Performansi *Response Time Query* Pada Hadoop-Hive Menggunakan Metode *Partition*

Marwan[1)], Zawiyah Saharuna[2)] Rini Nur[3)]
[1,2,3] Teknik Elektro, Politeknik Negeri Ujung Pandang,
email:marwan.elektro16@gmail.com[1], zawiyah@poliupg.ac.id[2], rini@poliupg.ac.id[3]

Abstract

Hive menggantikan teknik pemrosesan tradisional RDBMS yang tidak dapat digunakan pada big data. Tetapi, Hive dengan kondisi default akan mencari data secara menyeluruh saat mengeksekusi query. Metode partition mampu mengelompokkan data, sehingga dilakukan pengujian untuk mengetahui apakah dengan mengelompokkan data akan memberikan peningkatan performansi response time query atau sebaliknya. Pada penelitian ini, dibangun infrastruktur Hadoop cluster dengan sistem multi node menggunakan virtual machine. Dataset yang digunakan adalah dataset Movielens dengan kardinalitas atribut yaitu 5, 50 dan 100. Tiap dataset terdiri dari 15 juta records data. Berdasarkan hasil penelitian, metode partition selain mampu mengelompokkan data juga memberikan performansi response time query yang lebih cepat sebesar 30.8% dibandingkan kondisi default. Selain itu, Metode partition saat kardinalitas 100 lebih baik dibandingkan dua kardinalitas yang lebih kecil yaitu kardinalitas 5 dan kardinalitas 50.

Keywords: *Big Data, Hadoop, Hive, Partition.*

I. PENDAHULUAN

Pengelolaan *big data* yang berukuran hingga *petabyte*, membuat teknik pemrosesan tradisional seperti *Relational Database Management System* (RDBMS) tidak dapat lagi digunakan [1]. Saat ini, sistem *platform* yang dikenal untuk mengelola *big data* yaitu Hadoop. Hadoop adalah *platform* yang sangat mendukung untuk melakukan pengelolaan *big data* secara terdistribusi dengan melibatkan berkluster-kluster komputer [2]. Salah satu ekosistem dari Hadoop untuk menggantikan teknik pemrosesan tradisional pada *big data* adalah Hive.

Hive adalah sebuah solusi *platform open-source data warehousing* untuk pengguna mampu mengelola *big data* yang ada pada *Hadoop Distributed File System* (HDFS). *Data warehousing* merupakan sistem yang mengambil (*retrieve*) dan mengkonsolidasikan (*consolidate*) data secara periodik (*periodically*) dari sumber data ke dalam penyimpanan dimensional dan ternormalisasi. Hive sebagai *data warehousing* yang andal saat ini menyederhanakan pengelolaan *dataset* pada Hadoop *cluster* karena mendukung bahasa yang *familiar* dengan *Structured Query Language* (SQL) yang disebut dengan istilah *HiveQL*. Hive menjadi sangat populer di kalangan *non-programmer* karena menghilangkan kebutuhan untuk menulis program *Mapreduce* yang kompleks [3].

Penelitian sebelumnya telah dilakukan oleh Ashwitha (2017) untuk menganalisis *dataset* pada Hadoop *cluster* menggunakan Hive. Akan tetapi, penelitian Ashwitha belum membahas terkait pengujian performansi *response time query* menggunakan metode *partition* [4]. Metode *partition* akan mengelompokkan data berdasarkan kardinalitas atribut. Metode tersebut adalah metode yang efisien untuk memberikan struktur data yang ekstra pada Hive [5]. Hal ini penting diteliti untuk menganalisis apakah dengan struktur data yang telah dikelompokkan akan memberikan peningkatan performansi dalam mengeksekusi *query* dalam *big data* atau tidak. Jika data sedikit, tidak akan menimbulkan permasalahan, akan tetapi jika datanya mencapai jutaan atau puluhan juta *records*, akan menimbulkan permasalahan yaitu lamanya waktu pencarian data. Berdasarkan masalah yang telah dibahas diatas, maka penelitian ini melakukan pengujian performansi *response time query* berdasarkan metode *partition.*

II. KAJIAN LITERATUR

A. Big Data

Big data didefinisikan sebagai fenomena yang ditandai dengan adanya peningkatan volume yang sedang berlangsung, variasi, kecepatan dan kebenaran data yang membutuhkan teknologi yang canggih untuk dilakukan penangkapan, penyimpanan, pendistribusian, pengelolaan dan

penganalisisan suatu data yang besar [6]. *Big data* adalah sebuah istilah yang digunakan untuk menjelaskan data dengan ukuran yang besar disertai dengan volume yang akan terus bertambah.

