

**Laporan Tugas Proyek IF4044 Teknologi Big Data
Semester Genap Tahun Ajaran 2024/2025**

**Eksplorasi *Data Lakehouse*, *Data Mart*,
dan *Data Orchestration***

Oleh:

13521170 HAZIQ ABIYYU MAHDY

18221119 NATASYA VERCELLY HARIJADI

18221130 RAYHAN MAHESWARA PRAMANDA

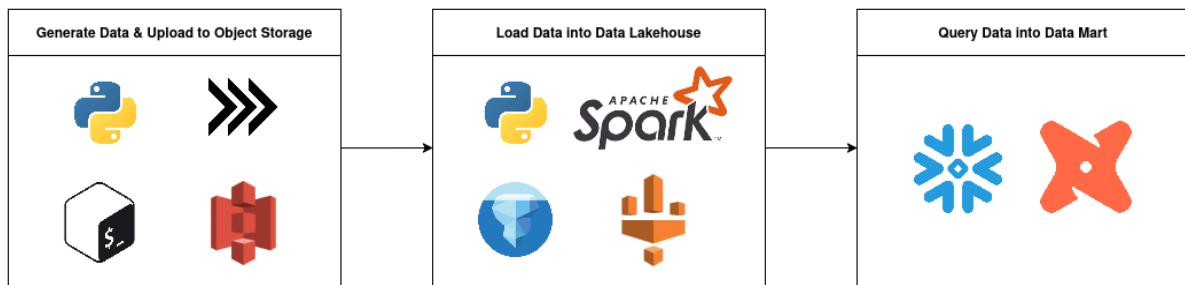


**PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
JUNI 2025**

Daftar Isi

Daftar Isi	2
1. Penjelasan Alur dan Arsitektur yang Digunakan	3
1) Membangkitkan dataset TPC-H dan meng-upload ke Object Storage	3
2) Memasukkan data ke Data Lakehouse	8
3) Memproses data ke Data Mart	16
2. Other Attempt	24
3. Lesson Learned	30
4. Repository	46

1. Penjelasan Alur dan Arsitektur yang Digunakan



Gambar 1.1 Alur dan Arsitektur yang digunakan

Alur dan arsitektur yang digunakan dijelaskan seperti pada Gambar 1.1. Terdapat tiga tahapan utama, yaitu:

1) Membangkitkan dataset TPC-H dan meng-upload ke Object Storage

Untuk membangkitkan dataset TPC-H, kami menggunakan *script* dari *repository* berikut (<https://github.com/electrum/tpch-dbgen/>). *Script* ini membangkitkan *file dataset* dalam format TBL (mirip CSV namun tanpa skema) dan menggunakan karakter *pipe* 'l' sebagai *separator*). Karena spesifikasi tugas memerlukan format CSV dan Parquet, kami membuat *script python* tambahan untuk konversi data ke format tersebut sebagai berikut.

- Berikut adalah cuplikan skema *database* TPC-H dalam Python *dictionary* berguna untuk menyimpan skema pada *file CSV* dan Parquet. Beberapa baris kode tidak ditampilkan agar *screenshot* tidak terlalu besar.

```
1 tpch_schema = {
2     "CUSTOMER": [
3         "column_names": [
4             'C_CUSTKEY', 'C_NAME', 'C_ADDRESS', 'C_NATIONKEY',
5             'C_PHONE', 'C_ACCTBAL', 'C_MKTSEGMENT', 'C_COMMENT'
6         ],
7         "dtypes": {
8             'C_CUSTKEY': int,
9             'C_NAME': str,
10            'C_ADDRESS': str,
11            'C_NATIONKEY': int,
12            'C_PHONE': str,
13            'C_ACCTBAL': float,
14            'C_MKTSEGMENT': str,
15            'C_COMMENT': str
16        }
17    },
18    # ...
19 }
```

Gambar 1.2 Cuplikan skema TPC-H

- Berikut adalah fungsi untuk mengubah *file* TBL ke CSV dan Parquet (*tbl_to_csv_parquet.py*). Beberapa baris kode tidak ditampilkan agar *screenshot* tidak terlalu besar. Untuk membaca *file* TBL dan menuliskan ke CSV, digunakan *library* Pandas, sedangkan untuk menuliskan ke Parquet, digunakan *library* PyArrow.

```
1 def tbl_to_csv_parquet(
2     input_tbl_file: str,
3     output_parquet_file: str,
4     output_csv_file: str,
5     table_name: str,
6     logger=None):
7
8     if logger is None:
9         logger = logging.getLogger(__name__)
10
11     column_names = tpch_schema[table_name]["column_names"]
12     dtypes = tpch_schema[table_name]["dtypes"]
13
14     logger.info(f"Reading {input_tbl_file}...")
15
16     try:
17         df = pd.read_csv(
18             input_tbl_file,
19             sep='|',
20             header=None,
21             skipinitialspace=False,
22             keep_default_na=False,
23         )
24         # ...
```

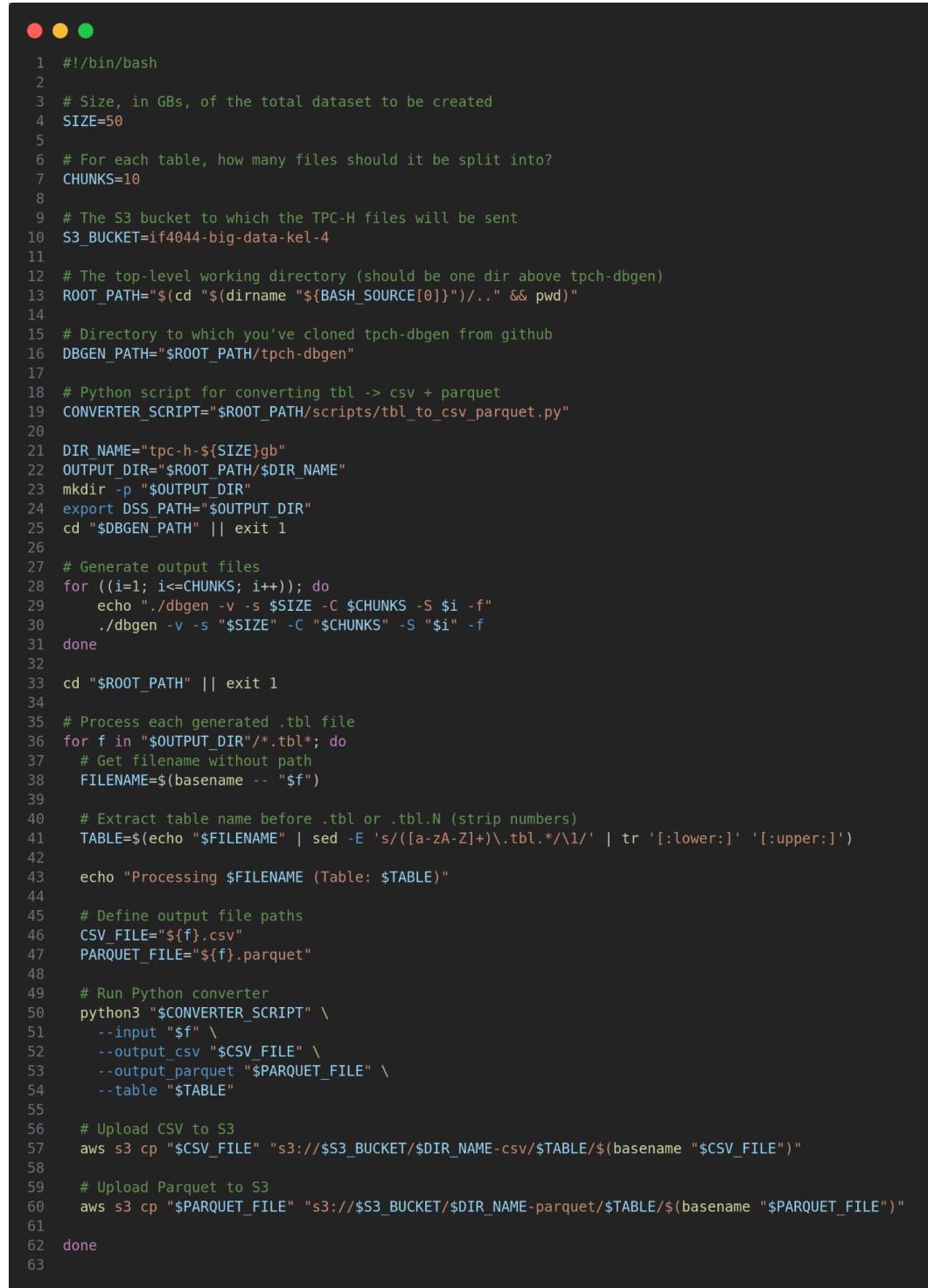
```
25     df.to_csv(output_csv_file, sep=",", index=False)
26     logger.info(f"Successfully converted {input_tbl_file} to {output_csv_file}")
27
28     schema = create_spark_compatible_schema(df, table_name)
29     table = pa.Table.from_pandas(df, schema=schema)
30
31     pq.write_table(
32         table,
33         output_parquet_file,
34         compression='snappy',
35         use_deprecated_int96_timestamps=False,
36         coerce_timestamps='ms',
37         allow_truncated_timestamps=True
38     )
39
40     logger.info(f"Successfully converted {input_tbl_file} to {output_parquet_file}")
41     logger.info(f"Parquet file saved at: {output_parquet_file}")
42
43 except Exception as e:
44     logger.error(f"Error processing {input_tbl_file}: {str(e)}")
45     raise
```

Gambar 1.3 Fungsi untuk mengubah *file* TBL ke CSV dan Parquet

Untuk menyimpan *file* yang telah dibangkitkan, kami memilih menggunakan *Object Storage* yaitu S3 daripada menyimpan secara lokal di HDFS, agar pembangkitan data cukup dilakukan satu kali dan langkah selanjutnya dapat menggunakan *file* yang telah tersimpan di S3. Dengan demikian, masing-masing anggota tidak perlu lagi membangkitkan *file* 50GB secara lokal. Selain itu, S3 juga cukup mudah digunakan.

