Volume 1 No. 2 (July - December 2021), pp. 83-102 | ISSN 2962-2360 (online) | ISSN 2962-3669 (print)
Arcitech: Journal of Computer Science and Artificial Intelligence
DOI : http://dx.doi.org/10.29240/arcitech.v1i2.4120

Analisis dan Perancangan Feature Engineering pada SQL Execution Plan Aplikasi Apache Spark

Eri Dariato
Fakultas Teknik Informatika
Universitas Dian Nusantara, Jakarta-Indonesia
Email : eri.dariato@undira.ac.id

Article Information

Article history

Received 20 August 2021
Revised 5 October 2021
Accepted 15 November 2021
Available 20 December 2021

Keywords

Database
Artificial Intelligence
Apache Spark
Feature Engineering
SQL

Corresponding Author:

Eri Dariato,
Fakultas Teknik dan Informatika,
Universitas Dian Nusantara, Jakarta
Email : eri.dariato@undira.ac.id

ABSTRACT

To process data in RDBMS, it is commonly used what is called SQL. This SQL can be simple or complex depending on the join statement, the tables involved, filter statement, and many more. If it is complex and burdensome to the system, then the performance of the database as a whole will decrease and SQL execution will not finish. That's why it's important to know, what components are involved when executing SQL through the SQL Execution Plan. The components in the SQL Execution Plan can be very diverse, so artificial intelligence is used to help analyze and conclude what factors have the most role in determining performance. Furthermore, these factors are called feature engineering. The purpose of this research is to formulate what features exist in the SQL Execution Plan when we send SQL commands to the system. The research method used is the Prototyping process model.

Keywords : *Database, Artificial Intelligence, Apache Spark, Feature Engineering, SQL*

ABSTRAK

Untuk memproses basis data RDBMS, sudah umum digunakan apa yang disebut dengan SQL. SQL ini bisa sederhana atau kompleks tergantung kebutuhan dan atribut tabel-tabel yang terlibat. Jika kompleks dan memberatkan sistem, maka performa dari basis data secara keseluruhan itu menjadi menurun dan eksekusi SQL tidak kunjung selesai. Untuk itulah penting untuk tahu, komponen-komponen apa saja yang terlibat saat mengeksekusi SQL melalui SQL Execution Plan. Komponen yang ada pada SQL Execution Plan bisa sangat beragam, karena itu digunakan kecerdasan buatan untuk membantu menganalisa dan menyimpulkan faktor-faktor apa yang paling beperan untuk penentuan performansi. Selanjutnya faktor-faktor itu hasilkan melalui proses yang disebut dengan feature engineering. Tujuan dari penelitian ini adalah merumuskan feature-feature apa saja yang ada pada SQL Execution Plan pada saat kita mengirimkan perintah SQL kedalam sistem. Metode penelitian yang digunakan adalah model proses Prototyping.

Kata Kunci : *Database, Artificial Intelligence, Apache Spark, Feature Engineering, SQL*

1. Pendahuluan

Basis Data sangat bermanfaat untuk sebuah organisasi, kemampuan penyimpanan dan pengelolaan data adalah hal utama yang dapat diandalkan, dengan penggunaan basis data maka pengelolaan dan pengaksesan data akan menjadi lebih mudah dan cepat. Untuk memproses data, digunakan metode yang disebut dengan SQL Query. SQL Query (berikutnya disebut Query) adalah sintaks atau perintah yang digunakan untuk mengakses dan menampilkan data pada sistem database. Query memiliki kemampuan untuk mengatur data mana yang perlu ditampilkan sesuai dengan yang pengguna inginkan. Selain itu, query dapat dipakai untuk membuat data dapat saling berinteraksi. Query bisa sederhana ataupun kompleks. Jika kompleks dan memberatkan sistem, maka performa dari Basis Data itu menjadi menurun. Untuk itulah penting untuk tahu bagaimana melakukan optimasi pada SQL dan salah satunya melalui kecerdasan buatan. Faktor-faktor apakah yang dipertimbangkan pada saat SQL, dituangkan dalam bentuk feature yang nantinya akan diproses oleh model kecerdasan buatan (Artificial Intelligence) [4] .

Pada penelitian ini, platform yang akan digunakan adalah Apache Spark. Apache Spark merupakan mesin analitik terintegrasi untuk memproses data berukuran sangat besar. Spark menyediakan API untuk bahasa Java, Scala, Python dan R, dan mesin optimal yang mendukung eksekusi graph secara umum. Spark juga mendukung perangkat dengan level-tinggi seperti Spark SQL untuk SQL dan pemrosesan data terstruktur, Mllib untuk machine learning, GraphX untuk pemrosesan Graph, dan Streaming untuk komputasi incremental dan pemrosesan streaming [1]. Pada penelitian ini, fokus akan ada pada Spark SQL.

Salah satu permasalahan yang dihadapi saat melakukan proses analitik dengan menggunakan Apache Spark adalah susahnya memprediksi apakah aplikasi yang disubmit ke sistem akan berjalan lancar, berjalan sangat lama, atau bahkan akhirnya terhenti. Proses SQL yang umumnya memiliki durasi lama (misalnya, diatas 12 jam), menjadi semakin lama durasinya, atau terpaksa terhenti karena faktor resource yang ternyata membutuhkan memori yang besar. Berikut bukti dari keluhan saat mengeksekusi perintah SQL ke sistem.