B. Hadoop

Hadoop diciptakan sebagai *software framework* dengan melibatkan berkluster-kluster komputer, artinya mampu menghubungkan beberapa komputer untuk dapat bekerja sama dalam hal menyimpan dan mengelola *big data*. Ketika penggunaan data melonjak tajam, komputer pada *cluster* tidak kewalahan karena tidak satu komputer saja yang akan mengurus data tetapi ada 2 atau lebih yang akan memproses data pada saat yang sama.

C. Hive

Hive merupakan *apache open source* data *warehousing* untuk Hadoop. Fungsi utama dari Hive adalah diciptakan untuk menyediakan data *summarization, query* dan analisis *dataset* yang besar [7].

D. Partition

Hive mengatur tabel dalam bentuk *partition* untuk mengelompokkan data yang sama menjadi beberapa bagian berdasarkan *uniq values* dari atribut [8]. Dalam banyak hal, skema *partition* ini mirip dengan apa yang disebut sebagai *partition* oleh banyak vendor *database*, tetapi terdapat perbedaan dimana nilai *partition key* disimpan dengan metadata dan bukan dengan data [9].

III. METODE PENELITIAN

Metode dalam penelitian diperlukan agar serangkaian kegiatan menjadi teratur dan sistematis sehingga hasil yang diperoleh sesuai dengan tujuan dari sebuah penelitian. Adapun tahapan penelitian seperti pada Gambar 1 berikut.

Gambar 1. Metode Penelitian

A. Studi Literatur

Tahapan penelitian dimulai dengan studi literatur. Studi literatur dilakukan untuk memahami konsep dan karakteristik *big data* serta *platform* Hadoop dan Hive terutama metode *partition* melalui pustaka-pustaka yang berkaitan dengan penelitian, berupa buku, jurnal dan skripsi ataupun *via web*. Pada tahapan ini dilakukan identifikasi terhadap permasalahan, serta rancangan skenario yang menjadi dasar dilakukannya sebuah penelitian.

B. Analisis Kebutuhan

Tahapan kedua adalah tahapan analisis kebutuhan. Analisis kebutuhan bertujuan untuk mengumpulkan informasi perangkat keras (*hardware*), perangkat lunak (*software*) serta bahan *dataset* yang dibutuhkan dalam penelitian. Pada tahap ini diperoleh perangkat keras (*hardware*) dan perangkat lunak (*software*) yang dibutuhkan adalah sebagai berikut:

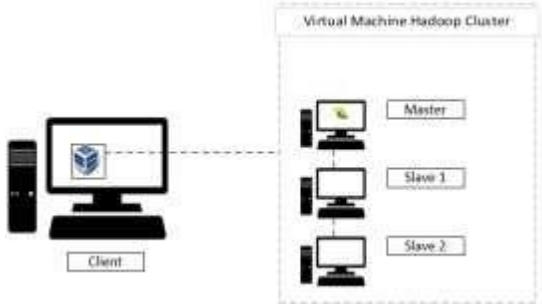
1) 1 unit *Personal Computer* (PC), *Memory* RAM 12 GB, Sistem Operasi Windows 10 Pro, *Processor Core* i5.
2) Oracle VM VirtualBox
3) Sistem Operasi Ubuntu *Server*
4) Putty
5) JavaKit 8
6) Ssh
7) Hadoop 2.8.0
8) Hive 1.2.1
9) WinsCP

Adapun untuk kebutuhan *dataset* yang digunakan dalam penenlitian ini berasal dari situs *website* penyedia *dataset Movielens*. *Dataset Movielens* adalah data yang berisikan informasi *user* yang telah memberikan *rating* pada sebuah judul film beserta data identitas diri dari *user*. *Dataset* ini terdiri dari tiga file *dataset* yaitu : file *dataset* ratings, file *dataset* users dan file *dataset* movies. *Dataset* akan dilakukan penggabungan dari tiga file *dataset* menjadi satu file data menggunakan *Hive Query Language* (*HiveQL*). *Dataset* yang digunakan untuk pengujian adalah data dengan kardinalitas atribut userid yaitu 5, 50 dan 100. Tiap *dataset* terdiri dari 15.000.000 *records* data.