Untuk meng-*upload* file ke S3, kami menggunakan *bash script* yang berasal dari *repository* berikut (<https://github.com/zecksr/tpch-dbggen-to-aws-s3>) sehingga tidak perlu *upload* secara manual. *Bash script* tersebut memanggil *script* tpc-h dbgen serta meng-*upload* hasilnya ke S3. *Bash script* tersebut kami sesuaikan untuk memanggil Python *script* tbl_to_csv_parquet.py sebelum meng-*upload* *file* CSV dan Parquet yang dihasilkan ke S3 *bucket* yang telah dibuat. Berhubung ukuran total *file* cukup besar, kami membangkitkan *file* dalam beberapa *chunk*, karena membangkitkan semua data dalam satu *file* sekaligus membuat program kehabisan memori. Untungnya, *bash script* tersebut sudah menangani hal tersebut dan menyediakan parameter CHUNKS yang menyatakan berapa banyak *chunk*

yang dibutuhkan dalam satu tabel pada *dataset TPC-H*. Berikut adalah *bash script* yang telah kami modifikasi.



```
1  #!/bin/bash
2
3  # Size, in GBs, of the total dataset to be created
4  SIZE=50
5
6  # For each table, how many files should it be split into?
7  CHUNKS=10
8
9  # The S3 bucket to which the TPC-H files will be sent
10 S3_BUCKET=if4044-big-data-kel-4
11
12 # The top-level working directory (should be one dir above tpch-dbggen)
13 ROOT_PATH=$(cd "$(dirname "${BASH_SOURCE[0]}")/.." && pwd)
14
15 # Directory to which you've cloned tpch-dbggen from github
16 DBGEN_PATH="$ROOT_PATH/tpch-dbggen"
17
18 # Python script for converting tbl -> csv + parquet
19 CONVERTER_SCRIPT="$ROOT_PATH/scripts/tbl_to_csv_parquet.py"
20
21 DIR_NAME="tpc-h-${SIZE}gb"
22 OUTPUT_DIR="$ROOT_PATH/$DIR_NAME"
23 mkdir -p "$OUTPUT_DIR"
24 export DSS_PATH="$OUTPUT_DIR"
25 cd "$DBGEN_PATH" || exit 1
26
27 # Generate output files
28 for ((i=1; i<=CHUNKS; i++)); do
29     echo "./dbgen -v -s $SIZE -C $CHUNKS -S $i -f"
30     ./dbgen -v -s "$SIZE" -C "$CHUNKS" -S "$i" -f
31 done
32
33 cd "$ROOT_PATH" || exit 1
34
35 # Process each generated .tbl file
36 for f in "$OUTPUT_DIR"/*.tbl; do
37     # Get filename without path
38     FILENAME=$(basename -- "$f")
39
40     # Extract table name before .tbl or .tbl.N (strip numbers)
41     TABLE=$(echo "$FILENAME" | sed -E 's/([a-zA-Z]+)\.tbl.*\//\1/' | tr '[[:lower:]]' '[[:upper:]]')
42
43     echo "Processing $FILENAME (Table: $TABLE)"
44
45     # Define output file paths
46     CSV_FILE="${f}.csv"
47     PARQUET_FILE="${f}.parquet"
48
49     # Run Python converter
50     python3 "$CONVERTER_SCRIPT" \
51         --input "$f" \
52         --output_csv "$CSV_FILE" \
53         --output_parquet "$PARQUET_FILE" \
54         --table "$TABLE"
55
56     # Upload CSV to S3
57     aws s3 cp "$CSV_FILE" "s3://$S3_BUCKET/$DIR_NAME-csv/$TABLE/$(basename \"$CSV_FILE\")"
58
59     # Upload Parquet to S3
60     aws s3 cp "$PARQUET_FILE" "s3://$S3_BUCKET/$DIR_NAME-parquet/$TABLE/$(basename \"$PARQUET_FILE\")"
61
62 done
63
```

Gambar 1.4 *Bash script* untuk meng-upload file TPC-H ke S3

Berikut adalah tampilan *folder S3* yang telah berisi *file TPC-H* dalam format CSV dan Parquet.

The screenshot shows the AWS S3 console with the path `Amazon S3 > Buckets > if4044-big-data-kel-4 > tpc-h-50gb-csv/`. The left sidebar shows general purpose buckets and storage lens settings. The main area displays the contents of the `tpc-h-50gb-csv/` folder, which contains 8 objects (folders) with the following names: CUSTOMER/, LINEITEM/, NATION/, ORDERS/, PART/, PARTSUPP/, REGION/, and SUPPLIER/. The objects are listed in a table with columns for Name, Type, Last modified, Size, and Storage class.

Gambar 1.5 *Folder S3* berisi *file TPC-H* dalam format CSV

The screenshot shows the AWS S3 console with the path `Amazon S3 > Buckets > if4044-big-data-kel-4 > tpc-h-50gb-parquet/`. The left sidebar shows general purpose buckets and storage lens settings. The main area displays the contents of the `tpc-h-50gb-parquet/` folder, which contains 8 objects (folders) with the following names: CUSTOMER/, LINEITEM/, NATION/, ORDERS/, PART/, PARTSUPP/, REGION/, and SUPPLIER/. The objects are listed in a table with columns for Name, Type, Last modified, Size, and Storage class.

Gambar 1.6 *Folder S3* yang telah berisi *file TPC-H* dalam format Parquet

2) Memasukkan data ke *Data Lakehouse*

Untuk memasukkan data ke *Data Lakehouse*, kami menggunakan Spark sebagai kakas pemrosesan data dan Iceberg sebagai *open table format*. Untuk menjalankan Spark, kami menggunakan AWS Glue dengan alasan berikut.

- Menjalankan *standalone* Spark secara lokal akan membutuhkan waktu lama karena ukuran data cukup besar. Spark didesain untuk berjalan pada kluster secara terdistribusi, sehingga menggunakan *standalone* Spark tidak memberikan keuntungan performa yang signifikan dibandingkan dengan menggunakan Python dengan *library* umum seperti Pandas.
- Menjalankan Spark pada *cluster* di *cloud* seperti AWS EMR (Elastic MapReduce) memerlukan *setup* yang cukup banyak seperti *provisioning* dan sebagainya.
- AWS Glue merupakan pilihan yang tepat karena *serverless*, sehingga tidak memerlukan terlalu banyak *setup* dan mudah digunakan tetapi mempunyai performa yang cukup baik. Selain itu, AWS Glue juga menyediakan Glue catalog, sehingga kami tidak perlu menyiapkan Hive Metastore sebagai catalog untuk *data lake*.

Berikut adalah fungsi untuk membaca *file* Parquet dari S3, lalu memasukkannya ke Iceberg. Sebelum memasukkan data ke Iceberg, kami membuat *namespace* terlebih dahulu.

The screenshot shows a Jupyter Notebook interface with the title "IF4044-iceberg". The notebook contains the following Python code:

```
[3]: spark.sql("CREATE NAMESPACE IF NOT EXISTS iceberg_catalog.db")
DataFrame[]

[7]: def parquet_to_iceberg(s3_folder_uri: str, table_name: str, partition_by: list[str] | None = None):
    df = spark.read.parquet(f"{s3_folder_uri}/{table_name}.parquet")

    writer = df.writeTo(f"iceberg_catalog.db.{table_name}") \
        .tableProperty("format-version", "2")

    if partition_by is not None:
        writer = writer.partitionedBy(partition_by)

    writer.createOrReplace()

[5]: parquet_location = "s3://if4044-big-data-kel-4/tpc-h-50gb-parquet"
```

Gambar 1.7 Fungsi untuk memasukkan data ke Iceberg

Berikut adalah pemanggilan fungsi tersebut.

```
[5]: parquet_location = "s3://if4044-big-data-kel-4/tpc-h-50gb-parquet"
```

```
[8]: parquet_to_iceberg(f'{parquet_location}/CUSTOMER', "customer")
parquet_to_iceberg(f'{parquet_location}/LINEITEM', "lineitem")
parquet_to_iceberg(f'{parquet_location}/NATION', "nation")
parquet_to_iceberg(f'{parquet_location}/ORDERS', "orders")
parquet_to_iceberg(f'{parquet_location}/PART', "part")
parquet_to_iceberg(f'{parquet_location}/PARTSUPP', "partsupp")
parquet_to_iceberg(f'{parquet_location}/REGION', "region")
parquet_to_iceberg(f'{parquet_location}/SUPPLIER', "supplier")
```

