yang diusulkan untuk melakukan perbandingan sistem rekomendasi menggunakan *relational database* dan *graph database*. Hal ini ditunjukkan pada Gambar 1.

Pada penelitian ini indikator yang dapat mempengaruhi hasil pengujian adalah nilai *threshold* yang digunakan untuk nilai *similarity* serta *graph algorithm*, dalam hal ini *similarity function* yang digunakan manual atau menggunakan *library* yang tersedia pada *database* Neo4j.

Proses pertama adalah tahap *preprocessing* yang di dalamnya terdapat pengolahan *dataset* untuk memasukkan data ke dalam PostgreSQL sebagai *relational database* dan Neo4j sebagai *graph database*. Setelah memasukkan *dataset* ke dalam *relational database* dan *graph database*, proses selanjutnya adalah membangun sistem rekomendasi dengan ketiga teknik, yaitu teknik *content-based filtering* dengan *Jaccard similarity*, *collaborative filtering* dengan *cosine similarity*, dan *hybrid filtering* untuk *relational database* dan *graph database* yang akan menghasilkan rekomendasi film.



Gambar 1 Kerangka pemikiran

Pengukuran yang digunakan adalah *query performance* dari *database* yang digunakan dan hasil performa dari sistem rekomendasi. Satuan ukur untuk *query performance* adalah perhitungan latensi *query* dan penggunaan memori pada *database*, sedangkan untuk performa sistem rekomendasi adalah perhitungan akurasi yang didasarkan pada survei penilaian hasil rekomendasi yang diberikan oleh pengguna sistem ketika mendapat hasil rekomendasi.

2) Flowchart Global

Terdapat urutan proses global dari metode yang diusulkan untuk melakukan perbandingan sistem rekomendasi menggunakan *relational database* dan *graph database* yang diperlihatkan pada Gambar 2.

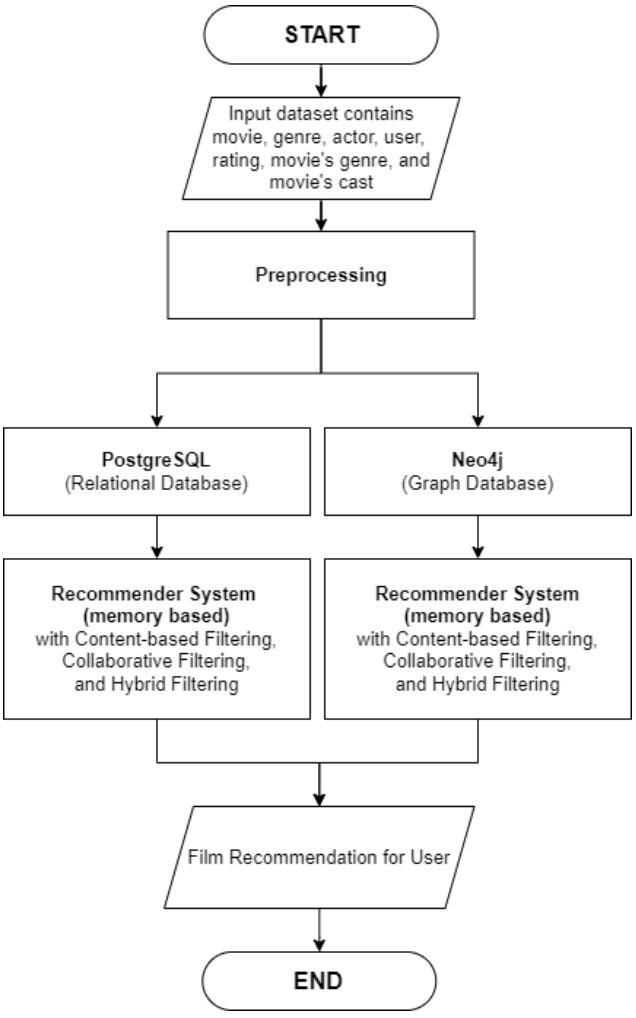
1. *Dataset* yang digunakan berisi data film, *genre*, aktor, relasi film dengan *genre*, relasi film dengan aktor, pengguna, dan nilai pengguna pada film tertentu atau yang biasa disebut *rating*.
2. Data pada *dataset* akan melalui tahap *preprocessing*. Dalam tahap ini data diubah untuk dimasukkan ke dalam PostgreSQL dan Neo4j. Untuk PostgreSQL, data akan disesuaikan dengan model tabel yang berisi kolom dan baris, sedangkan relasi akan dipetakan sesuai dengan konsep *relational database*. Untuk Neo4j, data akan disesuaikan dengan model grafik, yaitu *node*s dan *edges*.
3. Setelah data berhasil diintegrasi dengan PostgreSQL dan Neo4j, maka akan dibangun sistem rekomendasi *memory based* yang berarti akan ada *query* untuk sistem rekomendasi dengan 3 teknik pendekatan.
4. Sistem rekomendasi yang telah dibangun akan menghasilkan rekomendasi film untuk pengguna. Akan ada 6 hasil rekomendasi, 3 berasal dari *relational database* dan 3 berasal dari *graph database*.

Proses pengujian yang dilakukan akan berupa pencatatan latensi serta penggunaan memori dari *graph database* dan *relational database* ketika memroses rekomendasi. Survei dilakukan untuk mengetahui rekomendasi yang dihasilkan akurat dan sesuai dengan pengguna.

3) Dataset

Penelitian ini menggunakan satu *dataset* terdiri dari beberapa *file* .csv yang berisi data film, pengguna, dan nilai pengguna pada film yang diambil dari *website* Kaggle. Terdapat 3 *file* .csv yang dipakai untuk penelitian ini, yaitu:

1. movies_metadata.csv berisi data film, seperti *id*, *title*, dan *genre*. Data film pada *file* ini berjumlah 45.433. Data *genre* pada *file* ini juga akan diolah dengan jumlah data berbeda, yaitu sebanyak 20 *genre*.
2. credits.csv berisi data pemain atau aktor pada film. Dari *file* ini akan dibangun entitas aktor yang memiliki hubungan dengan film. Dari *file* ini didapat data aktor dengan jumlah data berbeda sebanyak 195.754 aktor.
3. *rating*s.csv berisi data nilai dari 270.896 pengguna terhadap film yang ada. Jumlah data *rating*s mencapai 11.436.568 data.