C. Perancangan Sistem

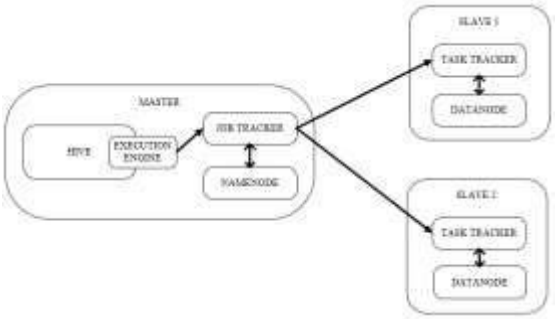
Tahapan perancangan sistem adalah tahapan bagaimana melakukan sebuah penelitian secara teknis untuk mencapai tujuan penelitian. Dari tahapan hasil studi literatur dan analisis kebutuhan, diambil simpulan dari beberapa sumber untuk menentukan model

sistem untuk penelitian. Model sistem yang dirancang terlihat pada Gambar 2 berikut:



Gambar 2. Infrastruktur Hadoop *Cluster*

Adapun alur proses *request* data saat melakukan eksekusi *query* dari Hive adalah sebagai berikut:



Gambar 3. Alur Proses *Request* Data saat Eksekusi *Query*

Perancangan model *query* yang akan dijadikan sebagai bahan uji dalam penelitian ini adalah didasarkan pada atribut userid yang dijadikan sebagai bahan untuk menerapkan metode *partition*, sehingga eksekusi *query* yang dilakukan berdasarkan *uniq values* dari atribut tersebut.

D. Implementasi

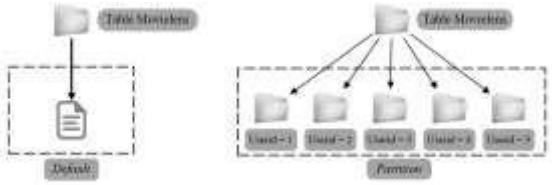
Tahapan ini adalah sebagai penerapan model sistem yang telah dirancang dengan tujuan agar sistem dapat beroperasi dengan baik dalam menjalankan skenario pengujian. Tahapan ini dimulai dengan penginstalan Oracle Vm VirtualBox hingga instalasi dan konfigurasi pada Hadoop dan Hive.

E. Pengujian

Tahap pengujian adalah tahapan untuk mengetahui perfomansi *response time query* pada metode *partition*. Secara umum, tahapan pengujian adalah sebagai berikut:

a) Skenario Pertama

Skenario pertama bertujuan untuk menguji performansi *response time query* pada metode *partition* dengan membandingkannya dengan kondisi *default*. Berikut kondisi pengujian pada skenario pertama menggunakan data kardinalitas 5:

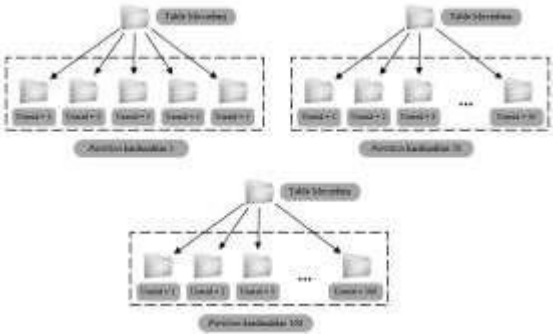


Gambar 4. Kondisi Pengujian pada Skenario Pertama

Pengujian dilakukan sebanyak 5 kali. Data hasil pengujian selanjutnya diinputkan kedalam tabel dan disajikan dalam bentuk grafik.

b) Skenario Kedua

Skenario kedua bertujuan untuk menguji performansi *response tme query* berdasarkan variasi kardinalitas pada metode *partition*. Hal ini dilakukan untuk melihat apakah variasi kardinalitas berpengaruh atau tidak terhadap performansi *response time query* pada metode *partition*. Pengujian dimulai pada kardinalitas 5, kemudian dilakukan pengujian pada kardinalitas 50, dan terakhir pada data kardinalitas 100. Berikut kondisi pengujian pada skenario kedua:



Gambar 5. Kondisi Pengujian pada Skenario Kedua

Pengujian dilakukan sebanyak 5 kali. Data hasil pengujian selanjutnya diinputkan kedalam tabel dan disajikan dalam bentuk grafik.