Gambar 1.8 Pemanggilan *fungsi* untuk memasukkan data ke Iceberg

Berikut adalah hasil *query* pada tabel Iceberg yang telah dibuat.

```
[10]: spark.table("iceberg_catalog.db.customer").show(5)
+-----+-----+-----+-----+-----+-----+-----+
|C_CUSTKEY| C_NAME| C_ADDRESS|C_NATIONKEY| C_PHONE|C_ACCTBAL|C_MKTSEGMENT| C_COMMENT|
+-----+-----+-----+-----+-----+-----+-----+
| 1|Customer#000000001| IVhzIApeRBo, ot,c,E| 15|25-989-741-2988| 711.56| BUILDING|to the even, regu...
| 2|Customer#000000002|XSTf4,NcwDVaWNe6t...| 13|23-768-687-3665| 121.65| AUTOMOBILE|l accounts. blith...
| 3|Customer#000000003| MG9kqTD2WBHm| 1|11-719-748-3364| 7498.12| AUTOMOBILE| deposits eat sly...
| 4|Customer#000000004| XxVSJjsLAGtn| 4|14-128-190-5944| 2866.83| MACHINERY| requests. final...
| 5|Customer#000000005|KvpyuHCplrB84WgAi...| 3|13-750-942-6364| 794.47| HOUSEHOLD|n accounts will h...
+-----+-----+-----+-----+-----+-----+-----+
only showing top 5 rows
```

```
[11]: spark.table("iceberg_catalog.db.lineitem").show(5)
+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+
|L_ORDERKEY|L_PARTKEY|L_SUPPKEY|L_LINENUMBER|L_QUANTITY|L_EXTENDEDPRICE|L_DISCOUNT|L_TAX|L_RETURNFLAG|L_LINESTATUS|L_SHIPDATE|L_COMMITDATE|L_RECEIPTDATE|L_SHIPINSTRUCT|L_SHIPMODE| L_COMMENT|
+-----+-----+-----+-----+-----+-----+-----+
| 6290498| 6462217| 212242| 3| 12.0| 14146.68| 0.07| 0.05| A| F|1992-11-0
5| 1992-11-08| 1992-11-08| NONE| RAIL| slow excuses about |
| 6290498| 469060| 219061| 4| 5.0| 5145.2| 0.0| 0.05| R| F|1993-01-1
5| 1992-11-26| 1993-01-30| COLLECT COD| TRUCK| e ideas. ent|
| 6290499| 6375209| 125234| 1| 31.0| 39800.59| 0.08| 0.08| R| F|1994-01-0
3| 1994-03-21| 1994-01-10| COLLECT COD| MAIL|elets promise alo...
| 6290499| 8123390| 248407| 2| 4.0| 5651.96| 0.06| 0.06| R| F|1994-02-2
5| 1994-03-15| 1994-03-02| COLLECT COD| TRUCK| o beans cou|
| 6290499| 9815627| 440647| 3| 49.0| 75564.37| 0.05| 0.0| R| F|1994-02-2
7| 1994-02-26| 1994-03-27| COLLECT COD| REG AIR| furiously regula...
+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+
only showing top 5 rows
```

```
[9]: spark.table("iceberg_catalog.db.nation").show(5)
+-----+-----+-----+
|N_NATIONKEY| N_NAME|N_REGIONKEY|          N_COMMENT|
+-----+-----+-----+
|      0| ALGERIA|          0|haggle. carefull...
|      1|ARGENTINA|          1|al foxes promise ...
|      2| BRAZIL|          1|y alongside of th...
|      3| CANADA|          1|eas hang ironic, ...
|      4| EGYPT|          4|y above the caref...
+-----+-----+-----+
only showing top 5 rows
```

```
[12]: spark.table("iceberg_catalog.db.orders").show(5)
+-----+-----+-----+-----+-----+-----+-----+-----+
|O_ORDERKEY|O_CUSTKEY|O_ORDERSTATUS|O_TOTALPRICE|O_ORDERDATE|O_ORDERPRIORITY|      O_CLERK|O_SHIPPRIORITY|      O_COMM
ENT|
+-----+-----+-----+-----+-----+-----+-----+-----+
| 102582913| 1026650|        O| 111074.28| 1996-06-04|      1-URGENT|Clerk#000008415|      0|ctions boost sly
l...|
| 102582914| 4900319|        P| 149567.02| 1995-06-01|      1-URGENT|Clerk#000030552|      0|unts. slyly iron
i...|
| 102582915| 2156444|        O| 236434.73| 1997-02-24|      3-MEDIUM|Clerk#000026208|      0| accounts. blith
e...|
| 102582916| 3338122|        F| 105807.92| 1993-03-24|      1-URGENT|Clerk#000018865|      0|arefully final p
i...|
| 102582917| 3811882|        O| 125658.05| 1996-06-15|      3-MEDIUM|Clerk#000005898|      0|thely pending dep
osi|
+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+
only showing top 5 rows
```

```
[13]: spark.table("iceberg_catalog.db.part").show(5)
+-----+-----+-----+-----+-----+-----+-----+-----+
|P_PARTKEY|          P_NAME|      P_MFGR| P_BRAND|          P_TYPE|P_SIZE|P_CONTAINER|P_RETAILPRICE|      P_COM
MENT|
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1|goldenrod lavende...|Manufacturer#1|Brand#13|PROMO BURNISHED C...|    7| JUMBO PKG|      901.0|      ly. slyly i
roni|
| 2|blush thistle blu...|Manufacturer#1|Brand#13| LARGE BRUSHED BRASS|    1| LG CASE|      902.0|      lar accounts
amo|
| 3|spring green yell...|Manufacturer#4|Brand#42|STANDARD POLISHED...|   21| WRAP CASE|      903.0|regular deposits
hag|
| 4|cornflower chocol...|Manufacturer#3|Brand#34| SMALL PLATED BRASS|   14| MED DRUM|      904.0|      p furious
ly r|
| 5|forest brown cora...|Manufacturer#3|Brand#32|STANDARD POLISHED...|   15| SM PKG|      905.0|      wake carefu
lly |
+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+
only showing top 5 rows
```

```
[14]: spark.table("iceberg_catalog.db.partsupp").show(5)
+-----+-----+-----+-----+-----+
|PS_PARTKEY|PS_SUPPKEY|PS_AVAILQTY|PS_SUPPLYCOST|          PS_COMMENT|
+-----+-----+-----+-----+-----+
| 8524289|     24290|     6913|    476.06|al pinto beans wa...
| 8524289|    149307|     7951|    259.61|un blithely acros...
| 8524289|    274324|     4259|    801.46|ly express reque...
| 8524289|    399341|     6770|    705.46|he blithely ironi...
| 8524290|    24291|      404|    654.73|ual dinos. blithe...
+-----+-----+-----+-----+-----+
only showing top 5 rows
```

```
[15]: spark.table("iceberg_catalog.db.region").show(5)
+-----+-----+-----+
|R_REGIONKEY| R_NAME| R_COMMENT|
+-----+-----+-----+
| 0| AFRICA|lar deposits. bli...
| 1| AMERICA|hs use ironic, ev...
| 2| ASIA|ges. thinly even ...
| 3| EUROPE|ly final courts c...
| 4|MIDDLE EAST|uickly special ac...
+-----+-----+-----+
only showing top 5 rows
```



```
[16]: spark.table("iceberg_catalog.db.supplier").show(5)
+-----+-----+-----+-----+-----+-----+
|S_SUPPKEY| S_NAME| S_ADDRESS|S_NATIONKEY| S_PHONE|S_ACCTBAL| S_COMMENT|
+-----+-----+-----+-----+-----+-----+
| 1|Supplier#00000001| N KD4on90M Ipw3,...| 17|27-918-335-1736| 5755.94|each styl... above ...
| 2|Supplier#00000002|89eJ5ksX3ImxJQBvx...| 5|15-679-861-2259| 4032.68| slyly bold instr...
| 3|Supplier#00000003|q1,63Pj60jiuUYFUo...| 1|11-383-516-1199| 4192.4|blithely silent r...
| 4|Supplier#00000004|Bk7ah4CK8SYOTepEm...| 15|25-843-787-7479| 4641.08|riously even requ...
| 5|Supplier#00000005| Gcdm2rJRzl5qlTVzc| 11|21-151-690-3663| -283.84|. slyly regular p...
+-----+-----+-----+-----+-----+-----+
only showing top 5 rows
```

Gambar 1.9 Hasil query pada tabel Iceberg yang telah dibuat.

Langkah selanjutnya adalah memilih *query engine* yang akan digunakan untuk menjalankan *query* pada *data lakehouse* serta membuat *data mart*. Terdapat dua kandidat teknologi yang dapat digunakan: Spark dan Snowflake. Namun, kami memilih menggunakan Snowflake dengan alasan sebagai berikut.

- Performa yang lebih baik dari Spark. Berikut adalah salah satu *benchmark* performa Snowflake dan Spark pada *dataset TPC-H* pada beberapa operasi (<https://www.fosfor.com/wp-content/uploads/2024/02/Fosfor-Spectra-Whitepaper-Benchmarking-Snowflake-vs-Spark-for-Optimized-DataOps.pdf>)
- Snowflake memiliki tampilan yang cukup mudah digunakan, setiap operasi dapat dilakukan hanya dengan SQL, serta mudah untuk berbagi *worksheet*.
- Penasaran cara kerja Snowflake (*mumpung ada free trial* 😊).

Berikut adalah *setup* yang diperlukan untuk mengintegrasikan tabel Iceberg yang tersimpan di S3 dengan Snowflake, sesuai dokumentasi Snowflake

(https://quickstarts.snowflake.com/guide/data_lake_using_apache_iceberg_with_snowflake_and_aws_glue/index.html?index=..%2F..index#0).

- Menyiapkan role serta data warehouse yang diperlukan (setup.sql)

```
HOL_ICE_DB.PUBLIC ▾ Settings ▾
```

Open in Workspaces 

```
1 USE ROLE SECURITYADMIN;
2
3 CREATE OR REPLACE ROLE HOL_ICE_RL COMMENT='Iceberg Role';
4 GRANT ROLE HOL_ICE_RL TO ROLE SYSADMIN;
5
6
7
8 USE ROLE ACCOUNTADMIN;
9
10 GRANT CREATE INTEGRATION ON ACCOUNT TO ROLE HOL_ICE_RL;
11 GRANT CREATE EXTERNAL VOLUME ON ACCOUNT TO ROLE HOL_ICE_RL;
12 GRANT CREATE DATABASE ON ACCOUNT TO ROLE HOL_ICE_RL;
13 GRANT CREATE WAREHOUSE ON ACCOUNT TO ROLE HOL_ICE_RL;
14
15
16
17 USE ROLE HOL_ICE_RL;
18
19 CREATE OR REPLACE DATABASE HOL_ICE_DB;
20
21 CREATE OR REPLACE WAREHOUSE HOL_ICE_WH
22   WITH WAREHOUSE_SIZE = 'XSML'
23     INITIALLY_SUSPENDED = TRUE;
```

Gambar 1.10 Menyiapkan *role* serta *data warehouse* di Snowflake

- Menyiapkan *external volume* untuk terhubung ke S3 (workflow.sql)

```
HOL_ICE_DB.PUBLIC ▾ Settings ▾
```

Open in Workspaces Code Versions

1 USE ROLE HOL_ICE_RL;
2
3 USE HOL_ICE_DB.PUBLIC;
4
5 USE WAREHOUSE HOL_ICE_WH;
6
7
8 CREATE OR REPLACE EXTERNAL VOLUME HOL_ICE_EXT_VOL
9 STORAGE_LOCATIONS =
10 (
11 (
12 NAME = 'if4044-big-data-kel-4'
13 STORAGE_PROVIDER = 'S3'
14 STORAGE_BASE_URL = 's3://if4044-big-data-kel-4/iceberg/'
15 STORAGE_AWS_ROLE_ARN = 'arn:aws:iam::591445344625:role/GlueSnowflakeLabRole'
16)
17)
18);
19
20 DESC EXTERNAL VOLUME HOL_ICE_EXT_VOL;