Gambar 2 *Flowchart* global

Dari keseluruhan *dataset*, maka akan ada setidaknya 4 entitas atau objek pada penelitian ini, yaitu:

1. *User* atau pengguna yang memiliki atribut *id*.
2. Film berisi data film sebagai objek rekomendasi yang memiliki atribut *id* dan *title*.
3. *Genre* berisi data *genre* atau jenis sebuah film yang memiliki atribut *id* dan *name*.
4. *Actor* berisi data aktor yang bermain pada film. Aktor memiliki atribut *id* dan *name*.

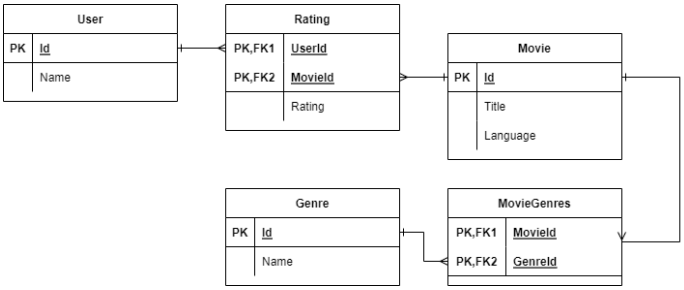
4) Perancangan Database

1. *Relational database* menggunakan PostgreSQL

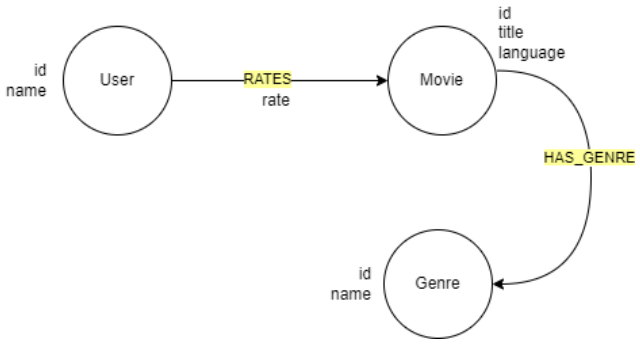
Pada penelitian ini *relational database* yang digunakan adalah PostgreSQL [9][10]. Gambar 3 adalah gambar *entity relationship diagram* yang akan merepresentasikan skema pada *database*. Dapat dilihat ada 7 tabel yang akan dibangun dalam PostgreSQL. *Rating*, MovieGenres, dan Cast adalah tabel hasil dari relasi *many-to-many*. *Rating* adalah tabel hasil dari relasi *many-to-many* antara *User* dan *Movie* yang berisi atribut *rating* yang merupakan nilai dari pengguna pada film tertentu. *MovieGenres* adalah tabel hasil dari relasi *many-to-many* antara *Movie* dan *Genre*. Banyak film bisa memiliki banyak *genre*. *Cast* adalah tabel hasil dari relasi *many-to-many* antara *Movie* dan *Actor* di mana banyak film bisa memiliki banyak aktor yang berperan.

2. *Graph database* menggunakan Neo4j

Tahap ini menjelaskan perubahan *dataset* dari *file* .csv menjadi data grafik dalam Neo4j. Gambar 4 adalah gambaran mengenai grafik yang akan tercipta pada *database* Neo4j. Satu



Gambar 3 *Entity relationship diagram* untuk pemetaan data pada *relational database*



Gambar 4 *Entity relationship diagram* untuk pemetaan data pada *relational database*

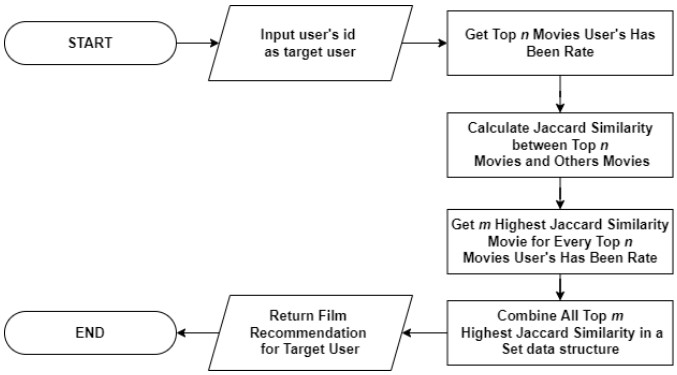
buah *node* pada *graph database* melambangkan satu baris data pada tabel di *relational database* [11][12]. *Node* dengan label *User* akan berisi properti yang sama seperti tabel *User* pada Gambar 5, sedangkan relasi antar entitas dan objek yang pada *relational database* dipetakan melalui atribut *foreign key*. Pada Gambar 6 dilambangkan panah. Dapat dilihat bahwa relasi *Rates* memiliki properti *rating* seperti tabel *Rating* pada Gambar 5.

5) Perancangan Query

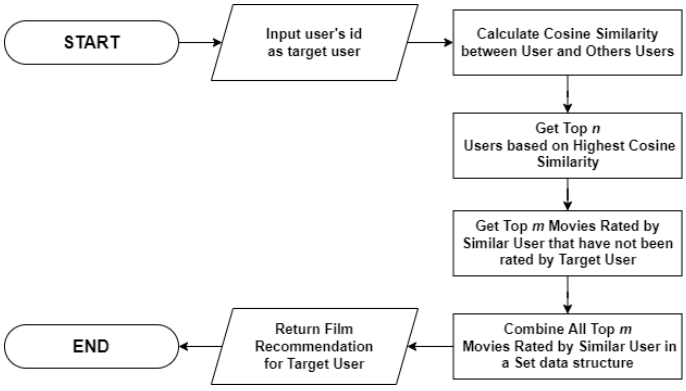
1. Teknik *content-based filtering*

Sistem rekomendasi dengan teknik *content-based filtering* akan memberikan rekomendasi berdasarkan kemiripan antar objek rekomendasi. Dalam penelitian ini, sistem rekomendasi akan memberikan rekomendasi film yang memiliki kemiripan dengan film yang telah dinilai sebelumnya oleh target pengguna [13].