Mungkin mudah saja untuk mengatasi keluhan ini yakni dengan cara menambahkan alokasi daya pada setiap aplikasi. Karena keterbatasan alokasi daya, tentu tidak semua aplikasi bisa mendapat alokasi sumber daya yang sama besar. Oleh karena itu, dibutuhkan pendekatan sehingga SQL Query bisa dieksekusi dengan lancar. Salah satunya dengan mendefinisikan variabel-variabel apa yang sangat berpengaruh. Selanjutnya variabel-variabel ini disebut dengan feature engineering.

Dapat disimpulkan jika rumusan masalah yang akan diselesaikan adalah dari berbagai metrik SQL Execution, bagaimana menentukan variabel yang dapat membantu

penentuan kompleksitas SQL? Sehingga nantinya bisa diketahui apakah SQL tersebut adalah Query yang optimal atau tidak.

2. Kajian Terdahulu

Landasan Teori

a. Sistem

Sistem adalah suatu jaringan kerja dari prosedur-prosedur yang saling berhubungan, berkumpul bersama-sama untuk melakukan suatu kegiatan atau untuk menyelesaikan suatu sasaran tertentu. (Harianto Antonio, Novi Safriadi, 2012)

b. Basis Data

Basis Data adalah sekumpulan data yang terintegrasi, yang diorganisasi untuk memenuhi kebutuhan para pemakai di dalam suatu organisasi. (Ni Ketut Dewi Ari Jayanti, Ni Kadek Sumiari, 2018)

c. Apache Spark

Apache Spark merupakan mesin analitik terintegrasi untuk memproses data berukuran sangat besar. Spark menyediakan API untuk bahasa Java, Scala, Python dan R, dan mesin optimal yang mendukung eksekusi graph secara umum. Spark juga mendukung perangkat dengan level-tinggi seperti Spark SQL untuk SQL dan pemrosesan data terstruktur, Mllib untuk machine learning, GraphX untuk pemrosesan Graph, dan Streaming untuk komputasi inkremental dan pemrosesan streaming (Apache Spark Foundation, 2021)

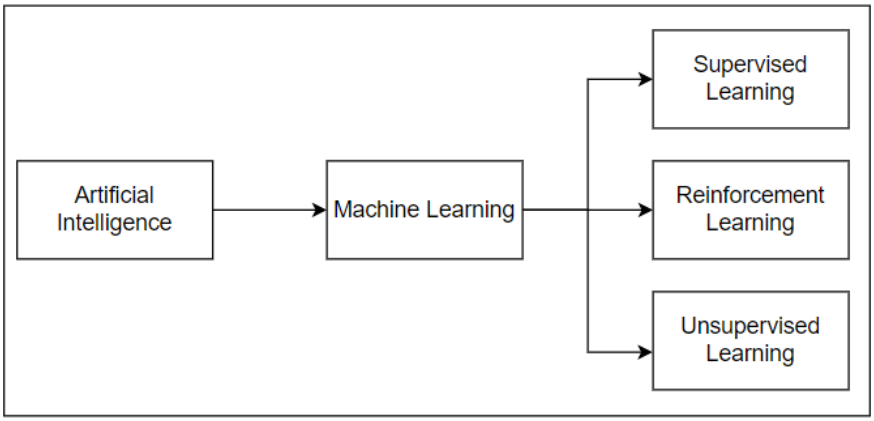
d. *Feature Engineering*

Feature Engineering adalah salah satu fitur utama dari machine learning untuk mengekstrak pola yang berguna dari data yang akan memudahkan model untuk membedakan kelas. Feature Engineering juga merupakan teknik yang paling penting untuk mencapai hasil yang baik pada tugas prediksi.

e. Kecerdasan Buatan

Machine learning dapat didefinisikan sebagai aplikasi komputer dan algoritma matematika yang diadopsi dengan cara pembelajaran yang berasal dari data dan menghasilkan prediksi di masa yang akan datang (Goldberg & Holland, 1988). Adapun proses pembelajaran yang dimaksud adalah suatu usaha dalam memperoleh kecerdasan yang melalui dua tahap antara lain latihan (training) dan pengujian (testing) (Huang, Zhu, & Siew, 2006). Bidang machine learning berkaitan dengan pertanyaan tentang bagaimana membangun program komputer agar meningkat

secara otomatis dengan berdasar dari pengalaman (Mitchell, 1997). Penelitian terkini mengungkapkan bahwa machine learning terbagi menjadi tiga kategori: Supervised Learning, Unsupervised Learning, Reinforcement Learning (Somvanshi & Chavan, 2016). Skema keterkaitan artificial intelligence dan machine learning dapat dijelaskan dalam Gambar 1.



Gambar 1. Skema Kecerdasan Buatan dan *Machine Learning*

Teknik yang digunakan oleh Supervised Learning adalah metode klasifikasi di mana kumpulan data sepenuhnya diberikan label untuk mengklasifikasikan kelas yang tidak dikenal.
Sedangkan teknik Unsupervised Learning sering disebut cluster dikarenakan tidak ada kebutuhan untuk pemberian label dalam kumpulan data dan hasilnya tidak mengidentifikasi contoh di kelas yang telah ditentukan.

Sedangkan Reinforcement Learning biasanya berada antara Supervised Learning dan Unsupervised Learning (Board, 2017), teknik ini bekerja dalam lingkungan yang dinamis di mana konsepnya harus menyelesaikan tujuan tanpa adanya pemberitahuan dari komputer secara eksplisit jika tujuan tersebut telah tercapai (Das & Nene, 2017).