Gambar 1.11 Menyiapkan *external volume* di Snowflake

- Menyiapkan *catalog integration* untuk terhubung ke Glue Catalog

```
23  CREATE or REPLACE CATALOG INTEGRATION HOL_ICE_GLUE_CAT_INT
24    CATALOG_SOURCE=GLUE
25    CATALOG_NAMESPACE='db'
26    TABLE_FORMAT=ICEBERG
27    GLUE_AWS_ROLE_ARN='arn:aws:iam::591445344625:role/GLueSnowflakeLabRole'
28    GLUE_CATALOG_ID='591445344625'
29    ENABLED=TRUE;
30
31
32 DESC CATALOG INTEGRATION HOL_ICE_GLUE_CAT_INT;
```

Gambar 1.12 Menyiapkan catalog integration di Snowflake

- Membuat tabel Iceberg

```
34  CREATE OR REPLACE ICEBERG TABLE nation
35    EXTERNAL_VOLUME='HOL_ICE_EXT_VOL'
36    CATALOG='HOL_ICE_GLUE_CAT_INT'
37    CATALOG_TABLE_NAME='nation';
38
39  CREATE OR REPLACE ICEBERG TABLE customer
40    EXTERNAL_VOLUME='HOL_ICE_EXT_VOL'
41    CATALOG='HOL_ICE_GLUE_CAT_INT'
42    CATALOG_TABLE_NAME='customer';
43
44  CREATE OR REPLACE ICEBERG TABLE part
45    EXTERNAL_VOLUME='HOL_ICE_EXT_VOL'
46    CATALOG='HOL_ICE_GLUE_CAT_INT'
47    CATALOG_TABLE_NAME='part';
48
49  CREATE OR REPLACE ICEBERG TABLE partsupp
50    EXTERNAL_VOLUME='HOL_ICE_EXT_VOL'
51    CATALOG='HOL_ICE_GLUE_CAT_INT'
52    CATALOG_TABLE_NAME='partsupp';
53
54  CREATE OR REPLACE ICEBERG TABLE region
55    EXTERNAL_VOLUME='HOL_ICE_EXT_VOL'
56    CATALOG='HOL_ICE_GLUE_CAT_INT'
57    CATALOG_TABLE_NAME='region';
58
59  CREATE OR REPLACE ICEBERG TABLE supplier
60    EXTERNAL_VOLUME='HOL_ICE_EXT_VOL'
61    CATALOG='HOL_ICE_GLUE_CAT_INT'
62    CATALOG_TABLE_NAME='supplier';
63
64  CREATE OR REPLACE ICEBERG TABLE lineitem
65    EXTERNAL_VOLUME='HOL_ICE_EXT_VOL'
66    CATALOG='HOL_ICE_GLUE_CAT_INT'
67    CATALOG_TABLE_NAME='lineitem';
68
69  CREATE OR REPLACE ICEBERG TABLE orders
70    EXTERNAL_VOLUME='HOL_ICE_EXT_VOL'
71    CATALOG='HOL_ICE_GLUE_CAT_INT'
72    CATALOG_TABLE_NAME='orders';
```

Gambar 1.13 Membuat tabel Iceberg di Snowflake

Selanjutnya, kami menjalankan *query* pada sebagai berikut.

```

74    SELECT * FROM nation LIMIT 5;
75    SELECT * FROM customer LIMIT 5;
76    SELECT * FROM part LIMIT 5;
77    SELECT * FROM partsupp LIMIT 5;
78    SELECT * FROM region LIMIT 5;
79    SELECT * FROM supplier LIMIT 5;
80    SELECT * FROM lineitem LIMIT 5;

```

Gambar 1.14 *Query* pada tabel Iceberg di Snowflake

# N_NATIONKEY	▲ N_NAME	# N_REGIONKEY	▲ N_COMMENT
1	0 ALGERIA	0	haggle. carefully final deposits detect slyly agai
2	1 ARGENTINA	1	al foxes promise slyly according to the regular accounts. bc
3	2 BRAZIL	1	y alongside of the pending deposits. carefully special pack
4	3 CANADA	1	eas hang ironic, silent packages. slyly regular packages are
5	4 EGYPT	4	y above the carefully unusual theodolites. final dugouts are

# C_CUSTKEY	▲ C_NAME	▲ C_ADDRESS	# C_NATIONKEY	▲ C_PHONE	# C_ACCTBAL	▲ C_MKTS
1 6000001	Customer#006000001	Mjb5hHVQsXoW7jdle	4	14-452-629-2920	3519.52	BUILDING
2 6000002	Customer#006000002	DOXTGPwmDTAD2LHLxG	6	16-106-789-8446	3561.38	HOUSEHO
3 6000003	Customer#006000003	dCCk6jd JgjksBPUJmrzK'	24	34-190-343-8143	3171.07	MACHINER
4 6000004	Customer#006000004	y4sTPWAEjhS5yf,GEJ184/	10	20-811-886-8293	-854.19	AUTOMOB
5 6000005	Customer#006000005	zwQg1D6PjHH6Qn7mIGV2	24	34-744-799-1324	7532.52	HOUSEHO

# P_PARTKEY	▲ P_NAME	▲ P_MFGR	▲ P_BRAND	▲ P_TYPE	# P_SIZE	▲ P_CONTAINER
1 6000001	smoke chartreuse slate lavender lemon	Manufacturer#5	Brand#55	LARGE BRUSHED COPPER	20	J
2 6000002	indian grey gainsboro metallic papaya	Manufacturer#5	Brand#52	STANDARD POLISHED STEEL	48	L
3 6000003	grey lemon bisque spring seashell	Manufacturer#1	Brand#11	PROMO BURNISHED NICKEL	20	V
4 6000004	indian papaya yellow firebrick sky	Manufacturer#1	Brand#14	LARGE PLATED COPPER	37	L
5 6000005	hot lavender burnished cornflower lace	Manufacturer#1	Brand#14	MEDIUM POLISHED BRASS	37	S

# PS_PARTKEY	# PS_SUPPKEY	# PS_AVAILQTY	# PS_SUPPLYCOST	▲ PS_COMMENT
1 4000001	2	119	664.78	ully even theodolites. blithely even deposits cajole furiously. i
2 4000001	125010	8858	319.14	hins boost regular deposits. fluffily even pinto beans sleep ne
3 4000001	250018	7082	363.89	sentiments detect slyly at the fluffy regular ideas. special th
4 4000001	375026	7503	183.57	ronic requests use after the furiously regular packages. fur
5 4000002	3	1986	650.67	ously express platelets. bravely regular courts are s

# R_REGIONKEY	A R_NAME	A R_COMMENT
1	0 AFRICA	lar deposits. blithely final packages cajole. regular waters are final requests. re
2	1 AMERICA	hs use ironic, even requests. s
3	2 ASIA	ges. thinly even pinto beans ca
4	3 EUROPE	ly final courts cajole furiously final excuse
5	4 MIDDLE EAST	uickly special accounts cajole carefully blithely close requests. carefully final a

# S_SUPPKEY	A S_NAME	A S_ADDRESS	# S_NATIONKEY	A S_PHONE	# S_ACCTBAL	A S_COMMENT
1 450001	Supplier#000450001	zIvdPRBK24VO	11	21-973-108-8645	3531.07	ing warthogs. bli
2 450002	Supplier#000450002	fYjM0rHOKIMfILYh8v	8	18-384-609-9108	-390.92	eas. special asym
3 450003	Supplier#000450003	kx7ixT8VPsGulfaDXc	17	27-573-847-2814	5665.61	blithely regular in
4 450004	Supplier#000450004	LtVsOdboYIWtAnh,dI	24	34-132-109-2713	3997.58	hy, ironic account
5 450005	Supplier#000450005	swFkeRKDBkTPFmJ	8	18-380-738-1245	8264.64	sly even package

# L_ORDERKEY	# L_PARTKEY	# L_SUPPKEY	# L_LINENUMBER	# L_QUANTITY	# L_EXTENDEDPRICE	# L_DISCOUNT
1 1	7759468	384484	1	17	25960.36	0.04
2 1	3365454	365455	2	36	54694.44	0.09
3 1	3184989	184990	3	8	16590.64	0.1
4 1	106575	231576	4	28	44283.96	0.09
5 1	1201332	76339	5	24	29598.48	0.1

Gambar 1.15 Hasil query pada tabel Iceberg di Snowflake

3) Memproses data ke *Data Mart*

Data mart merupakan subset dari keseluruhan *data lake/data warehouse* yang memiliki kegunaan khusus, misalnya *data mart* khusus untuk bagian *sales* atau *human resource*. Pembuatan *data mart* bisa dilakukan dengan berbagai cara. Namun, pada tugas ini, kami menggunakan *data build tool* (disingkat sebagai dbt) sebagai *orchestrator* data.

Proses instalasi dbt dapat dilakukan dengan perintah:

```
pip install dbt-snowflake
```

Perintah di atas akan mengunduh dan menginstal pustaka dan modul yang diperlukan, baik *core* maupun konektor khusus *snowflake* (karena data disimpan di *snowflake*). Kemudian jalankan perintah:

```
dbt init
```

Pada terminal, dbt akan meminta sejumlah informasi yang diperlukan, mulai dari nama proyek, jenis basis data, detail kredensial Snowflake, dan detail data (nama skema, nama *warehouse*, dsb). Lalu, buka *folder* dbt dengan nama proyek yang dipilih dan mulailah membuat model dengan menulis *query SQL* di *folder models*.