Berdasarkan Gambar 5, sistem rekomendasi akan menerima *id* pengguna yang meminta rekomendasi. Sistem akan mengambil sejumlah *n* film dengan nilai tertinggi yang pernah diberikan oleh target pengguna. Setiap film yang telah diambil akan dihitung nilai *Jaccard similarity* dengan film lainnya. Dari setiap film yang dihitung nilai *Jaccard similarity*-nya akan diambil sejumlah *m* film teratas dengan nilai *Jaccard similarity* melebihi nilai *threshold* yang sebelumnya telah ditentukan. Sejumlah *m* film dengan nilai *Jaccard similarity* tertinggi dari setiap film akan dijadikan satu ke dalam struktur *dataset* sehingga data film yang dihasilkan unik atau tidak duplikat.



Gambar 5 *Content-based filtering flowchart*



Gambar 6 Collaborative Filtering flowchart

Hasil film struktur *dataset* akan menjadi hasil rekomendasi film untuk target pengguna.

Tabel I adalah perintah *pseudocode* yang digunakan sebagai acuan untuk memenuhi proses pengambilan rekomendasi *content-based filtering*.

2. Teknik *collaborative filtering*

Sistem rekomendasi dengan teknik *collaborative filtering* akan memberikan rekomendasi dengan mencari pengguna lain yang menyukai *item* yang sama dan memberikan *item* lainnya yang disukai pengguna lain, namun belum disukai target pengguna. Dalam penelitian ini sistem rekomendasi akan memberikan film yang belum pernah dinilai target pengguna, namun yang pernah dinilai pengguna lain yang memiliki kesamaan nilai dengan target pengguna pada film tertentu [14][15].

Berdasarkan Gambar 6, sistem rekomendasi akan menerima *id* pengguna yang meminta rekomendasi. Sistem akan menghitung nilai *cosine similarity* antara target pengguna dengan pengguna lainnya. Sistem akan mengambil *n* pengguna lain dengan nilai *cosine similarity* yang melebihi nilai *threshold* yang telah ditentukan sebelumnya. Dari setiap pengguna yang telah diambil, akan diambil sejumlah *m* film dengan nilai tertinggi yang diberikan oleh pengguna lain, tetapi belum pernah dinilai target pengguna. Sejumlah *m* film dengan nilai tertinggi dari setiap pengguna lain akan dijadikan satu ke dalam struktur *dataset* sehingga data film yang dihasilkan unik atau tidak duplikat. Hasil film struktur *dataset* akan menjadi hasil rekomendasi film untuk target pengguna.

Tabel II adalah perintah *pseudocode* yang digunakan sebagai acuan untuk memenuhi proses pengambilan rekomendasi *collaborative filtering.*

TABEL I
CONTOH SQL *STATEMENT PSEUDOCODE* YANG DIGUNAKAN SEBAGAI ACUAN UNTUK PENGAMBILAN REKOMENDASI *CONTENT-BASED FILTERING*

No.	*Statement*	Keterangan
1	READ n movies WHERE movie rated by specific user_id SORT DESCENDING BY rating_value	Mengambil sejumlah *n* film dengan nilai tertinggi yang pernah diberikan oleh target pengguna. Sebutan *Query: Get Top Rating Movie by TargetUserId*
2	READ movies, union, intersection CALCULATE Jaccard using union and intersection WHERE one movie based on specific movie_id AND Jaccard > threshold SORT DESCENDING BY Jaccard	Menghitung *Jaccard similarity* antara 1 film dengan film lainnya dan mengambil *m* film dengan nilai *Jaccard similarity* yang melebihi *threshold*. Sebutan *Query: Get Top Movie by Jaccard Similarity*

3. Teknik *hybrid filtering*

Sistem rekomendasi dengan teknik *hybrid filtering* pada penelitian ini akan menerapkan sistem penggabungan antara *content-based filtering* dan *collaborative filtering*. Sistem rekomendasi akan memberikan rekomendasi film yang merupakan gabungan dari hasil rekomendasi film *content-based filtering* dan *collaborative filtering*.

Berdasarkan Gambar 7, sistem rekomendasi akan menerima *id* pengguna yang meminta rekomendasi. Sistem akan menjalankan kedua proses pengambilan rekomendasi film, yaitu *content-based filtering* dan *collaborative filtering*. Rekomendasi film hasil dari *content-based filtering* dan *collaborative filtering* akan dijadikan satu ke dalam struktur *dataset* sehingga data film yang dihasilkan unik atau tidak duplikat. Hasil film struktur *dataset* akan menjadi hasil rekomendasi film untuk target pengguna.

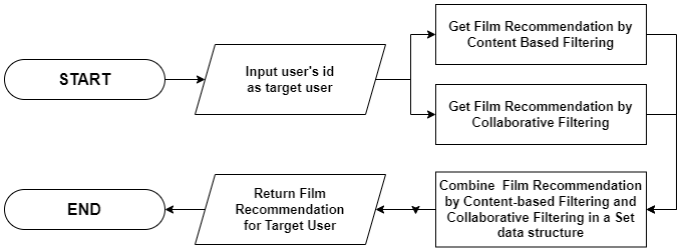
III. Hasil dan Pembahasan

A. *Hasil Pengujian*

1) *Hasil Pengujian Latensi Database*

Tabel II
Contoh SQL Statement Pseudocode Yang Digunakan Sebagai Acuan untuk Pengambilan Rekomendasi *Collaborative Filtering*

No.	*Statement*	Keterangan
1	READ users, rating user's had given CALCULATE cosine using rating value WHERE one user based on specific user_id AND cosine > threshold SORT DESCENDING BY cosine	Menghitung dan mengambil sejumlah *n* pengguna lain dengan nilai *cosine similarity* yang melebihi *threshold*. Sebutan *Query: Get Top User by Cosine Similarity*
2	READ m movies WHERE movie rated by specific user_id AND movie has not been rated by target_user_id SORT DESCENDING BY rating_value	Mengambil sejumlah *m* film dengan nilai tertinggi yang diberikan oleh pengguna lain, tetapi belum pernah dinilai target pengguna. Sebutan *Query: Get Top Rating Movie by UserId*