Metode supervised learning didasarkan pada kumpulan sampel data yang memiliki label. Kumpulan sampel digunakan untuk meringkas karakteristik distribusi ukuran perilaku dalam setiap jenis aplikasi sehingga membentuk model perilaku dari data (Amei, Huailin, Qingfeng, & Ling, 2011). Supervised learning dikelompokkan lebih lanjut dalam masalah klasifikasi dan regresi. Masalah klasifikasi adalah ketika variabel output berbentuk kategori, seperti merah atau biru atau penyakit dan tidak ada penyakit. Sedangkan masalah regresi adalah ketika variabel output adalah nilai riil, seperti dollar atau berat (Brownlee, 2016).

Supervised learning memiliki beberapa algoritma populer seperti Back-propagation (Negnevitsky, 2005), Linear regression, Random Forest, Support Vector Machines (Brownlee, 2016), Naive Bayesian, Metode Rocchio, Decision Tree, k-Nearest Neighbor, Neural Network (Darujati & Gumelar, 2012), Logistic Regression, dan Neural Network (Lakshmi & Sheshasaayee, 2015). Kemudian beberapa algoritma untuk klasifikasi pun disebutkan dalam seperti Support Vector Machines (SVM), Normal Bayesian Classifier (NBC), K-Nearest Neighbor (KNN), Trees Gradient Boosted (GBT), Random Trees (RT), dan Artificial Neural Networks (ANN) (Židek, Pitel', & Hošovsk\`y, 2017). Algoritma lainnya pun dibahas dalam (Athmaja, Hanumanthappa, & Kavitha, 2017) seperti Gaussian Mixture models, Hidden Markov Models, logistic regression, Kernel Regression, Deep neural networks, Deep belief networks, PCA, Kernel Perceptron.

f. SQL Query

SQL digunakan untuk memanipulasi dan menarik data yang tersimpan pada IBM database management system yang disebut dengan System R. SQL ini dikembangkan setelah mempelajari model rasional dari manajemen basis data yang ditemukan oleh E. F Codd di awal tahun 1970 (Codd, E.F., A). Berikut ini contoh SQL Query pada gambar dibawah ini.

```
mysql> SELECT * FROM daftar_dosen;
+------------+----------------+------------+-----------+
| NIP        | nama_dosen     | no_hp      | alamat    |
+------------+----------------+------------+-----------+
| 0160436012 | Sabrina Sari   | 0812349900 | Pekanbaru |
| 0260432002 | Maya Ari Putri | 0812345234 | Palembang |
| 0275430005 | Susi Indriani  | 0812656532 | Bogor     |
| 0480432066 | Tia Santrini   | 0812451177 | Padang    |
| 0576431001 | M. Siddiq      | 0812979005 | Jakarta   |
| 0770435006 | Rubin Hadi     | 0812567678 | Papua     |
| 0869437003 | Mustalifah     | 0812338877 | Aceh      |
| 1080432007 | Arif Budiman   | 0812456345 | Makasar   |
+------------+----------------+------------+-----------+
8 rows in set (0.266 sec)
```

Gambar 2. Contoh SQL Query Sederhana

Setiap sintaks SQL akan diubah menjadi *logical plan* dan *physical plan* sebelum dieksekusi oleh sistem basis data.

```sql
SELECT
c.calendar_date,
c.calendar_year,
c.calendar_month,
c.calendar_dayname,
COUNT(DISTINCT sub.order_id) AS num_orders,
COUNT(sub.book_id) AS num_books,
SUM(sub.price) AS total_price,
SUM(COUNT(sub.book_id)) OVER (
  PARTITION BY c.calendar_year, c.calendar_month
  ORDER BY c.calendar_date
) AS running_total_num_books,
LAG(COUNT(sub.book_id), 7) OVER (ORDER BY c.calendar_date) AS prev_books
FROM calendar_days c
LEFT JOIN (
  SELECT
  DATE_FORMAT(co.order_date, '%Y-%m') AS order_month,
  DATE_FORMAT(co.order_date, '%Y-%m-%d') AS order_day,
  co.order_id,
  ol.book_id,
  ol.price
  FROM cust_order co
  INNER JOIN order_line ol ON co.order_id = ol.order_id
) sub ON c.calendar_date = sub.order_day
GROUP BY c.calendar_date, c.calendar_year, c.calendar_month, c.calendar_dayname
ORDER BY c.calendar_date ASC;
```

Gambar 3. Contoh SQL Query Kompleks

g. *Logical Plan* hanya menggambarkan output yang diharapkan setelah menerapkan rangkaian proses transformasi seperti join, filter, where, groupBy, dan lain lain pada tabel-tabel tertentu. *Physical Plan* bertanggung jawab untuk menentukan tipe join , urutan eksekusi filter , where, groupBy dan sebagainya

2.2 Penelitian Terdahulu

Penelitian terdahulu tentang *feature engineering* telah banyak dilakukan, namun sebagai bahan pertimbangan, perbandingan dan sumber referensi dalam penelitian ini maka penulis memilih beberapa penelitian terdahulu sebagai berikut:

Tabel 1. Penelitian Terdahulu

No	Nama	Judul	Tahun
1	Sunarya, Santoso, & Sentanu, 2015	Kecerdasan Buatan merupakan salah satu bidang dalam ilmu komputer yang ditujukan pada pembuatan software dan hardware yang dapat berfungsi sebagai sesuatu yang dapat berpikir seperti manusia	2015
2	Rahardja, Roihan, & others	Kecerdasan buatan banyak digunakan untuk memecahkan berbagai masalah seperti bisnis	2017
3	Russell & Norvig	Kecerdasan buatan juga banyak digunanakan untuk memecahkan masalah seperti robotika,	2016

		bahasa alami,matematika, game, persepsi, diagnosis medis, teknik, analisis keuangan, analisis sains, dan penalaran	
4	Sirait, E. R. E.	Implementasi Teknologi Big Data Di Lembaga Pemerintahan Indonesia.	2016
5	Windarto, A. P., Dewi, L. S., & Hartama, D.	Implementation of Artificial Intelligence in Predicting the Value of Indonesian Oil and Gas Exports With BP Algorithm	2017

