2 LANDASAN TEORI

2.1 Pengertian Graph Database

Graph database merepresentasikan data sebagai kumpulan *node* dan *edge* yang disebut *graph* (Sahatqija et al., 2018). *Edge* merepresentasikan hubungan antar data, yang dalam *relational database* disebut sebagai *relationship*. *Node* merepresentasikan objek atau entitas, dan *property* merepresentasikan atribut. *Graph database* menyediakan solusi yang efektif dan efisien untuk menyimpan data yang saling berhubungan atau memiliki banyak *relation* seperti sistem rekomendasi, pendeteksi penipuan, jaringan sosial pada sosial media, aplikasi geospasial, dan lain-lain. *Relational database* memiliki kelemahan dalam mengatasi relasi antar data yang kompleks dan model tabularnya yang menyulitkan penambahan data serta relasi (Fernandes & Bernardino, 2018). Hal ini yang menjadi keunggulan *graph database*, karena dapat menyimpan relasi antar data dengan efektif dan memiliki fleksibilitas dalam beradaptasi dengan kebutuhan serta penambahan relasi dan data baru. *Graph database* menjadi pilihan yang menjanjikan karena seiring berkembangnya zaman, waktu *query real-time* serta frekuensi perubahan skema dan volume data akan semakin meningkat. Hal ini membuat kelebihan *graph database* semakin terlihat dibandingkan dengan *relational database* yang kurang mampu mengatasi data dengan *relation* yang kompleks serta volume yang besar. Contoh *graph database* antara lain adalah Neo4j, AllegroGraph, ArangoDB, OrientDB, dan InfiniteGraph.

2.2 Pengertian Neo4j

Neo4j adalah *graph database management system* dan memiliki *query language* bernama Cypher. Neo4j merupakan *graph database* yang paling populer digunakan saat ini. Hal ini karena kemampuannya yang mampu menangani miliaran *node* dan *relationship* dalam suatu jaringan (N. S. et al., 2018). Dalam Neo4j, suatu *node* dan *relationship* dapat memiliki *property* pula. Platfom Neo4j untuk didesain untuk memetakan, menganalisis, menyimpan, dan melintasi jaringan data-data yang terhubung, serta mengungkapkan *relationship* atau fakta-fakta yang tersembunyi (Srivastava & Singh, 2018). Kelebihan lain dari Neo4j adalah fleksibilitasnya yang jauh lebih unggul dibandingkan dengan *relational database*.

2.3 Pengertian Similarity Search

Similarity search pada *graph database* merujuk pada proses mencari *node* atau *graph* yang memiliki kemiripan terhadap *node* atau *subgraph* lain yang dihasilkan dari suatu *query*.

Suatu *node* dapat dikatakan mirip dengan *node* lainnya adalah jika memiliki *neighbor* atau *node-node* lain yang berelasi yang mirip pula (Koutra et al., 2011). Dalam Neo4j, hal ini disebut dengan algoritma Node Similarity, yang mana membandingkan sekumpulan *node* berdasarkan *node* yang terhubung dengannya. Dua *node* dianggap serupa jika terdapat mereka memiliki banyak *neighbor* yang sama. Terdapat 3 metrik untuk mengukur *node similarity*, yaitu Jaccard Similarity, Overlap Coefficient, dan Cosine Similarity.

2.3.1 Jaccard Similarity

Jaccard yang awalnya dikembangkan oleh Jaccard (1901), ukuran yang digunakan untuk menganalisis kesamaan (atau ketidaksamaan) antara dua sampel data. Jaccard mendefinisikan *similarity* atau kesamaan antara dua himpunan dengan jumlah elemen yang sama antar dua himpunan tersebut dibagi dengan jumlah elemen gabungan dari dua himpunan (Verma & Aggarwal, 2020). Nilai *similarity* yang dihasilkan adalah 0 hingga 1, dengan nilai 1 adalah kedua himpunan sama dan identik, sedangkan nilai 0 menandakan kedua himpunan tersebut sama sekali berbeda.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

Gambar 2.1 Rumus Jacard Similarity

2.3.2 Overlap Coefficient

Overlap Coefficient atau Szymkiewicz–Simpson Coefficient memiliki cara pengukuran yang hampir sama dengan Jaccard. Perbedaannya adalah jumlah elemen yang sama antar dua himpunan tersebut dibagi dengan jumlah elemen himpunan yang paling sedikit (Verma & Aggarwal, 2020).