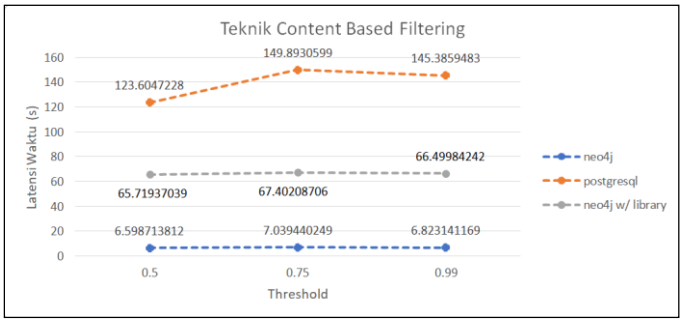


Gambar 7 *Hybrid filtering flowchart*

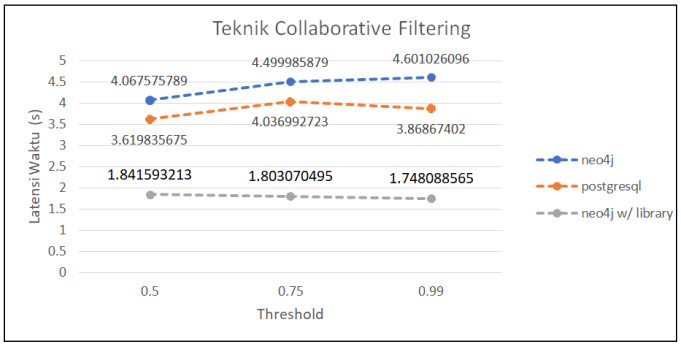
Hasil pengujian latensi *database* PostgreSQL dan Neo4j dikelompokkan berdasarkan teknik rekomendasi dan nilai *threshold* yang digunakan. Nilai latensi yang ditampilkan merupakan nilai rata-rata dari 100 permintaan rekomendasi.

Berdasarkan Gambar 8 didapatkan hasil pengujian antara *database* PostgreSQL dan Neo4j di berbagai *threshold* yang memberikan perbedaan latensi yang cukup signifikan. Dari hasil tersebut disimpulkan bahwa *database* PostgreSQL pada ketiga *threshold* memiliki latensi lebih besar dibandingkan Neo4j. PostgreSQL memiliki rata-rata 123 hingga 150 detik untuk menghasilkan 1 permintaan rekomendasi film dengan teknik *content-based filtering*. Sementara itu, Neo4j memiliki rata-rata 6 hingga 8 detik untuk *query* manual dan 65 hingga 68 detik untuk *query* yang menggunakan *library* dalam menghasilkan 1 permintaan rekomendasi film dengan teknik *content-based filtering*. Dari hasil tersebut juga dapat dikatakan bahwa *threshold* yang digunakan untuk membatasi nilai *Jaccard similarity* yang diambil tidak memberikan dampak besar pada latensi dari kedua *database*.

Berdasarkan Gambar 9 didapatkan hasil pengujian antara *database* PostgreSQL dan Neo4j di berbagai *threshold* yang memberikan perbedaan latensi yang cukup signifikan. Dari hasil tersebut disimpulkan bahwa *database* PostgreSQL pada ketiga *threshold* memiliki latensi lebih kecil dibanding Neo4j. PostgreSQL memiliki rata-rata 3 hingga 4 detik untuk menghasilkan 1 permintaan rekomendasi film dengan teknik *collaborative filtering*. Neo4j memiliki rata-rata 4 hingga 5 detik untuk menghasilkan 1 permintaan rekomendasi film dengan teknik *collaborative filtering*. Dari hasil tersebut juga



Gambar 8 Visualisasi plot latensi PostgreSQL dan Neo4j dengan *threshold* yang digunakan untuk *content-based filtering*

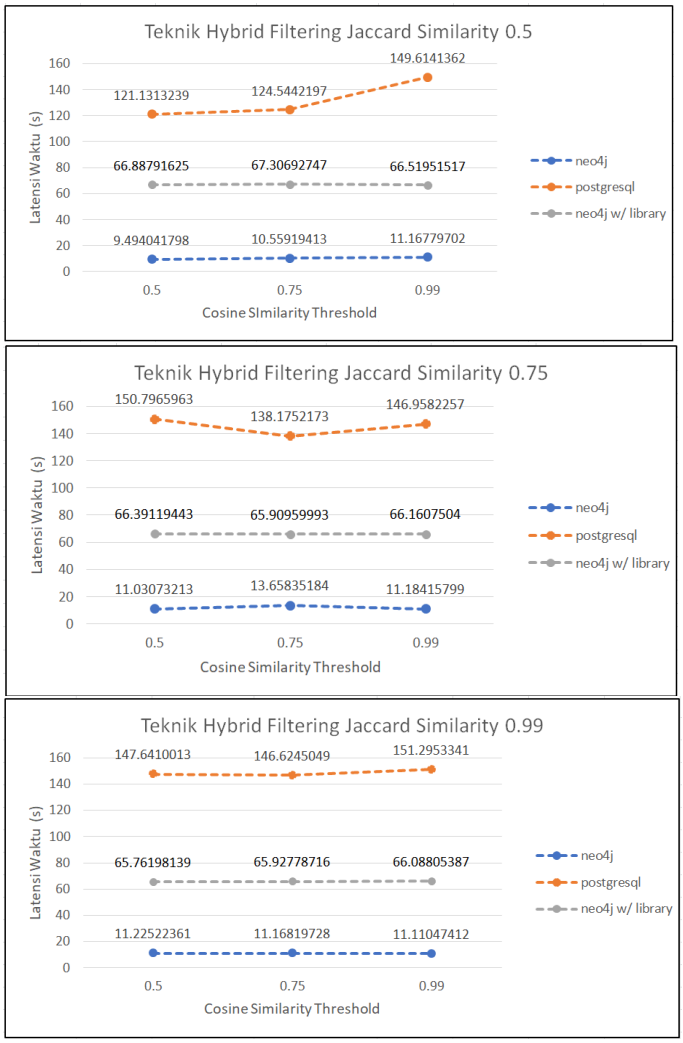


Gambar 9 Visualisasi plot latensi waktu PostgreSQL dan Neo4j dengan *threshold* yang digunakan untuk *collaborative filtering*

dapat dikatakan bahwa *threshold* yang digunakan untuk membatasi nilai *cosine similarity* yang diambil tidak memberikan dampak besar pada latensi waktu dari kedua *database*.