F. Analisis Performansi

Pada tahap ini dilakukan analisis performansi dari hasil pengujian yang telah didapatkan selama penelitian dan selanjutnya akan menghasilkan sebuah kesimpulan dari penelitian ini.

IV. HASIL DAN PEMBAHASAN

A. Import *Dataset* ke Hadoop Cluster

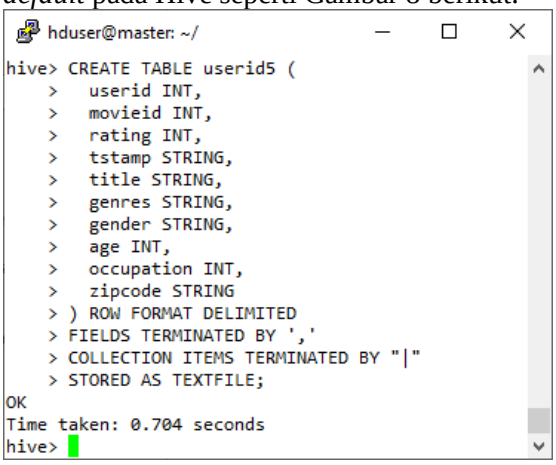
Gambar 6. *Dataset* Akhir

Gambar 7. Tampilan pada HDFS

B. Implementasi Data pada Hive

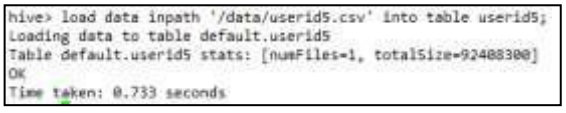
1) Implementasi Data pada Kondisi *Default*

Implementasi data pada kondisi *default* diawali dengan membuat metadata kondisi *default* pada Hive seperti Gambar 8 berikut:



Gambar 8. Pembuatan Metadata Kondisi *Default*

Kemudian dilakukan *load data* dari file *dataset* yang telah ada di direktori /data ke direktori kondisi *default* menggunakan *statement* atau *query* seperti pada Gambar 9 dibawah:



Gambar 9. *Load Data* pada Kondisi *Default*

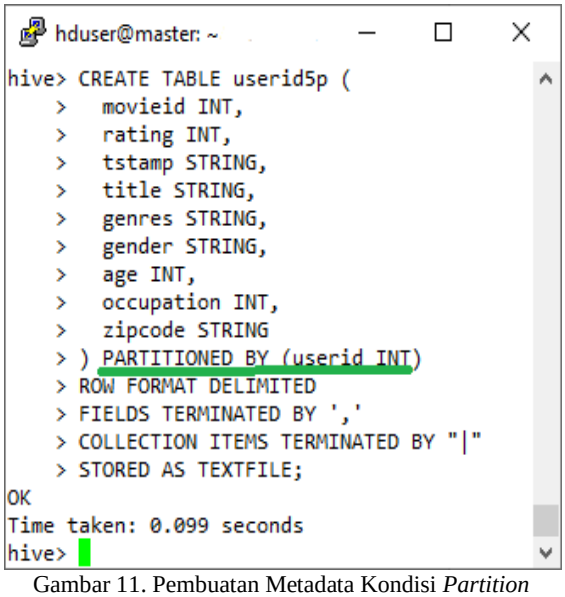
Keberhasilan dalam melakukan *load data* dibuktikan dengan berpindahnya file *dataset* ke direktori kondisi *default* seperti Gambar 10 berikut:

Gambar 10. Tampilan Direktori Kondisi Default

Selanjutnya, untuk implementasi data pada kondisi *default* untuk kardinalitas 50 dan kardinalitas 100 menggunakan cara yang sama pada implementasi kardinalitas 5 yang dilakukan diatas.