3. Metodologi Penelitian

3.1. Metode Pengumpulan Data

Data yang digunakan dalam penelitian ini dikumpulkan dari log sistem aplikasi Apache Spark di perusahaan XYZ. Selain itu, sebagai landasan teori dan referensi, data referensi SQL juga berasal dari ebook, jurnal, buku, observasi, dokumentasi platform, sistem platform dan berbagai informasi lain di Internet.

3.2. Model Proses Prototyping

Model proses yang digunakan dalam penelitian ini adalah model proses prototyping, yaitu:

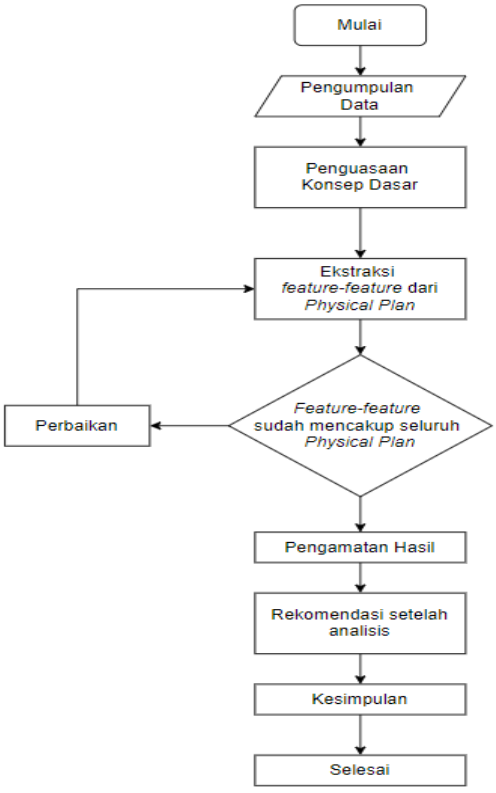
a. Komunikasi
   Tahap komunikasi adalah sebuah tahapan dimana tim peneliti mengadakan komunikasi dengan responder atau nara sumber di tempat penelitian. Di tahap ini juga dilakukan pengumpulan yakni sebuah tahapan dimana tim peneliti mengambil data sumber atau data bukti dari sistem yang terkait.
b. Perencanaan cepat
   Tahap perencanaan cepat adalah sebuah tahapan dimana dibuat perencanaan sumber daya dan fitur-fitur dalam aplikasi yang akan dirancang.
c. Pemodelan
   Pemodelan menggunakan model desain pembelajaran ADDIE yang terdiri dari :
   1. Analisis
      Desain tahap analisis berfokus pada metrik-metrik yang ada di SQL Execution Plan yang ada di Apache Spark.
   2. Design
      Tahap desain terkait dengan penentuan sasaran, konten, dan analisis yang terkait.

3. *Development*
   Dalam tahan pengembangan dilakukan pembuatan dan penggabungan konten yang sudah dirancang pada tahapan desain.
4. Implementation
   Fase ini, dibuat prosedur untuk implementasi variabel-variabel kedalam *feature engineering*
5. Evaluation
   Tahap evaluasi harus memastikan apakah masalah yang ada dapat diatasi dengan variabel-variabel dari hasil penelitian ini

d. Konstruksi
   Tahap konstruksi variabel-variabel kedalam *feature engineering* sistem basis data
e. Deployment
   Pada tahap ini *feature engineering* diujicoba pada kegiatan operasional organisasi.

3.3. Diagram Alir Penelitian

Tahapan proses yang akan dilakukan dalam penelitian ini digambarkan dalam diagram alir pada gambar 5 sebagai berikut :



Gambar 4. Diagram Alir Penelitian

4. Hasil dan Pembahasan

4.1. Hasil

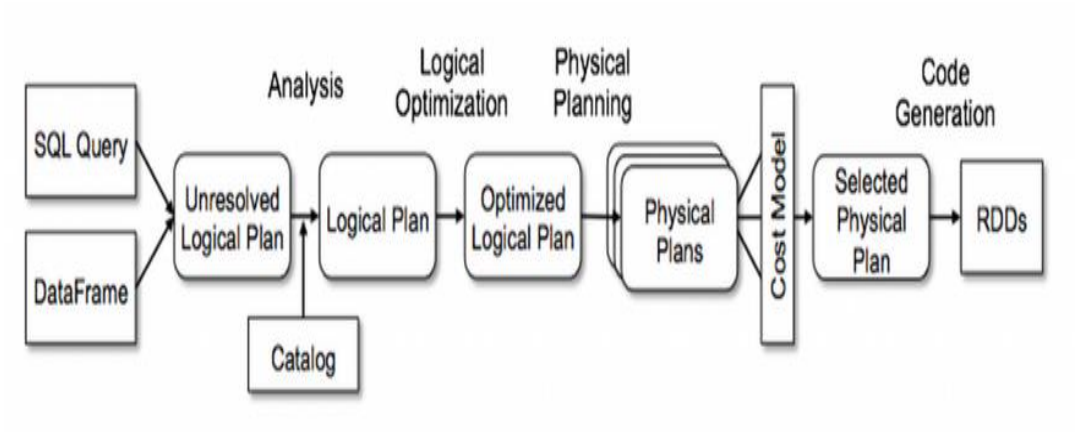
Sintaks SQL adalah sintaks yang fleksibel. Eksekusi SQL ini bisa memakan waktu singkat ataupun lama tergantung dari operasi yang dijalankan. Berikut contoh SQL yang dieksekusi ke Apache Spark.

```
from pyspark.sql.functions import sum

y=(items.join(orders,items.id==orders.itemid, how="inner"))\
        .where(items.id==2)\
        .groupBy("name","price").agg(sum("count")\
        .alias("c"))
```