Kami membuat sejumlah model: empat model terkait *sales* dan satu model berupa *query TPC-H ke-3*. *Query* tersebut dapat dilihat pada cuplikan kode yang tertera di bawah ini.

```
SELECT
    l.l_orderkey,
    SUM(l.l_extendedprice * (1 - l.l_discount)) AS revenue,
    o.o_orderdate,
    o.o_shippriority
FROM
    {{ source('tpch', 'customer') }} AS c,
    {{ source('tpch', 'orders') }} AS o,
    {{ source('tpch', 'lineitem') }} AS l
WHERE
    c.c_mktsegment = 'BUILDING' -- RANDOMLY SELECTED WITHIN THE
    LIST OF VALUES DEFINED FOR SEGMENTS IN CLAUSE 4.2.2.13
    AND c.c_custkey = o.o_custkey
    AND l.l_orderkey = o.o_orderkey
    AND o.o_orderdate < DATE '1995-03-15' -- RANDOMLY SELECTED
    DAY WITHIN [1995-03-01 .. 1995-03-31]
    AND l.l_shipdate > DATE '1995-03-15' -- RANDOMLY SELECTED
    DAY WITHIN [1995-03-01 .. 1995-03-31]
GROUP BY
    l.l_orderkey,
    o.o_orderdate,
    o.o_shippriority
ORDER BY
    revenue DESC,
    o.o_orderdate
```

Model tersebut juga dicatat pada berkas `schema.yml` sebagai dokumentasi dan memungkinkan penggunaan `test`. Kami melakukan tes terhadap kolom `l_orderkey` dan kolom `revenue` berupa `not null` (lihat bagian berwarna kuning pada cuplikan kode di bawah).

```
models:
  - name: q03_revenue
    description: >
      TPC-H Q3: Revenue by customer for orders placed before
      1995-03-15 and shipped after
    columns:
      - name: l_orderkey
        tests:
          - not_null
      - name: revenue
        tests:
          - not_null
    - name: stg_sales_data
      description: |
        Cleaned, single-source-of-truth sales fact.
        Joins TPC-H tables: lineitem, orders, customer, nation,
        region, part.
      config:
        materialized: table
      columns:
        - name: order_key
          description: "Surrogate key representing the TPC-H
            lineitem order key"
        - name: part_key
          description: "Foreign key to the part table"
        - name: line_revenue
          description: "Revenue for each line calculated as
            extendedprice * (1 - discount)"
        - name: order_date
          description: "Date on which the order was placed"
        - name: region
          description: "Region name derived from customer → nation
            → region"
        - name: product_name
          description: "Name of the product (part)"
```

Selain model tersebut, kami juga membuat empat model lainnya, yakni `stg_sales_data`, `monthly_growth`, `regional_revenue`, dan `top_products_by_region`. Query SQL lengkap untuk masing-masing model dapat dilihat pada *repository GitHub* tugas ini. Pembahasan lanjutan di subbab ini hanya akan membahas *notable aspect* dari masing-masing model.

Dalam pembuatan model, kami menginginkan adanya dependensi antarmodel sehingga dokumentasi dbt dapat menampilkan contoh DAG yang menggambarkan dependensi antarmodel. Keseluruhan model bergantung terhadap model utama, yakni `stg_sales_data`. Contoh dependensi terhadap model tersebut dapat dilihat pada cuplikan kode CTE berikut (lihat bagian yang ditandai dengan warna kuning).

```
WITH base AS (
    SELECT
        region,
        {{ month_trunc("order_date") }} AS order_month,
        SUM(line_revenue) AS revenue
    FROM {{ ref('stg_sales_data') }}
    GROUP BY 1, 2
),
```

Selain dependensi, kami juga membuat sejumlah *macro* untuk membuat kode semakin *reusable*. Perhatikan bagian berwarna hijau pada cuplikan kode di atas. Kode tersebut akan melakukan *truncation* di level bulan dari suatu kolom. Kode *macro*-nya tertera pada *folder macros* atau dapat dilihat pada cuplikan kode berikut.

```
{% macro month_trunc(date_col) %}
    DATE_TRUNC('month', {{ date_col }})
{% endmacro %}
```

Dua *macro* lainnya berupa *macro* untuk menghitung pendapatan yang

digunakan pada model `stg_sales_data` dan `macro` untuk menentukan *top n* suatu grup. Kode `macro top n` tadi cukup menarik karena menggunakan parameterisasi *jinja* untuk menentukan berapa banyak *n* yang diambil.

```
{% macro top_n_per_group(table, group_cols, order_col,
n_var='top_n') %}

{#- read `top_n` with default=1 -#}
{% set n_value = var(n_var, 1) %}

WITH ranked AS (
    SELECT
        *,
        ROW_NUMBER() OVER (
            PARTITION BY {{ group_cols | join(', ') }}
            ORDER BY {{ order_col }})
        ) AS _row_num
    FROM {{ table }}
)
SELECT *
FROM ranked
WHERE _row_num ≤ {{ n_value }}
{% endmacro %}
```

Nilai *top_n* dapat diatur dengan memasukan nilai ke parameter saat pemanggilan *macro* atau melalui parameter di berkas `dbt_project.yml`. Sebagai *fallback*, jika pengguna tidak memasukan parameter di kedua tempat tadi, maka akan digunakan nilai 1.

Berikut pada Gambar 1.16 adalah tangkapan layar dari terminal saat kami menjalankan perintah `dbt test` untuk menguji data.

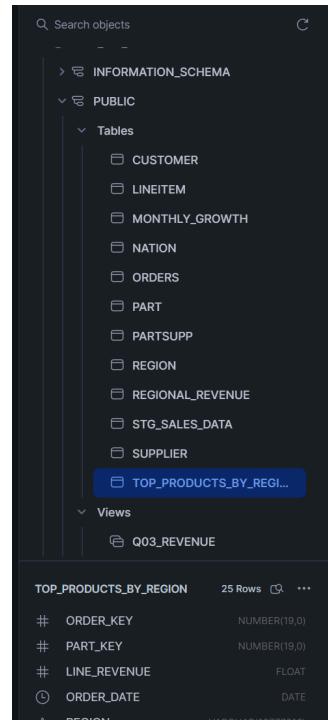
```
PS C:\Users\rayhalif4044-data-lakehouse\if4044_tpch> dbt test
10:47:54  Running with dbt=1.9.6
C:\Users\rayha\AppData\Local\Programs\Python\Python310\lib\site-packages\snowflake\connector\options.py:104: UserWarning: You have an incompatible version of 'pyarrow' installed (19.0.1), please install a version that adheres to: 'pyarrow<19.0.0; extra == "pandas"'
    warn_incompatible_dep()
10:47:57 Registered adapter: snowflake=1.9.4
10:47:58 Unable to do partial parsing because saved manifest not found. Starting full parse.
10:48:03 Found 5 models, 2 data tests, 8 sources, 479 macros
10:48:03
10:48:03 Concurrency: 1 threads (target='dev')
10:48:03
10:48:06 1 of 2 START test not_null_q03_revenue_1_orderkey ..... [RUN]
10:48:09 1 of 2 PASS not_null_q03_revenue_1_orderkey ..... [PASS] in 2.36s
10:48:09 2 of 2 START test not_null_q03_revenue_revenue ..... [RUN]
10:48:17 2 of 2 PASS not_null_q03_revenue_revenue ..... [PASS] in 8.17s
10:48:18
10:48:18 Finished running 2 data tests in 0 hours 0 minutes and 14.35 seconds (14.35s).
10:48:18
10:48:18 Completed successfully
10:48:18
10:48:18 Done. PASS=2 WARN=0 ERROR=0 SKIP=0 TOTAL=2
PS C:\Users\rayha\if4044-data-lakehouse\if4044_tpch>
```

Gambar 1.16 Hasil dbt test pada Data

Setelah tes berhasil dijalankan dengan *flag* PASS, maka selanjutnya kami buat masing-masing model dengan perintah *dbt run*. Keseluruhan model sukses dibuat oleh dbt sebagaimana tertera pada Gambar 1.17. Kami juga melakukan pengecekan pada Snowflake untuk memastikan bahwa tabel dan *view* benar-benar terbuat dan datanya ada. Hasilnya dapat dilihat pada Gambar 1.18 dan Gambar 1.19.

```
104: UserWarning: You have an incompatible version of 'pyarrow' installed (19.0.1), please install a version that adheres to: 'pyarrow<19.0.0; extra == "pandas"'
    warn_incompatible_dep()
10:51:02 Registered adapter: snowflake=1.9.4
10:51:05 Found 5 models, 2 data tests, 8 sources, 479 macros
10:51:05
10:51:05 Concurrency: 1 threads (target='dev')
10:51:05
10:51:08 1 of 5 START sql view model PUBLIC.q03_revenue ..... [RUN]
10:51:10 1 of 5 OK created sql view model PUBLIC.q03_revenue ..... [SUCCESS 1 in 1.70s]
10:51:10 2 of 5 START sql table model PUBLIC.stg_sales_data ..... [RUN]
10:52:43 2 of 5 OK created sql table model PUBLIC.stg_sales_data ..... [SUCCESS 1 in 92.77s]
10:52:43 3 of 5 START sql table model PUBLIC.monthly_growth ..... [RUN]
10:52:48 3 of 5 OK created sql table model PUBLIC.monthly_growth ..... [SUCCESS 1 in 5.21s]
10:52:48 4 of 5 START sql table model PUBLIC.regional_revenue ..... [RUN]
10:52:50 4 of 5 OK created sql table model PUBLIC.regional_revenue ..... [SUCCESS 1 in 1.46s]
10:52:50 5 of 5 START sql table model PUBLIC.top_products_by_region ..... [RUN]
10:54:17 5 of 5 OK created sql table model PUBLIC.top_products_by_region ..... [SUCCESS 1 in 87.37s]
10:54:19
10:54:19 Finished running 4 table models, 1 view model in 0 hours 3 minutes and 14.30 seconds (194.30s).
10:54:19
10:54:19 Completed successfully
10:54:19
10:54:19 Done. PASS=5 WARN=0 ERROR=0 SKIP=0 TOTAL=5
PS C:\Users\rayha\if4044-data-lakehouse\if4044_tpch>
```

Gambar 1.17 Hasil dbt run



Gambar 1.18 Daftar Tabel dan View pada Snowflake setelah dbt run

The screenshot shows the results of a query: "SELECT * FROM REGIONAL_REVENUE;". The results table has two columns: REGION and TOTAL_REVENUE. The data is as follows:

REGION	TOTAL_REVENUE
AFRICA	1928735490317.5
MIDDLE EAST	1930539682268.13
ASIA	1926308023551.99
EUROPE	1928810763537.38
AMERICA	1928460555384.12

On the right side of the interface, there are "Query Details" sections for Query duration (243ms), Rows (5), and Query ID (01bccd71-3204-7eb-0...). Below the table, there are two charts: a bar chart for REGION and a histogram for TOTAL_REVENUE.