2) Implementasi Data pada Kondisi *Partition*

Implementasi data pada kondisi *partition* diawali dengan membuat metadata kondisi *partition* seperti Gambar 11 berikut:



Gambar 11. Pembuatan Metadata Kondisi *Partition*

Sebelum melakukan *insert* data ke tabel dengan metode *partition*, konfigurasi parameter yang diberikan dibawah harus di *set* pada *Hive shell*:

```
hive> set hive.exec.dynamic.partition.mode=nonstrict;
```

Kemudian dilakukan *insert data* dari file data yang telah ada di direktori kondisi *default* menuju ke kondisi *partition* dengan menggunakan *statement* atau *query* seperti pada Gambar 12 berikut:

Gambar 12. Insert Data pada Kondisi *Partition*

Keberhasilan dalam melakukan *insert data* dibuktikan dengan terbentuknya direktori berdasarkan kardinalitas atribut userid yaitu 5 seperti pada Gambar 13 berikut:

Gambar 13. Tampilan Direktori Kondisi *Partition*

Selanjutnya, untuk implementasi data pada kondisi *partition* untuk kardinalitas 50 dan kardinalitas 100 menggunakan cara yang sama pada implementasi kardinalitas 5 yang dilakukan diatas.

C. Hasil Pengujian Performansi

Hasil dari pengujian performansi *response time query* yang telah dilakukan adalah sebagai berikut:

1) Hasil Skenario Pertama

Tabel 1. Hasil Pengujian Skenario Pertama

Skenario *Partition*	**Rata-rata *Response Time* (s) Model *Query***				
	Q1	**Q2**	**Q3**	**Q4**	**Q5**
Default	39.5434	38.6836	39.5958	39.3248	38.4588
Partition	27.0372	26.0532	27.8714	27.338	27.0718

Gambar 14. Diagram Hasil Pengujian Skenario Pertama

Tabel 2. Keterangan Model *Query* Skenario Pertama

Model *Query*	**Kondisi**	***Query***	***Output (records)***
Q1	*default*	select count(*) from userid5 where userid=1;	3.000.000
	partition	select count(*) from userid5p where userid=1;	3.000.000
Q2	*default*	select count(*) from userid5 where userid=2;	3.000.000
	partition	select count(*) from userid5p where userid=2;	3.000.000
Q3	*default*	select count(*) from userid5 where userid=3;	3.000.000
	partition	select count(*) from userid5p where userid=3;	3.000.000
Q4	*default*	select count(*) from userid5 where userid=4;	3.000.000
	partition	select count(*) from userid5p where userid=4;	3.000.000
Q5	*default*	select count(*) from userid5 where userid=5;	3.000.000
	partition	select count(*) from userid5p where userid=5;	3.000.000

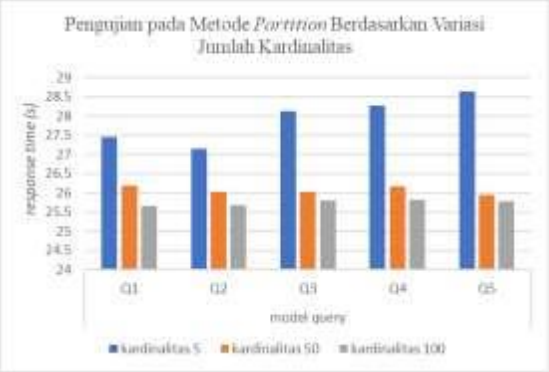
Dari pengujian yang telah dilakukan, pada Gambar 14 terlihat bahwa metode *partition* selain mampu mengelompokkan data, juga memberikan performansi *response time* yang lebih cepat dibandingkan dengan mengeksekusi *query* pada kondisi *default*. Persentase rata-rata *response time query* pada metode *partition* mengalami penurunan sebesar 30.8% dari kondisi *default*.

2) Hasil Skenario Kedua

Tabel 3. Hasil Pengujian Skenario Kedua

Variasi Kardinalitas	**Rata-rata *Response Time* (s) Model *Query***				
	Q1	**Q2**	**Q3**	**Q4**	**Q5**
kardinalitas 5	27.458	26.1456	28.1226	28.273	28.6394
kardinalitas 50	26.1986	26.035	26.0356	26.1714	25.941

kardinalitas 100	25.6666	25.684	25.8066	25.8222	25.7776



Gambar 15. Diagram Hasil Pengujian Skenario Kedua

Tabel 4. Keterangan Model *Query* Skenario Kedua

Variasi Kardinalitas	Model *Query*	*Query*	*Output (records)*
5	Q1	select count(*) from userid5p where userid=1;	3.000.000
	Q2	select count(*) from userid5p where userid=2;	3.000.000
	Q3	select count(*) from userid5p where userid=3;	3.000.000
	Q4	select count(*) from userid5p where userid=4;	3.000.000
	Q5	select count(*) from userid5p where userid=5;	3.000.000
50	Q1	select count(*) from userid50p where userid=1;	300.000
	Q2	select count(*) from userid50p where userid=2;	300.000
	Q3	select count(*) from userid50p where userid=3;	300.000
	Q4	select count(*) from userid50p where userid=4;	300.000
	Q5	select count(*) from userid50p where userid=5;	300.000
100	Q1	select count(*) from userid100p where userid=1;	150.000
	Q2	select count(*) from userid100p where userid=2;	150.000
	Q3	select count(*) from userid100p where userid=3;	150.000
	Q4	select count(*) from userid100p where userid=4;	150.000
	Q5	select count(*) from userid100p where userid=5;	150.000