Gambar 5 Contoh SQL Apache Spark

Saat sintaks SQL diajukan kedalam sistem, maka proses yang terjadi sebenarnya adalah sintaks SQL ini dirubah kedalam *Logical Plan* dan *Physical Plan.* Data *Physical Plan* inilah yang akan dijadikan bahan analisa utama dalam penelitian ini, karena *Physical Plan* adalah eksekusi yang sebenarnya dari sebuah sintaks SQL. Dari semua informasi yang terdapat pada *Physical Plan*, diekstrak *feature-feature* yang berkorelasi dengan kompleksitas dan waktu eksekusi suatu sintaks SQL. Perhatikan diagram dibawah ini, diagram ini menjelaskan alur dari SQL Query menuju ke *Logical Plan* lalu ke *Physical Plan,* sebelum akhirnya di eksekusi kedalam RDD.



Gambar 6 Flow Diagram Query ke Physical Plan

```
== Analyzed Logical Plan ==
name: string, price: float, c: bigint
Aggregate [name#2222, price#2223], [name#2222, price#2223, sum(cast(count#2229 as bigint)) AS c#2239L]
+- Filter ((id#2221 = itemid#2228) AND (id#2221 = 2))
   +- Join Inner
      :- SubqueryAlias items
      :  +- LogicalRDD [id#2221, name#2222, price#2223], false
      +- SubqueryAlias orders
         +- LogicalRDD [id#2227, itemid#2228, count#2229], false

== Optimized Logical Plan ==
Aggregate [name#2222, price#2223], [name#2222, price#2223, sum(cast(count#2229 as bigint)) AS c#2239L]
+- Project [name#2222, price#2223, count#2229]
   +- Join Inner, (id#2221 = itemid#2228)
      :- Filter (isnotnull(id#2221) AND (id#2221 = 2))
      :  +- LogicalRDD [id#2221, name#2222, price#2223], false
      +- Project [itemid#2228, count#2229]
         +- Filter ((itemid#2228 = 2) AND isnotnull(itemid#2228))
            +- LogicalRDD [id#2227, itemid#2228, count#2229], false

== Physical Plan ==
*(4) HashAggregate(keys=[name#2222, price#2223], functions=[finalmerge_sum(merge sum#2274L) AS sum(cast(count#2229 as bigint))#2240L], output=[name#2222, price#2223, c#2239L])
+- Exchange hashpartitioning(name#2222, price#2223, 200), true, [id=#5183]
   +- *(3) HashAggregate(keys=[name#2222, knownfloatingpointnormalized(normalizenanandzero(price#2223)) AS price#2223], functions=[partial_sum(cast(count#2229 as bigint)) AS sum#2274L], output=[name#2222, price#2223, sum#2274L])
      +- *(3) Project [name#2222, price#2223, count#2229]
         +- *(3) SortMergeJoin [id#2221], [itemid#2228], Inner
            :- Sort [id#2221 ASC NULLS FIRST], false, 0
            :  +- Exchange hashpartitioning(id#2221, 200), true, [id=#5172]
            :     +- *(1) Filter (isnotnull(id#2221) AND (id#2221 = 2))
            :        +- *(1) Scan ExistingRDD[id#2221,name#2222,price#2223]
            +- Sort [itemid#2228 ASC NULLS FIRST], false, 0
               +- Exchange hashpartitioning(itemid#2228, 200), true, [id=#5176]
                  +- *(2) Project [itemid#2228, count#2229]
                     +- *(2) Filter ((itemid#2228 = 2) AND isnotnull(itemid#2228))
                        +- *(2) Scan ExistingRDD[id#2227,itemid#2228,count#2229]
```