Berdasarkan Gambar 10 didapatkan hasil pengujian antara *database* PostgreSQL dan Neo4j di berbagai *threshold* yang memberikan perbedaan latensi waktu yang cukup signifikan. Dari hasil tersebut disimpulkan bahwa *database* PostgreSQL pada setiap kombinasi *threshold* memiliki latensi waktu lebih besar dibanding Neo4j. PostgreSQL memiliki rata-rata 120 hingga 150 detik untuk menghasilkan 1 permintaan rekomendasi film dengan teknik *hybrid filtering*. Neo4j memiliki rata-rata 9 hingga 11 detik untuk menghasilkan satu permintaan rekomendasi film dengan teknik *hybrid filtering*.

Pada pemaparan hasil pengujian latensi memperlihatkan bahwa pembatasan nilai *similarity* dengan berbagai nilai *threshold* pada ketiga teknik tidak menimbulkan dampak yang signifikan terhadap latensi dari kedua *database*. Perbedaan rata-rata latensi berada pada 1-2 detik antar nilai *threshold*.



Gambar 10 Visualisasi plot latensi waktu PostgreSQL dan Neo4j dengan *threshold* yang digunakan untuk *hybrid filtering*

Dari hasil pengujian yang sudah dilakukan, nilai rata-rata latensi *database* PostgreSQL pada teknik *content-based filtering* dan *hybrid filtering* lebih besar dibanding Neo4j, sedangkan pada teknik *collaborative filtering* nilai rata-rata latensi *database* PostgreSQL lebih kecil dibanding Neo4j. Jika dilihat lebih dalam berdasarkan *query* yang disusun, *Top Movie by Jaccard Similarity* merupakan *query* dengan latensi terbesar. Banyaknya *query* ini dipanggil akan memengaruhi latensi keseluruhan dalam menghasilkan rekomendasi untuk satu permintaan. Walaupun proses *collaborative filtering* PostgreSQL lebih cepat dibanding Neo4j, namun perbedaan untuk proses *collaborative filtering* untuk keduanya hanya 1-2 detik. Sementara itu, proses *content-based filtering* pada PostgreSQL lebih lambat dari Neo4j dengan perbedaan mencapai lebih dari 100 detik. Oleh karena itu, untuk *hybrid filtering* secara keseluruhan Neo4j lebih cepat dibanding PostgreSQL. Pada *query* SQL, latensi meningkat seiring dengan banyaknya perintah *Join*, sub*query*, komputasi matematika pada *query*, dan pemakain perintah SQL. Pada *query Cypher*, latensi meningkat seiring dengan banyaknya relasi yang harus ditelusuri dan jumlah *node* yang sesuai dengan kondisi pada *query*. Pada sistem rekomendasi film yang dibangun, dari hasil pengujian, dapat disimpulkan bahwa PostgreSQL lebih unggul dalam latensi pada teknik *collaborative filtering* dan Neo4j lebih unggul pada teknik *content-based filtering* dan *hybrid filtering*.

Keunggulan PostgreSQL pada teknik *collaborative filtering* dapat ditangani oleh Neo4j dengan memakai *library* dari Neo4j untuk melakukan perhitungan *cosine similarity*. Penggunaan *library* GDS yang ada pada Neo4j dalam perhitungan *cosine similarity* menurunkan latensi yang cukup signifikan hingga mencapai rata-rata 2 detik dari 4 detik. Namun, pada teknik *content-based filtering* berdasarkan penggunaan *library* untuk menghitung *Jaccard similarity* menghasilkan latensi yang lebih besar dibanding *query* manual. Oleh karena itu, dari aspek latensi, pemakaian *library* untuk membantu menghitung nilai *similarity* maksimal, digunakan untuk *cosine similarity* pada teknik *collaborative filtering*, sedangkan untuk *Jaccard similarity* lebih unggul *query* manual.

2) Hasil Pengujian Penggunaan Memori Database

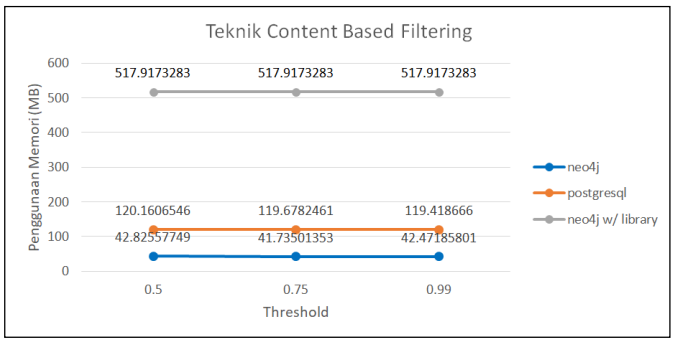
Hasil pengujian penggunaan memori *database* PostgreSQL dan Neo4j dikelompokan berdasarkan teknik rekomendasi serta nilai *threshold* yang digunakan. Nilai latensi yang ditampilkan merupakan nilai rata-rata dari 100 permintaan rekomendasi.