Dari pengujian yang telah dilakukan, terlihat bahwa pada kardinalitas 100 memberikan performansi *response time query* yang lebih cepat dibandingkan dengan kardinalitas 50 dan kardinalitas 5. Persentase rata-rata *response time query* pada kardinalitas 100 mengalami penurunan sebesar 1.24% dari kardinalitas 50, sedangkan persentase rata-rata *response time query* pada kardinalitas 100 mengalami penurunan sebesar 7.76 % dari kardinalitas 5. Hal ini bisa terjadi karena pada kardinalitas 100, hasil eksekusi *query* (*output*) lebih kecil (150.000 *records*) dibandingkan pada variasi kardinalitas 5 dan 50 (masing-masing 3.000.000 *records* dan 300.000 *records*), kardinalitas 100 lebih baik.

V. KESIMPULAN

Berdasarkan dari hasil penelitian yang telah dilakukan, dapat disimpulkan bahwa metode *partition* selain mampu mengelompokkan data pada Hadoop-Hive, juga memberikan performansi *response time query* yang lebih cepat sebesar 30.8% dibandingkan kondisi *default*. Jika data terdiri dari jumlah *records* yang sama tetapi atribut berbeda kardinalitas, dari yang sudah diujicobakan metode *partition* pada kardinalitas 100 memberikan performansi *response time query* yang lebih baik dibandingkan pada dua kardinalitas lainnya yang lebih kecil yaitu kardinalitas 5 dan kardinalitas 50. Untuk implementasi metode *partition*, dapat diterapkan pada kondisi data yang terpartisi terdistribusi merata, sehingga kecepatan *response time query* pada tiap *uniq values* dari atribut akan tidak berbeda jauh.

UCAPAN TERIMA KASIH

Penulis mengucapkan banyak terima kasih kepada Allah SWT, kedua orang tua, saudara, dan khususnya kepada kedua dosen pembimbing, serta seluruh dosen Teknik Elektro terutama Program Studi Teknik Komputer dan Jaringan.

REFERENSI

[1] A. Gupta, M. Saxena, and R. Gill, "Performance Analysis of RDBMS and Hadoop Components with Their File Formats for the Development of Recommender Systems," *2018 3rd Int. Conf. Converg. Technol. I2CT 2018*, pp. 1–6, 2018.

[2] M. Asha Kiran M and Sreedevi, "Hive Based Geospatial Analysis for Tracking and Envisioning of Geospatial Data in Hadoop Environment," no. 6, pp. 570–573, 2019.

[3] J. Mariam, "An Experimental Study On Different Data Models In Apache Hive," vol. 6, no. 7, pp. 43–51, 2019.

[4] T. A. Ashwitha, A. P. Rodrigues, and N. N. Chiplunkar, "Movie Dataset Analysis using Hadoop-Hive," *2017 2nd Int. Conf. Comput. Syst. Inf. Technol. Sustain. Solut.*, pp. 1–5, 2017.

[5] A. S. Kumar, "Performance Analysis of MySQL Partition , Hive Partition-Bucketing and Apache Pig," 2016.

[6] A. Ramadhana and I. Krisnadi, "Identifikasi Strategi Pendekatan Big Data Yang Tepat Dalam Perusahaan,"

[7] Y. Huai *et al.*, "Major technical advancements in Apache Hive," *Proc. ACM SIGMOD Int. Conf. Manag. Data*, pp. 1235–1246, 2014.

[8] T. DataFlair, "Hive Data Model – Learn to Develop Data Models in Hive," 2020. [Online]. Available: https://data-flair.training/blogs/hive-data-model/.

[9] A. Thusoo *et al.*, "Hive - A petabyte scale data warehouse using hadoop," *Proc. - Int. Conf. Data Eng.*, pp. 996–1005, 2010.