Gambar 7 Contoh Physical Plan dan Logical Plan

Untuk melakukan ekstraksi dari *Physical Plan* ke dalam bentuk *feature* , dilakukan proses *feature engineering* dibawah ini :

```
import re
import logging
from pyspark.sql.types import StructType, StructField, ArrayType, StringType,
LongType, DoubleType, BooleanType

schema = StructType([
    StructField('application_id', StringType(), nullable=False),
    StructField('query_id', StringType(), nullable=False),
    StructField('event_dt', StringType(), nullable=False),
    StructField('event_date', StringType(), nullable=False),
    StructField('sql_is_valid', BooleanType(), nullable=False),
    StructField("sql_stage", ArrayType(StringType(), False), nullable=True),
    StructField("sql_stage_count", LongType(), nullable=True),
    StructField("sql_table_scanned", ArrayType(StringType(), False), nullable=True),
    StructField("sql_table_scanned_count", LongType(), nullable=True),
    StructField("sql_scan_stage_count", LongType(), nullable=True),
    StructField("sql_filter_stage_count", LongType(), nullable=True),
    StructField("sql_project_stage_count", LongType(), nullable=True),
```

```
    StructField("sql_projected_columns_count", ArrayType(LongType(), False),
nullable=True),
    StructField("sql_shuffle_stage_count", LongType(), nullable=True),
    StructField("sql_sort_stage_count", LongType(), nullable=True),
    StructField("sql_sort_index_count", ArrayType(LongType(), False),
nullable=True),
    StructField("sql_join_stage_count", LongType(), nullable=True),
    StructField("sql_join_index_count", ArrayType(LongType(), False),
nullable=True),
    StructField("sql_join_function", ArrayType(StringType(), False), nullable=True),
    StructField("sql_cross_join_count", LongType(), nullable=True),
    StructField("sql_inner_join_count", LongType(), nullable=True),
    StructField("sql_left_join_count", LongType(), nullable=True),
    StructField("sql_right_join_count", LongType(), nullable=True),
    StructField("sql_outer_join_count", LongType(), nullable=True),
    StructField("sql_broadcast_stage_count", LongType(), nullable=True),
    StructField("sql_aggregate_stage_count", LongType(), nullable=True),
    StructField("sql_aggregate_cols_count", ArrayType(LongType(), False),
nullable=True),
    StructField("sql_aggregate_keys_count", ArrayType(LongType(), False),
nullable=True),
    StructField("sql_unknown_stage", ArrayType(StringType(), False), nullable=True)
])

def safe_list_get (l, idx, default):
    try:
        return l[idx]
    except IndexError:
        return default
```

```
def split_string_extended(string, sep, enclosed_start='"',
enclosed_end=None):
    if not enclosed_end:
        enclosed_end = enclosed_start

    regex = f"{sep}(?![^{enclosed_start}]*[{enclosed_end}]) "
    return re.split(regex, string)
```

```
def get_plan_table_scan_info(plan):
    regex = r',(?![^\[]*[\]]) '
    list_of_value = split_string_extended(plan, ',', '\[', '\]')
    table_name = re.search('^Scan +[hive|parquet|avro|csv]+ (\S+)(.*)$',
list_of_value[0])[1]
    partition_cols = safe_list_get(list_of_value, 4, None)
    partition_filter = safe_list_get(list_of_value, 5, None)

    if partition_cols:
        partition_cols = [x.strip(' ') for x in re.sub(r'(#\w+)', '',
partition_cols.strip('[]')).split(',')]

    if partition_filter:
        partition_filter = [x.strip(' ') for x in re.split(r',(?!(?:[^()]*\([^)]*\))*[^()]*\))',
re.sub(r'(#\w+)', '',  partition_filter.strip('[]')))]

    schema = table_name.split('.')[0]
    table = table_name.split('.')[1]

    return table_name
```

```
def extract_feature(query):
    temp = {
        'sql_is_valid': True,
        'sql_stage': [],
        'sql_stage_count': 0,
        'sql_table_scanned' : [],
        'sql_table_scanned_count': 0,
        'sql_scan_stage_count': 0,
        'sql_filter_stage_count': 0,
        'sql_project_stage_count': 0,
        'sql_projected_columns_count': [],
        'sql_shuffle_stage_count': 0,
        'sql_sort_stage_count': 0,
        'sql_sort_index_count': [],
        'sql_join_stage_count': 0,
        'sql_join_index_count': [],
        'sql_join_function': [],
```

```
        'sql_cross_join_count': 0,
        'sql_inner_join_count':0,
        'sql_left_join_count':0,
        'sql_right_join_count':0,
        'sql_outer_join_count':0,
        'sql_broadcast_stage_count': 0,
        'sql_aggregate_stage_count': 0,
        'sql_aggregate_cols_count': [],
        'sql_aggregate_keys_count': [],
        'sql_unknown_stage': [],
    }
    try:
        plan_list = [re.sub('^([^a-zA-Z]+)', '', plan) for plan in query.split("\n")]

        for plan in plan_list[:0:-1]:
            if re.match('^Scan +[hive|parquet|avro|csv]+ .*$', plan):
                table_name = get_plan_table_scan_info(plan)
                temp.get('sql_table_scanned',[]).append(table_name)
                temp.get('sql_stage', []).append('Scan')
            elif plan.startswith('Filter'):
                temp.get('sql_stage', []).append('Filter')
            elif plan.startswith('Project'):
                temp.get('sql_stage', []).append('Project')
                project_list = re.findall(r'\[(.*)\]', plan)
                if len(project_list) > 0:
                    cols_projected = len(project_list[0].split(','))
                    temp.get('sql_projected_columns_count', []).append(cols_projected)
            elif any(plan.startswith(s) for s in ['SortAggregate', 'HashAggregate',
'ObjectHashAggregate']):
                temp.get('sql_stage', []).append('Aggregate')
                cols_agg = len(re.findall(r'function[|s]\=\[(.*)\]', plan)[0].split(','))
                temp.get('sql_aggregate_cols_count', []).append(cols_agg)
            elif any(plan.startswith(s) for s in ['SortMergeJoin', 'BroadcastHashJoin',
'BroadcastNestedLoopJoin']):
                if plan.startswith('BroadcastNestedLoopJoin'):
                    temp.get('sql_stage', []).append('CrossJoin')
                    join_keys = 0
                else:
```

```
                temp.get('sql_stage', []).append('Join')
                join_keys = len(re.findall(r'\[(.*)\]', plan)[0].split(','))

            temp.get('sql_join_function', []).append(plan.split(',')[-1])
            temp.get('sql_join_index_count', []).append(join_keys)

        elif plan.startswith('Exchange'):
            temp.get('sql_stage', []).append('Shuffle')
        elif plan.startswith('Broadcast'):
            temp.get('sql_stage', []).append('Broadcast')
        elif plan.startswith('Sort'):
            temp.get('sql_stage', []).append('Sort')
            sort_idx = len(re.findall(r'\[(.*)\]', plan)[0].split(','))
        else:
            stage = plan.split(' ')[0]
            temp.get('sql_unknown_stage', []).append(stage)

        temp.update({'sql_stage_count': len(temp.get('sql_stage',[]))})
        temp.update({'sql_table_scanned_count':
len(temp.get('sql_table_scanned',[]))})

        stage_pair_list = [
            ('sql_scan_stage_count', 'Scan'),
            ('sql_filter_stage_count', 'Filter'),
            ('sql_project_stage_count', 'Project'),
            ('sql_shuffle_stage_count', 'Shuffle'),
            ('sql_broadcast_stage_count', 'Broadcast'),
            ('sql_aggregate_stage_count', 'Aggregate'),
            ('sql_sort_stage_count', 'Sort'),
            ('sql_join_stage_count', 'Scan'),
            ('sql_cross_join_count', 'Join')
        ]

        for (key, item) in stage_pair_list:
            temp.update(
                {key: len([ x for x in temp.get('sql_stage',[]) if item in x])}
            )
```

```
        join_function_pair_list = [
            ('sql_inner_join_count', 'Inner'),
            ('sql_left_join_count', 'Left'),
            ('sql_right_join_count', 'Right'),
            ('sql_outer_join_count', 'Outer')
        ]

        for (key, item) in join_function_pair_list:
            temp.update(
                {key: len([ x for x in temp.get('sql_join_function',[]) if item in x])}
            )

    return temp
except Exception as e:
    temp["sql_is_valid"] = False
    logging.exception("Something awful happened!")
    print(query)
    print("=========== ^^^ QUERY ERROR ^^^ ===========")
    return temp
```