Gambar 1.19 Contoh Isi Tabel Regional_Revenue

Setelah membuat model, kami juga menggunakan fitur dokumentasi dari dbt. Dokumentasi tersebut membuat antarmuka berisi deskripsi, keterangan, dan lain sebagainya seperti pada Gambar 1.20. Dependensi atau ketergantungan antarmodel juga dapat dilihat pada Gambar 1.21.

The screenshot shows the dbt interface with the following details:

- Sources:** tpch (customer, lineitem, nation, orders, partsupp, part, region, supplier).
- Projects:** if4044_tpch (macros, calculate_revenue, month_trunc, top_n_per_group, models, marts, sales, monthly_growth, regional_revenue, top_products_by_region, q03_revenue, staging, stg_sales_data).
- Table Documentation:**
 - monthly_growth table**
 - Details:** OWNER: ACCOUNTADMIN, TYPE: table, PACKAGE: if4044_tpch, LANGUAGE: sql, RELATION: HOL_ICE_DB.PUBLIC.monthly_growth, ACCESS: protected, VERSION: Not Enforced.
 - Description:** Month-over-month revenue growth per region.
 - Columns:**

COLUMN	TYPE	DESCRIPTION	CONSTRAINTS	DATA TESTS	MORE?
region	TEXT	Region name			>
order_month	DATE	First day of the month (bucketed via mont...			>
revenue	FLOAT	Total revenue for that region-month			>
prev_revenue	FLOAT	Revenue of the previous month for the sa...			>
growth_rate	FLOAT	(revenue - prev_revenue) / prev_revenue * 100			>

Gambar 1.20 Dokumentasi dari dbt



Gambar 1.21 Graf Dependensi Antarmodel dari dbt

2. Other Attempt

Selain alur penggeraan di atas, ada beberapa hal lain yang sempat dicoba.

1) Data Generation with DuckDB

Saat mencoba *repository* yang disarankan pada dokumen panduan tugas, kami sempat kesulitan menggunakan programnya. Maka, kami mencari cara lain untuk melakukan pembentukan *dataset* TPC-H. Salah satu alternatif yang menjanjikan adalah dengan menggunakan *core extension* TPC-H dari DuckDB. Berdasarkan keterangan pada dokumentasinya, seharusnya prosesnya luar biasa mudah, hanya memerlukan 3 baris kode saja.

Namun, saat dicoba di laptop, ternyata prosesnya relatif berat karena bersifat *memory-bound*. Sebagai contoh, pada panduannya, untuk melakukan pembentukan *dataset* TPC-H dengan *scale factor* 100, maka membutuhkan memori sebesar 71 GB. Apabila kami menggunakan *scale factor* 50 (menghasilkan data sebesar 50 GB), maka anggap saja kebutuhan memorinya adalah setengahnya, yakni 35,5 GB. Memori sebesar itu kurang lazim ditemukan pada laptop konsumen, setidaknya pada laptop kami. DuckDB sendiri sebenarnya memberikan opsi untuk melakukan pembentukan *dataset* dalam beberapa partisi, tetapi kami merasa prosesnya akan lebih lama dan tetap menyita memori laptop kami juga. Selain itu, ada juga isu *storage* (maaf laptop *issue*, SSD kami penuh). Terakhir, setelah kami pikir-pikir, mengunggah 50 GB data ke S3 akan cukup lama, setidaknya jika dengan koneksi internet kos ataupun Eduroam ITB.

Akhirnya, kami memutuskan untuk menyewa *instance VM* dari RunPod saja karena koneksi internetnya kencang, *storage*-nya bisa diatur, dan RAM-nya bisa mencapai 256 GB. Harganya juga relatif masuk akal. Singkat cerita, kami berhasil melakukan pembentukan *dataset* TPC-H dengan perintah berikut.

```
duckdb tpc-h.db  
INSTALL tpch;  
LOAD tpch;  
CALL dbgen(sf = 50);
```

Setelahnya, kami melakukan konversi dengan *export* ke format CSV dan Parquet dengan perintah berikut.

```
COPY <nama_tabel> TO '<nama_tabel>.csv' (HEADER, DELIMITER ',');  
COPY <nama_tabel> TO <nama_tabel>.parquet' (FORMAT parquet);
```

Kemudian, berkas kami unggah ke *object storage* dengan menggunakan bantuan program *rclone*. Detail lebih lanjut mengenai proses pembentukan dataset dan metode unggahnya ke *object storage* dapat dilihat di *repository GitHub* yang tertera.

2) Set-up Trino x Nessie x Iceberg

Trino merupakan salah satu *query engine* yang populer digunakan. Oleh karena itu, kami tertarik untuk mencoba Trino. Kami berencana untuk menggunakan Trino dan Iceberg karena keduanya gratis. Sebenarnya, kami sudah mencoba set-up Trino dengan menggunakan Hive metastore. Namun, karena sangat ribet, akhirnya kami belok menggunakan metastore lain, yakni Nessie. Mengapa Nessie? Karena dari pilihan yang ada di Gambar 2.1, kami memutuskan:

- Hive metastore → sudah dicoba, ribet → menyerah
- AWS Glue catalog → sudah dicoba, tapi dengan Snowflake (alur program utama)
- JDBC dan REST catalog → tidak menarik
- Snowflake catalog → sepertinya lebih baik kalau sekalian pakai Snowflake daripada pakai Iceberg

- **Nessie** → akhirnya pilih Nessie dari hasil eliminasi dan karena Nessie tergolong teknologi baru. Yang menarik dari Nessie adalah ia berperan juga sebagai “git for data”.

Requirements

To use Iceberg, you need:

- Network access from the Trino coordinator and workers to the distributed object storage.
- Access to a [Hive metastore service \(HMS\)](#), an [AWS Glue catalog](#), a [JDBC catalog](#), a [REST catalog](#), a [Nessie server](#), or a [Snowflake catalog](#).
- Data files stored in the file formats [Parquet](#)(default), [ORC](#), or Avro on a [supported file system](#).

Gambar 2.1 Requirements Iceberg connector melalui Trino

OK sudah *decide* mau pakai Trino x Nessie x Iceberg. Selanjutnya set-up Trino. Trino 475 sudah ada Iceberg di dalamnya, sehingga lebih mudah untuk set-up Icebergnya, hanya tinggal atur connector-nya saja. Menurut [Iceberg Connector](#) jika ingin menggunakan Nessie, set `iceberg.catalog.type` ke “nessie” di file `<catalog>.properties`. Namun, halaman tersebut tidak menjelaskan lebih lanjut bagaimana pengaturan konfigurasi connector dengan Nessie. Baru di awal tahapan tapi Trino sudah menyebalkan 😞🙏.

Lalu kami juga harus set-up konfigurasi berdasarkan jenis *file system* yang digunakan. Kami juga punya Parquet-nya di Cloudflare R2 (mengapa Cloudflare R2? Karena Cloudflare R2 zero egrees fee!), jadi Trino yang ini di-connect ke Cloudflare R2. Panduannya ada di sini [File System S3](#). Pada tahap ini, setting-up Nessie x Trino masih misterius. Di mana sebenarnya set-up Nessie catalog? Ternyata ada di halaman lain... Gambar 2.2 memperlihatkan bab yang harus diakses untuk konfigurasi catalog di Trino. Baru catalognya saja (cries).

Trino 475 Documentation

Overview
Installation
Clients
Security
Administration
Query optimizer
Connectors
Object storage
Azure Storage file system support
Google Cloud Storage file system support
S3 file system support
Local file system support
HDFS file system support
File system cache
Alluxio file system support
Metastores
Object storage file formats
Functions and operators
User-defined functions
SQL language
SQL statement syntax
Developer guide
Glossary
Appendix
Release notes

Gambar 2.2 Konfigurasi catalog pada Trino 475

Trino sudah punya Docker image default, jadi kami pikir sudah selesai set-up-nya. *Long story short* konfigurasi minimal Trino adalah config.properties, jvm.config, log.properties, dan node.properties juga. Lalu kami *mount volume* ke /etc/node.properties, /etc/jvm.config, dst. karena melihat [Deploying Trino](#) yang hasil cuplikannya dapat dilihat pada Gambar 2.3.

Node properties

The node properties file, `/etc/node.properties`, contains configuration specific to each node. A *node* is a single installed instance of Trino on a machine. This file is typically created by the deployment system when Trino is first installed. The following is a minimal `/etc/node.properties`:

```
node.environment=production  
node.id=ffffffff-ffff-ffff-ffff-ffffffffffff  
node.data-dir=/var/trino/data
```

Gambar 2.3 Konfigurasi dasar Trino 475

Namun, malah error/tidak ter-mount. Ternyata, harusnya di-mount ke `/etc/trino` bukan `/etc`. Ini ada di [Container Configuration](#) yang cuplikannya bisa dilihat di Gambar 2.4.

Configuring Trino

The image already contains a default configuration to get started, and some catalogs to allow you to explore Trino. You can also use the container with your custom configuration files in a local `etc` directory structure as created in the [Deploying Trino](#). If you mount this directory as a volume in the path `/etc/trino` when starting the container, your configuration is used instead of the default in the image.

```
$ docker run --name trino -d -p 8080:8080 --volume $PWD/etc:/etc/trino trinodb/trino
```

To keep the default configuration and only configure catalogs, mount a folder at `/etc/trino/catalog`, or individual catalog property files in it.

If you want to use additional plugins, mount them at `/usr/lib/trino/plugin`.