Berdasarkan Gambar 11 didapatkan hasil pengujian antara *database* PostgreSQL dan Neo4j di berbagai *threshold* yang memberikan perbedaan penggunaan memori yang cukup signifikan. Dari hasil tersebut disimpulkan bahwa *database* PostgreSQL pada ketiga *threshold* memiliki penggunaan memori lebih besar dibanding Neo4j. PostgreSQL memiliki rata-rata 119 hingga 121 MB untuk menghasilkan satu permintaan rekomendasi film dengan teknik *content-based filtering*. Neo4j memiliki rata-rata 41 hingga 43 MB untuk menghasilkan satu permintaan rekomendasi film dengan teknik *content-based filtering*. Dari hasil tersebut juga dapat dikatakan bahwa *threshold* yang digunakan untuk membatasi nilai

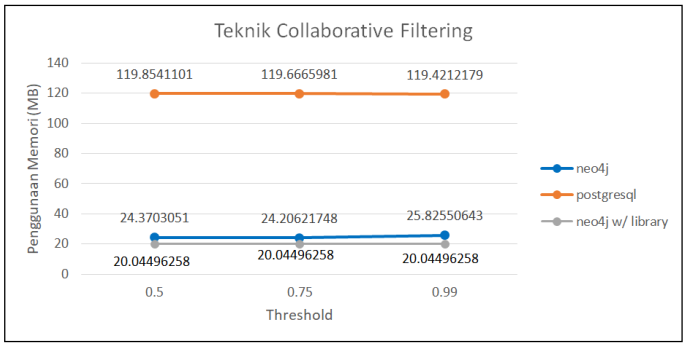
Jaccard similarity yang diambil tidak memberikan dampak besar pada penggunaan memori dari kedua *database*.

Berdasarkan Gambar 12 didapatkan hasil pengujian antara *database* PostgreSQL dan Neo4j di berbagai *threshold* memberikan perbedaan penggunaan memori yang cukup signifikan. Dari hasil tersebut disimpulkan bahwa *database* PostgreSQL pada ketiga *threshold* memiliki penggunaan memori lebih besar dibanding Neo4j. PostgreSQL memiliki rata-rata 119 hingga 120 MB untuk menghasilkan satu permintaan rekomendasi film dengan teknik *collaborative filtering*, sedangkan Neo4j memiliki rata-rata 24 hingga 26 MB untuk menghasilkan satu permintaan rekomendasi film dengan teknik *collaborative filtering*. Dari hasil tersebut juga dapat dikatakan bahwa *threshold* yang digunakan untuk membatasi nilai *cosine similarity* yang diambil tidak memberikan dampak besar pada penggunaan memori dari kedua *database*.

Berdasarkan Gambar 13 didapatkan hasil pengujian antara *database* PostgreSQL dan Neo4j di berbagai *threshold* memberikan perbedaan penggunaan memori yang cukup signifikan. Dari hasil tersebut disimpulkan bahwa *database* PostgreSQL pada ketiga *threshold* memiliki penggunaan memori yang lebih besar dibanding Neo4j. PostgreSQL memiliki rata-rata 119 hingga 120 MB untuk menghasilkan satu permintaan rekomendasi film dengan teknik *hybrid*, sedangkan Neo4j memiliki rata-rata 32 hingga 34 MB untuk menghasilkan satu permintaan rekomendasi film dengan teknik *hybrid filtering*. Dari hasil tersebut juga dapat dikatakan bahwa *threshold* yang digunakan untuk membatasi nilai *cosine similarity* yang diambil tidak memberikan dampak besar pada penggunaan memori dari kedua *database*.



Gambar 11 Visualisasi plot penggunaan memori PostgreSQL dan Neo4j dengan *threshold* yang digunakan untuk *content-based filtering*

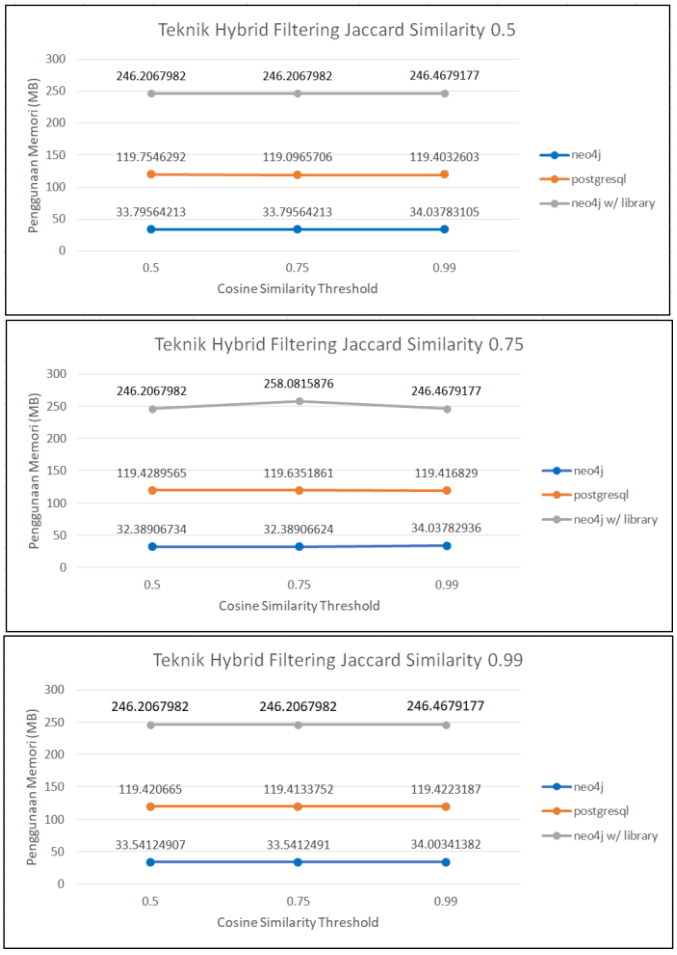


Gambar 12 Visualisasi plot penggunaan memori PostgreSQL dan Neo4j dengan *threshold* yang digunakan untuk *collaborative filtering*

larity yang diambil tidak memberikan dampak besar pada penggunaan memori dari kedua *database*.