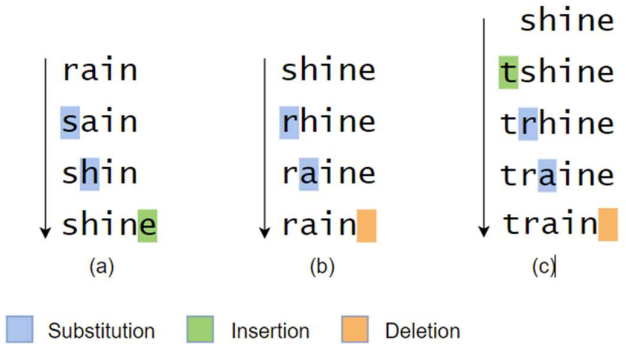
$$O(A,B) = \frac{|A \cap B|}{min(|A|, |B|)}$$

Gambar 2.2 Rumus *Overlap Coefficient*

2.3.3 Levenshtein Similarity

Levenshtein Similarity membandingkan kemiripan antara dua sampel String dengan menghitung jumlah perubahan yang diperlukan untuk mengubah satu String ke lainnya.

Perubahan dapat berupa *substitution* atau mengganti karakter, *deletion* atau menghapus karakter, dan *insertion* atau menyisipkan karakter (Po, 2020). Gambar 2.3 adalah contoh proses Levenshtein Similarity untuk mengubah kata "rain" menjadi "shine" (arvindpdmn, 2019).



Gambar 2.3 Contoh Levenshtein Similarity

2.3.4 Sorensen-Dice Similarity

Sorensen-Dice Similarity adalah koefisien yang digunakan untuk mengukur kemiripan antar dua *set* data. Sorensen-Dice memberi bobot pada elemen yang muncul pada kedua *set* dengan mengalikannya dengan 2, dan kemudian dibagi dengan jumlah total elemen dari kedua *set* (Pati & Rautray, 2021).

$$DSC(A,B) = \frac{2|A \cap B|}{|A|+|B|}$$

Gambar 2.4 Rumus Sorensen-Dice Similarity

2.3.5 Cosine Similarity

Cosine Similarity adalah pengukuran metode *similarity* yang berbasis sudut. Cosine Similarity menghitung kosinus sudut antara dua vektor untuk mengetahui kesamaan antar dua himpunan (Afzali & Kumar, 2017). Jika kedua himpunan serupa, maka vektor-vektornya akan searah dari titik asal, sehingga membentuk sudut yang relatif kecil, yang nilai kosinusnya mendekati 1.

$$cos_w(A,B) = \sum_i \frac{\alpha_i \cdot \beta_i}{\sqrt{\sum_i \alpha_i^2} \cdot \sqrt{\sum_i \beta_i^2}}.$$

Gambar 2.5 Rumus *Cosine Similarity*

2.4 Pengertian Cypher

Cypher adalah bahasa *query* yang digunakan secara komersial oleh berbagai perusahaan multinasional dan peneliti *data mining* (Srivastava & Singh, 2018). Cypher menggunakan metode ASCII untuk merepresentasikan pola yang berbeda. Menurut *website* resmi Neo4j, Cypher memberikan cara visual untuk merepresentasikan *node* dan *relationship*. Cypher menggunakan tanda kurung siku '[]' dan simbol panah '->' untuk merepresentasikan *relationship* dan tanda kurung '()' untuk merepresentasikan *node*.

2.5 Pengertian Rodex Tours & Travel

Rodex Tours & Travel Surabaya adalah perusahaan yang menawarkan layanan *tour* dan *travel*, baik di dalam maupun di luar negeri. Mereka memiliki platform website untuk memesan tiket pesawat dan hotel. Saat memesan hotel, pelanggan harus memasukkan nama hotel, durasi menginap, dan jumlah orang yang akan menginap. Sistem memberikan saran berdasarkan nama hotel, dan setelah dipilih, sistem mencari detail hotel beserta beberapa pilihan harga dari berbagai *vendor*. *Vendor*-*vendor* ini menyediakan informasi pemesanan, seperti tipe kamar dan layanan tambahan, kepada agen perjalanan. Ini menghasilkan beberapa opsi kamar dan layanan dengan harga berbeda karena berasal dari *vendor* yang berbeda. Setelah pelanggan memilih hotel dan kamar yang diinginkan, mereka diminta mengisi nama tamu, kontak pemesanan, dan melakukan pembayaran. Data pemesanan pelanggan dikirim kembali kepada *vendor* yang dipilih untuk diproses oleh hotel terkait.

Vendor yang berbeda tentunya dapat memberikan hotel-hotel yang sama, walaupun dengan pilihan harga yang berbeda. Dan untuk menghindari kesalahan pemrosesan data, perlu dilakukan proses pemetaan data antara *vendor* dan data master Rodex Tours & Travel. Kesalahan pemetaan dapat mengakibatkan hasil pencarian yang tidak akurat, seperti menampilkan lebih dari satu hotel, tidak menampilkan hasil sama sekali, atau menggabungkan data hotel yang tidak seharusnya. Penting untuk memastikan pemetaan yang tepat untuk mencegah kendala dalam proses pemesanan dan potensi kerugian perusahaan.