Ada 24 *feature* yang dihasilkan dari proses *feature engineering* diatas. Detailny akan ada di bab pembahasan.

4.2. Pembahasan

Dari hasil iterasi yang dilakukan, maka berikut adalah *feature-feature* yang dapat digunakan untuk menentukan kompleksitas SQL :

Tabel 2 **Daftar *Features***

No	***Features***	**Tipe Data**	**Deskripsi *Features***
1	sql_stage	Array String	Urutan tahapan yang dijalankan SQL
2	sql_stage_count	Long Type	Jumlah urutan tahapan yang dijalankan
3	sql_table_scanned	Array String	Nama tabel yang terlibat di SQL
4	sql_table_scanned_count	Long Type	Jumlah tabel yang terlibat di SQL
5	sql_scan_stage_count	Long Type	Jumlah tahapan yang di *scan*

6	sql_filter_stage_count	Long Type	Jumlah tahapan yang ada di sintaks filter
7	sql_project_stage_count	Long Type	Jumlah tahapan dalam keseluruhan query
8	sql_projected_columns_count	Array Long	Jumlah kolom yang diproyeksi oleh *logical plan*
9	sql_shuffle_stage_count	Long Type	Jumlah tahapan yang akan di *shuffle*
10	sql_sort_stage_count	Long Type	Jumlah tahapan yang akan diurut (*sort*)
11	sql_sort_index_count	Array Long	Jumlah index yang akan diurut
12	sql_join_stage_count	Long Type	Jumlah tahapan yang akan di join
13	sql_join_index_count	Array Long	Jumlah index yang akan di join
14	sql_join_function	Array String	Kumpulan fungsi yang terlibat dalam join
15	sql_cross_join_count	Long Type	Jumlah cross join yang terlibat
16	sql_inner_join_count	Long Type	Jumlah inner join yang terlibat
17	sql_left_join_count	Long Type	Jumlah left join yang terlibat
18	sql_right_join_count	Long Type	Jumlah right join yang terlibat
19	sql_outer_join_count	Long Type	Jumlah outer join yang terlibat
20	sql_broadcast_stage_count	Long Type	Jumlah tahapan broadcast
21	sql_aggregate_stage_count	Long Type	Jumlah tahapan aggregasi
22	sql_aggregate_cols_count	Array Long	Jumlah kolom yang di aggregasi
23	sql_aggregate_keys_count	Array Long	Jumlah *keys* yang terlibat di aggregasi
24	sql_unknown_stage	Array String	tahapan yang tidak teridentifikasi

Dari definisi *features* yang dihasilkan, berikut contoh data yang digunakan.

Tabel 3 **Tabel Contoh Data** ***Features***

No	***Features***	**Contoh Data**
1	sql_stage_count	1
2	sql_table_scanned	[niw.loena_loan_payment_update]
3	sql_table_scanned_count	1

4	sql_scan_stage_count	1
5	sql_filter_stage_count	4
6	sql_project_stage_count	4
7	sql_projected_columns_count	[1, 1, 1, 2]
8	sql_shuffle_stage_count	5
9	sql_sort_stage_count	4
10	sql_sort_index_count	[]
11	sql_join_stage_count	0
12	sql_join_index_count	[2, 2, 2]
13	sql_join_function	[LeftOuter, LeftOuter, LeftOuter, LeftOuter, LeftOuter]
14	sql_cross_join_count	3
15	sql_inner_join_count	0
16	sql_left_join_count	3
17	sql_right_join_count	0
18	sql_outer_join_count	3
19	sql_broadcast_stage_count	0
20	sql_aggregate_stage_count	10
21	sql_aggregate_cols_count	[5, 4, 13, 13, 73, 73, 5, 4, 13, 13, 52, 52, 5, 4, 13, 13, 28, 28]
22	sql_aggregate_keys_count	[]
23	sql_unknown_stage	[FileScan, Scan, InMemoryRelation, InMemoryTableScan]
24	Sql_stage	[Scan, Aggregate, Shuffle, Aggregate, Sort]