Pada pemaparan hasil pengujian penggunaan memori, diperlihatkan bahwa pada *database* PostgreSQL, penggunaan memori untuk ketiga teknik tidak berbeda jauh di antara 119 hingga 120 MB. Teknik tidak mempengaruhi penggunaan memori pada *database* PostgreSQL, sedangkan pada *database* Neo4j teknik *content-based filtering* memiliki penggunaan memori lebih besar dibanding teknik *collaborative filtering*. Pembatasan nilai *similarity* dengan berbagai nilai *threshold* pada ketiga teknik tidak menimbulkan dampak yang signifikan terhadap penggunaan memori dari kedua *database*. Perbedaan rata-rata penggunaan memori berada pada 1-2 MB antar nilai *threshold*.

Pengujian nilai penggunaan memori pada *database* PostgreSQL dan Neo4j merupakan indikator dalam melihat performa kedua *database* dalam kondisi menghasilkan sistem rekomendasi yang sama. Dari hasil pengujian yang sudah dilakukan, nilai rata-rata penggunaan memori *database* PostgreSQL pada ketiga teknik rekomendasi lebih besar dibanding Neo4j. Pada *database* PostgreSQL penggunaan memori bergantung pada seberapa banyak dan besar data tabel yang di-*load* dalam memori. Bila dalam satu tabel hanya diam-



Gambar 13 Visualisasi plot penggunaan memori PostgreSQL dan Neo4j dengan *threshold* yang digunakan untuk *hybrid filtering*

bil 5-6 baris data, tetap terdapat satu tabel yang di-*load* dalam memori. Pada *database* Neo4j, penggunaan memori bergantung pada seberapa banyak *node* dan relasi yang di-*load* dalam memori. Penggunaan memori Neo4j lebih rendah karena *node* dan relasi yang tidak berkepentingan tidak ikut ditelusuri. Jika dilihat pada pemaparan hasil pengujian ini, dalam setiap teknik rata-rata penggunaan memori *database* PostgreSQL ada pada lebih dari 100 MB, sedangkan untuk Neo4j kurang dari 100 MB. Dapat disimpulkan bahwa Neo4j lebih unggul dalam penggunaan memori dibanding dengan PostgreSQL pada setiap teknik rekomendasi.

Optimalisasi dilakukan pada Neo4j dalam menghitung nilai *similarity*. Berdasarkan pemaparan penggunaan *library* GDS yang ada pada Neo4j, perhitungan *cosine similarity* menurunkan penggunaan memori 1 hingga 2 MB. Namun, pada teknik *content-based filtering* penggunaan *library* untuk menghitung *Jaccard similarity* menghasilkan penggunaan memori yang jauh lebih besar dibanding *query* manual. Oleh karena itu, dari aspek penggunaan memori, pemakaian *library*, untuk membantu menghitung nilai *similarity*, maksimal digunakan untuk *cosine similarity* pada teknik *collaborative filtering*, sedangkan untuk *Jaccard similarity* lebih unggul *query* manual.

3) Hasil Pengujian Survei Kepuasan Pengguna

Pada survei kepuasan pengguna terdapat total 20 responden yang menilai hasil rekomendasi dengan 3 teknik, yaitu *content-based filtering*, *collaborative filtering*, dan *hybrid filtering*. Pertama, responden akan memberi nilai terhadap setidaknya 10 film yang ada pada *database*. Nilai yang diberikan akan digabung dengan keseluruhan *dataset*. Sistem akan memberikan rekomendasi dengan 3 teknik berbeda. Hasil rekomendasi yang dinilai untuk setiap teknik paling banyak adalah 10 per teknik. Sepuluh hasil rekomendasi diurutkan berdasarkan *similarity* dan nilai *rating* tertinggi. Responden akan menilai satu-satu apakah film sesuai dengan preferensi pengguna.

Didapatkan rata-rata kepuasan pengguna terhadap hasil rekomendasi untuk teknik *content-based filtering* adalah 86%, untuk teknik *collaborative filtering* adalah 41,75%, dan untuk teknik *hybrid filtering* adalah 69,35%. Pada penelitian ini, teknik *content-based filtering* memberikan persentase kepuasan pengguna tertinggi dibanding teknik lainnya.

Pemilihan teknik rekomendasi dapat dilakukan dengan mempertimbangkan kualitas data yang dimiliki. Jika data pengguna sedikit dan data objek rekomendasi banyak, teknik *content-based filtering* cocok digunakan karena hanya mempertimbangkan nilai target pengguna dan kesamaan antar film. Jika data pengguna banyak dan data objek rekomendasi sedikit, teknik *collaborative filtering* cocok digunakan karena dapat mempertimbangkan dari penilaian pengguna lain pada film. Jika ingin mendapatkan hasil yang bervariasi, dapat menggunakan *hybrid filtering* agar mendapat rekomendasi dari 2 teknik.

IV. SIMPULAN

Berdasarkan hasil penelitian disimpulkan bahwa nilai rata-rata latensi *relational database* PostgreSQL lebih besar dibandingkan *graph database* Neo4j pada teknik *content-based filtering* dan teknik *hybrid filtering*. Pada teknik *collaborative filtering*, nilai rata-rata latensi *relational database* PostgreSQL lebih kecil dibandingkan *graph database* Neo4j. Nilai rata-rata penggunaan memori *relational database* PostgreSQL lebih besar dibandingkan *graph database* Neo4j pada ketiga teknik rekomendasi. Penggunaan *library* GDS yang ada pada Neo4j dalam perhitungan *cosine similarity* menurunkan penggunaan memori sebesar 20% dan menurunkan latensi sebesar 60%. Namun, pada teknik *content-based filtering* penggunaan *library* untuk menghitung *Jaccard similarity* menghasilkan penggunaan memori 10 kali lebih besar dan menghasilkan latensi yang 10 kali lebih besar dibanding *query* manual. Disimpulkan pemakaian library untuk melakukan perhitungan belum tentu mengoptimalisasi latensi waktu dan penggunaan memori *database*.

Pada survey kepuasan pengguna, dari total 20 responden rata-rata kepuasan pengguna terhadap hasil rekomendasi terbesar adalah teknik *content-based filtering* dengan 86%, sedangkan terendah adalah teknik *collaborative filtering* dengan 41,75%. Pada pengujian teknik *content-based filtering* menghasilkan persentase kepuasan pengguna terbesar, namun latensi yang dihasilkan juga terbesar sehingga lebih lambat. Teknik *collaborative filtering* menghasilkan persentase kepuasan terkecil, namun latensi yang dihasilkan terkecil sehingga lebih cepat.