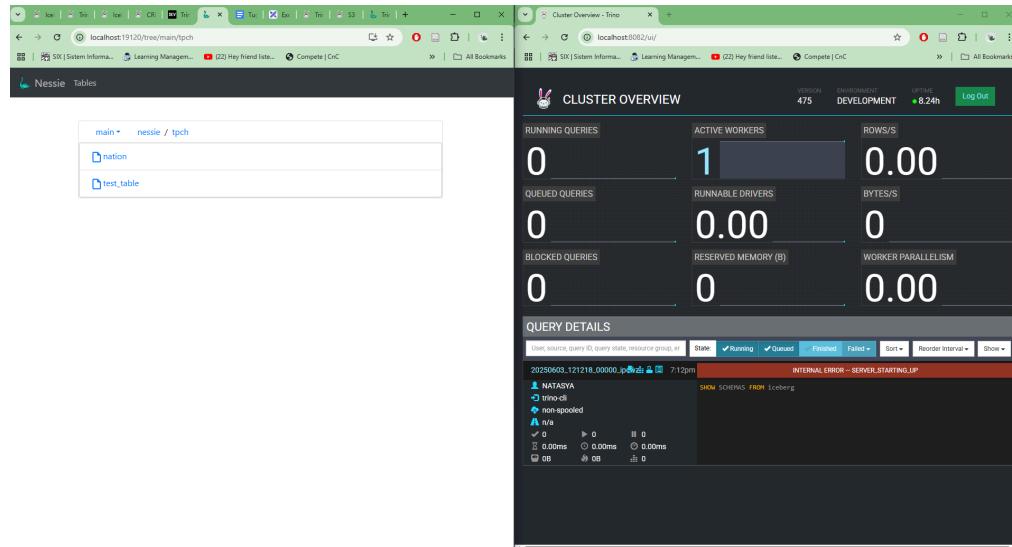
To avoid having to create catalog files and mount them in the container, you can enable dynamic catalog management by setting the `CATALOG_MANAGEMENT` environmental variable to `dynamic`.

```
$ docker run -name trino -d -p 8080:8080 -e CATALOG_MANAGEMENT=dynamic trinodb/trino
```

After connecting to Trino, execute [Catalog management](#) statements to create drop catalogs as desired. To make these changes persistent across container restarts, a volume must be mounted at `/etc/trino/catalog`.

Gambar 2.4 Konfigurasi Trino pada Container

Long story short setelah perang dengan Docker, akhirnya Trino UI dan Nessie UI dapat diakses, yeay! Gambar 2.5 memperlihatkan Trino UI dan Nessie UI.



Gambar 2.5 Trino UI dan Nessie UI

Permasalahan dimulai di sini karena datanya tidak *persistent*. Kami sudah membuat schema, tapi schemanya akan hilang setelah docker compose down. Ternyata, schema tidak akan tersimpan jika tidak ada tabel di dalamnya. Akhirnya kami bisa menyimpan tabel dan schema secara persistent ke Cloudflare R2, di-query menggunakan Trino, dan query + commit + metadata tabelnya tersimpan di Nessie.

Step selanjutnya, tinggal membuat tabel Iceberg menggunakan Trino. Di sini lah letak puncak masalahnya. **Ternyata, Trino tidak bisa membuat Iceberg table dari file Parquet 😞**. Nama sectionnya "Add files" jadi kami pikir Trino bisa **membuat table dari files**. Tidak semudah itu ferguso. Trino hanya bisa menambahkan files dari existing Hive table ke existing Iceberg table, bukan dari files seperti dilihat pada Gambar 2.6. Akhir kata, dokumentasi Trino sangat membingungkan dan menyebalkan.

Add files

The connector can add files from tables or locations to an existing Iceberg table if `iceberg.add-files-procedure.enabled` is set to `true` for the catalog.

Use the procedure `add_files_from_table` to add existing files from a Hive table in the current catalog, or `add_files` to add existing files from a specified location, to an existing Iceberg table.

The data files must be the Parquet, ORC, or Avro file format.

The procedure adds the files to the target table, specified after `ALTER TABLE`, and loads them from the source table specified with the required parameters `schema_name` and `table_name`. The source table must be accessible in the same catalog as the target table and use the Hive format. The target table must use the Iceberg format. The catalog must use the Iceberg connector.

Gambar 2.6 Add files with Trino

Pada akhirnya kami tidak melanjutkan Trino Nessie ini.... Kami menggunakan Trino x Nessie x Iceberg karena ingin menghindari konfigurasi Hive. Namun setelah sejauh ini malah disuruh konfigurasi Hive-Hive lagi 😊😊😊. Karena yang Snowflake sudah work, jadi kami pakai itu saja deh.

Jika dilihat dari sejarahnya, Trino ini hadir sepertinya dengan tujuan untuk menggantikan Hive, makanya seperti itu (hanya bisa migrasi dari Hive table).

3. Lesson Learned

Berikut adalah beberapa refleksi serta *lesson learned* yang didapat dari tugas ini.

1) Perbedaan ukuran *file CSV* dan *Parquet* untuk menyimpan *dataset*

TPC-H.

File CSV membutuhkan 49GB, sedangkan *file Parquet* membutuhkan 16GB (tiga kali lebih kecil dari CSV).

```
haziqam@haziqam-ubuntu:~/Education/IF4044 Big Data/if4044-data-lakehouse/tpc-h/tpc-h-50gb$ find . -type f -name "*.csv" -exec du -ch {} + | grep total$  
49G      total  
haziqam@haziqam-ubuntu:~/Education/IF4044 Big Data/if4044-data-lakehouse/tpc-h/tpc-h-50gb$ find . -type f -name "*.parquet" -exec du -ch {} + | grep total$  
16G      total
```

Gambar 3.1 Perbedaan ukuran *file* CSV dan Parquet

2) Perbandingan waktu yang diperlukan untuk menjalankan *query* pada CSV dan Parquet dengan DuckDB

Proses *benchmark* dilakukan dengan *instance* VM berjenis *Compute Optimized* dari [RunPod](#). Spesifikasi dari VM yang digunakan adalah 16 vCPU dengan *clock speed* 3.7 Ghz, 32 GB RAM DDR5, dan penyimpanan SSD NVMe sebesar 160 GB. Perbandingan menggunakan sejumlah *query* standar dari TPC-H, yakni Q1, Q2, Q3, Q5, dan Q12. Waktu eksekusi *query* dicatat dengan *dot command* berupa *timer*.

Data tidak di-*load* ke dalam DuckDB karena berpotensi memengaruhi hasil (data dikonversi ke format internal DuckDB). Oleh karena itu, kami menggunakan sintaks CREATE VIEW agar berkas selalu dibaca (berupa tabel eksternal). Perintah yang digunakan adalah sebagai berikut.

```
CREATE VIEW <nama_tabel> AS read_csv('<nama_tabel>.csv');  
CREATE VIEW <nama_tabel> AS  
read_parquet('<nama_tabel>.parquet');
```

Setelah membuat seluruh *view* untuk kedua jenis *file*, nyalakan *timer*, lalu jalankan masing-masing *query*. Hasil dari masing-masing *query* untuk masing-masing jenis *file* dapat dilihat pada tabel berikut.

Q1 TPC-h pada CSV files

```
D SELECT
    l_returnflag,
    l_linenstatus,
    SUM(l_quantity) AS sum_qty,
    SUM(l_extendedprice) AS sum_base_price,
    SUM(l_extendedprice * (1 - l_discount)) AS sum_disc_price,
    SUM(l_extendedprice * (1 - l_discount) * (1 + l_tax)) AS sum_charge,
    AVG(l_quantity) AS avg_qty,
    AVG(l_extendedprice) AS avg_price,
    AVG(l_discount) AS avg_disc,
    COUNT(*) AS count_order
FROM lineitem
WHERE l_shipdate <= DATE '1998-12-01' - INTERVAL '90' DAY
GROUP BY l_returnflag, l_linenstatus
ORDER BY l_returnflag, l_linenstatus;
```

100% [REDACTED]

l_returnflag	l_linenstatus	sum_qty	...	avg_price	avg_disc	count_order
varchar	varchar	double	...	double	double	int64
A	F	2265244648.0	...	38236.31971863085	0.05000093973518583	88833540
N	F	59131415.0	...	38247.10127294446	0.04997341965366381	2317878
N	O	4461409727.0	...	38236.618973927725	0.04999975874396146	174959429
R	F	2265353354.0	...	38239.38233294368	0.05000241499071525	88832641

4 rows 10 columns (6 shown)

Run Time (s): real 32.751 user 181.917723 sys 26.256410

Q1 TPC-h pada Parquet files

```
D SELECT
    l_returnflag,
    l_linenstatus,
    SUM(l_quantity) AS sum_qty,
    SUM(l_extendedprice) AS sum_base_price,
    SUM(l_extendedprice * (1 - l_discount)) AS sum_disc_price,
    SUM(l_extendedprice * (1 - l_discount) * (1 + l_tax)) AS sum_charge,
    AVG(l_quantity) AS avg_qty,
    AVG(l_extendedprice) AS avg_price,
    AVG(l_discount) AS avg_disc,
    COUNT(*) AS count_order
FROM lineitem
WHERE l_shipdate <= DATE '1998-12-01' - INTERVAL '90' DAY
GROUP BY l_returnflag, l_linenstatus
ORDER BY l_returnflag, l_linenstatus;
```

100% [REDACTED]

l_returnflag	l_linenstatus	sum_qty	...	avg_price	avg_disc	count_order
varchar	varchar	double	...	double	double	int64
A	F	2265244648.0	...	38236.31971863083	0.05000093973515981	88833540
N	F	59131415.0	...	38247.10127294449	0.0499734196536253	2317878
N	O	4461409727.0	...	38236.61897392762	0.049999758743992956	174959429
R	F	2265353354.0	...	38239.382332943576	0.05000241499068908	88832641

4 rows 10 columns (6 shown)