Dari seluruh variabel yang dihasilkan, maka dapat kita kategorikan menjadi operasi atau proses dibawah ini :

1. Tabel yang terlibat
2. Proses *stage* yang terlibat
3. Proses *scan* yang terlibat
4. Proses *filter*
5. Proses *sorting*
6. Sintaks *Aggregation*
7. Sintaks *Broadcast*
8. Penyatuan (*join*) tabel

Nantinya dengan *feature-feature* ini, dapat ditentukan tingkat kompleksitas dari suatu sintaks SQL. *Feature-feature* ini merupakan input dari suatu *Machine Learning* yang mana akan memprediksi apakah suatu sintaks SQL Query merupakan sintaks yang kompleks atau tidak, dan mampu memprediksi apakah suatu job membutuhkan alokasi daya yang cukup atau kurang.

5. Kesimpulan

Berdasarkan penelitian dan hasil analisis yang dilakukan, dapat diperoleh kesimpulan bahwa dari *Physical Plan* yang dihasilkan oleh sebuah sintaks SQL, bisa dikeluarkan *feature* yang terkait dengan kompleksitas dan waktu eksekusi SQL, diperlukan proses *feature engineering* untuk menghasilkan *features,* total 24 features yang dihasilkan, dapat dibagi kedalam operasi / bidang :

a. Tabel yang terlibat
b. Proses *stage* yang terlibat
c. Proses *scan* yang terlibat
d. Proses *filter*
e. Proses *sorting*
f. Sintaks *Aggregation*
g. Sintaks *Broadcast*
h. Penyatuan (*join*) tabel

Seluruh variabel yang dihasilkan dari penelitian ini akan dimasukkan kedalam model *Machine Learning*. Tujuan utamanya adalah untuk bisa menentukan apakah sintaks SQL yang diproses akan berhasil atau gagal di tengah proses.

5.2. Ucapan Terima Kasih

Kami berharap semoga penelitian ini dapat digunakan sebagai informasi yang dapat menambah wawasan tentang cara merancang *feature engineering* yang efektif sebelum membangun model *machine learning*, serta berguna bagi pihak-pihak yang terkait dan dapat digunakan sebagai referensi untuk penelitian selanjutnya.

Penelitian ini dapat kami selesaikan berkat bantuan dari berbagai pihak yang mendukung, untuk itu tidak lupa kami ucapkan terima kasih kepada:

1. Rektor Universitas Dian Nusantara
2. Direktur LRPM Universitas Dian Nusantara
3. Dekan Fakultas Teknik Universitas Dian Nusantara
4. Kaprodi Teknik Informatika Universitas Dian Nusantara
5. Seluruh pihak yang tidak dapat kami sebutkan satu persatu yang telah membantu penelitian ini.

6. Pernyataan Penulis

Penulis menyatakan bahwa tidak ada konflik kepentingan terkait publikasi artikel ini. Penulis menegaskan bahwa data dan makalah bebas dari plagiarisme.

Bibliografi

Apache Hadoop. (2022). Diambil kembali dari http://hadoop.apache.org/.

Apache Spark. (2022). Diambil kembali dari https://spark.apache.org/.

Armbrust, M., Huai, Y., Liang, C., Xin, R., & Zaharia, M. (2015, April 13). *Deep Dive into Spark SQL's Catalyst Optimizer*. Diambil kembali dari https://databricks.com: https://databricks.com/blog/2015/04/13/deep-dive-into-spark-sqls-catalyst-optimizer.html

Chanowich, E. d. (2001). *Query Optimization Advanced*.

Goutam, S. (2021, Februari 12). *Apache Spark Logical And Physical Plans*. Diambil kembali dari https://blog.clairvoyantsoft.com/: https://blog.clairvoyantsoft.com/spark-logical-and-physical-plans-469a0c061d9e

Han, J. a. (2000). *Data Mining Concepts & Techniques*. Morgan Kaufmann Publishers.

Harianto Antonio, Novi Safriadi. (2012). Rancang Bangun Sistem Informasi Administrasi Informatika.

Korth, H. d. (1991). *Database System Concepts*. Singapura: McGraw Hill.

Leturgez, L. (2020, Juli 23). *Spark's Logical and Physical plans … When, Why, How and Beyond*. Diambil kembali dari http://www.medium.com: https://medium.com/datalex/sparks-logical-and-physical-plans-when-why-how-and-beyond-8cd1947b605a

Ni Ketut Dewi Ari Jayanti, Ni Kadek Sumiari. (2018). *Teori Basis Data*. Yogyakarta: Penerbit ANDI.

Rahardja, U. R. (2017). Design of Business Intelligence in Learning Systems Using iLearning Media. *Universal Journal of Management*, 227-235.

Russell, S. J. (2016). *Artificial Intelligence : a modern approach*. Malaysia: Pearson Education Limited.

Sunarya, A. S. (2015). Sistem Pakar Untuk Mendiagnosa Gangguan Jaringan Lan. *CCIT*, 8(2), 1-11.

Wahono, R. S. (2014, Januari 10). *romisatriawahono.net/2014/01/10/kontribusi-penelitian-dan-perbaikan-metode/*. Diambil kembali dari romisatriawahono.net: https://romisatriawahono.net/2014/01/10/kontribusi-penelitian-dan-perbaikan-metode/

Y. Bengio, A. C. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1798–1828