Hasil penelitian dapat berbeda untuk kasus lainnya, selain rekomendasi film. Oleh karena itu, penelitian lanjutan perlu dilakukan dengan objek rekomendasi yang berbeda, seperti toko atau restoran. Pengembangan sistem rekomendasi dengan teknik pendekatan lain juga bias dilakukan, seperti *knowledge-based filtering* dan pendekatan *hybrid filtering* lainnya [16].

Hasil penelitian juga dapat berbeda untuk jenis *database* lainnya. Oleh karena itu, penelitian lanjutan perlu dilakukan dengan jenis *database* lain yang cocok untuk melakukan sistem rekomendasi, seperti *document database* MongoDB, *column-based database* Cassandra, dan jenis *database* lainnya. Bisa juga dilakukan perbandingan *relational database* dan *graph database*, selain PostgreSQL dan Neo4j [17][18].

DAFTAR REFERENSI

[1] F. Ricci, L. Rokach, B. Shapira, dan Kantor Paul B., *Recommender Systems Handbook*. Springer US, 2011. DOI: 10.1007/978-0-387-85820-3.

[2] I. Robinson, J. Webber, dan E. Eifrem, *Graph Databases*, 2nd ed. O'Reilly Media, Inc., 2015.

[3] L. Stanescu, "A comparison between a relational and a graph database in the context of a recommendation system," dalam *Position and Communication Papers of the 16th Conference on Computer Science and Intelligence Systems*, PTI, Sep. 2021, hlm. 133–139. DOI: 10.15439/2021f33.

[4] R. J. Sholichah, M. Imrona, dan A. Alamsyah, "Performance analysis of Neo4j and MySQL databases using public policies decision making data," dalam *7th International Conference on Information Technology, Computer, and Electrical Engineering, ICITACEE 2020 - Proceedings*, Institute of Electrical and Electronics Engineers Inc., Sep. 2020, hlm. 152–157. DOI: 10.1109/ICITACEE50144.2020.9239206.

[5] Z. Nassima dan L. Zineb, "Towards the development of a recommender system for product delivery using graph databases and related

algorithms," *International Journal of Systematic Innovation*, vol. 7, no. 2, hlm. 48–60, 2022, DOI: 10.6977/IJoSI.202206_7(2).0004.
[6] H. Lu, Z. Hong, dan M. Shi, "Analysis of film data based on Neo4j," dalam *Proceedings - 16th IEEE/ACIS International Conference on Computer and Information Science, ICIS 2017*, Institute of Electrical and Electronics Engineers Inc., Jun. 2017, hlm. 675–677. DOI: 10.1109/ICIS.2017.7960078.
[7] J. L. Harrington, *Relational Database Design and Implementation*. Morgan Kaufmann, 2016.
[8] K. Falk, *Practical Recommender Systems*. Manning Publications Co., 2019.
[9] M. Sarwat, R. Moraffah, M. F. Mokbel, dan J. L. Avery, "Database system support for personalized recommendation applications," dalam *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, Apr. 2017, hlm. 1320–1331. [Daring]. Tersedia: https://github.com/DataSystemsLab/recdb-postgresql
[10] R. Obe *et al.*, *PostgreSQL: Up and Running*. O'Reilly Media, Inc., 2012.
[11] R. van Bruggen dan P. Mohanta, *Learning Neo4j: Run Blazingly Fast Queries on Complex Graph Datasets with The Power of the Neo4j Graph Database*. Packt Publishing, 2014.
[12] E. Zhang, J. Fiaidhi, dan S. Mohammed, "Social recommendation using graph database Neo4j: mini blog, Twitter social network graph case study," *International Journal of Future Generation Communication and Networking*, vol. 10, no. 2, hlm. 9–20, Feb. 2017, DOI: 10.14257/ijfgcn.2017.10.2.02.
[13] A. Kumar, "Implementing Real Time Recommendation Systems using Graph Algorithms & Exploring Graph Analytics in a Graph Database Platform (Neo4j)," Dissertation, Dublin Business School, 2018.
[14] C. K. Raghavendra dan K. C. Srikantaiah, "Similarity based collaborative filtering model for movie recommendation systems," dalam *Proceedings - 5th International Conference on Intelligent Computing and Control Systems, ICICCS 2021*, Institute of Electrical and Electronics Engineers Inc., May 2021, hlm. 1143–1147. DOI: 10.1109/ICICCS51141.2021.9432354.
[15] P. E. N. Lutu, "Methods for speeding up recommender system computations using a graph database," dalam *Proceedings of the World Congress on Engineering 2021*, Jul. 2021, hlm. 134–149. [Daring]. Tersedia: http://www.cs.up.ac.za/~plutu;
[16] B. Ramzan *et al.*, "An intelligent data analysis for recommendation systems using machine learning," *Sci Program*, vol. 2019, 2019, DOI: 10.1155/2019/5941096.
[17] S. Mahmud and T. Das Santa, "Oracle, MySQL, PostgreSQL, SQLite, SQL Server Performance based," Daffodil International University Dhaka, 2019.
[18] D. Fernandes dan J. Bernardino, "Graph databases comparison: Allegrograph, arangoDB, infinitegraph, Neo4J, and orientDB," dalam *Data 2018 - Proceedings of the 7th International Conference on Data Science, Technology and Applications*, SciTePress, 2018, hlm. 373–380. DOI: 10.5220/0006910203730380.

Jennifer Florentina, menerima gelar Sarjana Komputer tahun 2023 di Institut Teknologi Harapan Bangsa.
Hans Christian Kurniawan, menerima gelar Sarjana Teknik Informatika dari Institut Teknologi Harapan Bangsa (ITHB) Bandung tahun 2016 dan Magister Informatika dari ITB pada tahun 2019. Saat ini aktif sebagai pengajar di Prodi Informatika ITHB dan juga sebagai Engineering Lead di Bukalapak.