Run Time (s): real 3.450 user 33.945318 sys 3.986566

D |

Q2 TPC-h pada CSV files

9991.62	Supplier#000272821	CHINA	...	NHqLaQnDOLK6dK6tJzq	28-209-404-4693	kly final deposits...	
9991.62	Supplier#000272821	CHINA	...	NHqLaQnDOLK6dK6tJzq	28-209-404-4693	kly final deposits...	
.
.
9970.37	Supplier#000538491	INDONESIA	...	4tCExua2TSVZc0s6Ju...	19-984-612-7213	unts aCustomer sil...	
9970.33	Supplier#000203273	VIETNAM	...	nq95w8jHFCzJlnXuD...	31-622-938-5047	lyly daringly expr...	
9970.06	Supplier#000494220	INDIA	...	AsNmMrxNvp4MEcG4Fk...	18-566-441-1457	special deposits ...	
9968.93	Supplier#000055759	VIETNAM	...	uVHbs6mK,4	31-211-861-1258	lace of the carefu...	
9968.59	Supplier#000512636	JAPAN	...	Hsk1,qGyoOyyjAl,Vh...	22-468-913-3394	, ironic dependenc...	
9968.47	Supplier#000496607	CHINA	...	Op9HoJimpMgmhISvDq...	28-769-525-7872	l dolphins are sil...	
9968.03	Supplier#000546112	VIETNAM	...	y2SH3CUWEU3CnM02Kg...	31-412-475-6235	cording to the iro...	
9967.03	Supplier#000527133	INDIA	...	,Xhz8WLb,DngaJlApe...	18-126-283-6895	ans haggle. final ...	
9966.72	Supplier#000147528	VIETNAM	...	NHshmuKR70yKGtegD	31-340-465-6146	ole slyly. special...	
9966.17	Supplier#000143040	INDONESIA	...	hxboclgdUxNNF9R_n,4A	19-173-638-3695	accounts! thin, i...	
9965.69	Supplier#000368609	INDIA	...	TbJThP3MD6BtyGWUzd...	18-569-590-5992	gular deposits sle...	
9965.51	Supplier#000425133	CHINA	...	E4DB1hJqA6b3JccJ	28-656-592-7894	ely. carefully iro...	
9965.51	Supplier#000340917	CHINA	...	E4DB1hJqA6b3JccJ	18-845-450-2491	odolites cajole fu...	
9964.67	Supplier#00009953	JAPAN	...	aUpOTXDPk3oGFbbhiCPe	22-258-738-4585	use carefully alo...	
9964.27	Supplier#000482260	INDIA	...	n7SV8kWIvpvQ0lo81A...	18-312-704-8478	to the quickly reg...	
9964.27	Supplier#000482260	INDIA	...	H8wxZpudZf0oD4qvts...	18-312-704-8478	to the quickly reg...	
9964.0	Supplier#000181635	INDONESIA	...	H8wxZpudZf0oD4qvts...	19-893-689-3730	gage furiously flu...	
9963.46	Supplier#000289108	JAPAN	...	ywGIaphKXzoY1y0IJu...	22-356-171-1540	lyly carefully bol...	
9962.79	Supplier#000373066	INDONESIA	...	XrRs4,Q rPgywDQ596...	19-279-176-4748	riously express fo...	

100 rows (40 shown)

8 columns (6 shown)

Run Time (s): real 9.723 user 33.563440 sys 9.398698

D |

Q2 TPC-h pada Parquet files

9991.62	Supplier#000272821	CHINA	...	NHqLaQnDOLK6dK6tJzq	28-209-404-4693	kly final deposits...	
9991.62	Supplier#000272821	CHINA	...	NHqLaQnDOLK6dK6tJzq	28-209-404-4693	kly final deposits...	
.
.
9970.37	Supplier#000538491	INDONESIA	...	4tCExua2TSVZc0s6Ju...	19-984-612-7213	unts aCustomer sil...	
9970.33	Supplier#000203273	VIETNAM	...	nq95w8jHFCzJlnXuD...	31-622-938-5047	lyly daringly expr...	
9970.06	Supplier#000494220	INDIA	...	AsNmMrxNvp4MEcG4Fk...	18-566-441-1457	special deposits ...	
9968.93	Supplier#000055759	VIETNAM	...	uVHbs6mK,4	31-211-861-1258	lace of the carefu...	
9968.59	Supplier#000512636	JAPAN	...	Hsk1,qGyoOyyjAl,Vh...	22-468-913-3394	, ironic dependenc...	
9968.47	Supplier#000496607	CHINA	...	Op9HoJimpMgmhISvDq...	28-769-525-7872	l dolphins are sil...	
9968.03	Supplier#000546112	VIETNAM	...	y2SH3CUWEU3CnM02Kg...	31-412-475-6235	cording to the iro...	
9967.03	Supplier#000527133	INDIA	...	,Xhz8WLb,DngaJlApe...	18-126-283-6895	ans haggle. final ...	
9966.72	Supplier#000147528	VIETNAM	...	NHshmuKR70yKGtegD	31-340-465-6146	ole slyly. special...	
9966.17	Supplier#000143040	INDONESIA	...	hxboclgdUxNNF9R_n,4A	19-173-638-3695	accounts! thin, i...	
9965.69	Supplier#000368609	INDIA	...	TbJThP3MD6BtyGWUzd...	18-569-590-5992	gular deposits sle...	
9965.51	Supplier#000425133	CHINA	...	E4DB1hJqA6b3JccJ	28-656-592-7894	ely. carefully iro...	
9965.51	Supplier#000425133	CHINA	...	E4DB1hJqA6b3JccJ	28-656-592-7894	ely. carefully iro...	
9964.67	Supplier#000340917	INDIA	...	aUpOTXDPk3oGFbbhiCPe	18-845-450-2491	odolites cajole fu...	
9964.58	Supplier#00009953	JAPAN	...	n7SV8kWIvpvQ0lo81A...	22-258-738-4585	use carefully alo...	
9964.27	Supplier#000482260	INDIA	...	H8wxZpudZf0oD4qvts...	18-312-704-8478	to the quickly reg...	
9964.27	Supplier#000482260	INDIA	...	H8wxZpudZf0oD4qvts...	18-312-704-8478	to the quickly reg...	
9964.0	Supplier#000181635	INDONESIA	...	ywGIaphKXzoY1y0IJu...	19-893-689-3730	gage furiously flu...	
9963.46	Supplier#000289108	JAPAN	...	XrRs4,Q rPgywDQ596...	22-356-171-1540	lyly carefully bol...	
9962.79	Supplier#000373066	INDONESIA	...	RLPM4029nTX4	19-279-176-4748	riously express fo...	

100 rows (40 shown)

8 columns (6 shown)

Run Time (s): real 0.787 user 4.604903 sys 1.852002

D |

Q3 TPC-h pada CSV files

```
orders,
lineitem
WHERE c_mktsegment = 'BUILDING'
  AND c_custkey    = o_custkey
  AND l_orderkey   = o_orderkey
  AND o_orderdate < DATE '1995-03-15'
  AND l_shipdate   > DATE '1995-03-15'
GROUP BY l_orderkey, o_orderdate, o_shippriority
ORDER BY revenue DESC, o_orderdate
LIMIT 10;
100%
```

l_orderkey int64	revenue double	o_orderdate date	o_shippriority int64
107677314	493861.56810000003	1995-03-07	0
221295395	484987.7808	1995-02-25	0
132278465	467147.0282	1995-03-01	0
212980356	465037.83439999993	1995-02-14	0
242271747	461284.44739999995	1995-03-13	0
277818180	459517.6084	1995-03-13	0
49108805	459214.7448	1995-02-24	0
322002375	456716.5754	1995-03-06	0
75796454	456068.11179999996	1995-03-10	0
117041860	451737.339	1995-03-06	0

10 rows 4 columns

Run Time (s): real 36.517 user 175.463326 sys 28.325994

D |

Q3 TPC-h pada Parquet files

```
FROM customer,
      orders,
      lineitem
WHERE c_mktsegment = 'BUILDING'
  AND c_custkey    = o_custkey
  AND l_orderkey   = o_orderkey
  AND o_orderdate  < DATE '1995-03-15'
  AND l_shipdate   > DATE '1995-03-15'
GROUP BY l_orderkey, o_orderdate, o_shippriority
ORDER BY revenue DESC, o_orderdate
LIMIT 10;
```

l_orderkey int64	revenue double	o_orderdate date	o_shippriority int64
107677314	493861.56810000003	1995-03-07	0
221295395	484987.7808	1995-02-25	0
132278465	467147.0282	1995-03-01	0
212980356	465037.83439999993	1995-02-14	0
242271747	461284.44739999995	1995-03-13	0
277818180	459517.6084	1995-03-13	0
49108805	459214.7448	1995-02-24	0
322002375	456716.5754	1995-03-06	0
75796454	456068.11179999996	1995-03-10	0
117041860	451737.339	1995-03-06	0

10 rows

4 columns

Run Time (s): real 1.588 user 16.173434 sys 3.622199
D |

Q5 TPC-h pada CSV files

```
FROM customer,
      orders,
      lineitem,
      supplier,
      nation,
      region
 WHERE c_custkey    = o_custkey
   AND l_orderkey   = o_orderkey
   AND l_suppkey    = s_suppkey
   AND c_nationkey  = s_nationkey
   AND s_nationkey  = n_nationkey
   AND n_regionkey  = r_regionkey
   AND r_name        = 'ASIA'
   AND o_orderdate >= DATE '1993-01-01'
   AND o_orderdate <  DATE '1994-01-01'
 GROUP BY n_name
 ORDER BY revenue DESC;
```

100%

n_name varchar	revenue double
VIETNAM	3217676683.6969004
INDONESIA	3200420916.1836996
INDIA	3185369892.0217004
JAPAN	3167844885.420297
CHINA	3155220897.983099

Run Time (s): real 37.332 user 168.295563 sys 29.200941
D |

Q5 TPC-h pada Parquet files

```
        SUM(l_extendedprice * (1 - l_discount)) AS revenue
    FROM customer,
         orders,
         lineitem,
         supplier,
         nation,
         region
   WHERE c_custkey      = o_custkey
     AND l_orderkey    = o_orderkey
     AND l_suppkey     = s_suppkey
     AND c_nationkey   = s_nationkey
     AND s_nationkey   = n_nationkey
     AND n_regionkey   = r_regionkey
     AND r_name         = 'ASIA'
     AND o_orderdate   >= DATE '1993-01-01'
     AND o_orderdate   <  DATE '1994-01-01'
   GROUP BY n_name
  ORDER BY revenue DESC;
```

n_name varchar	revenue double
VIETNAM	3217676683.6969004
INDONESIA	3200420916.1837006
INDIA	3185369892.0216994
JAPAN	3167844885.420299
CHINA	3155220897.9830995

Run Time (s): real 1.456 user 17.409175 sys 2.626334
D |

Q12 TPC-h pada CSV files

```

INDONESIA | 3200420916.1836996
INDIA    | 3185369892.0217004
JAPAN    | 3167844885.420297
CHINA   | 3155220897.983099

Run Time (s): real 37.332 user 168.295563 sys 29.200941
D | SELECT
      l_shipmode,
      SUM(CASE WHEN o_orderpriority IN ('1-URGENT','2-HIGH') THEN 1 ELSE 0 END) AS high_line_count,
      SUM(CASE WHEN o_orderpriority NOT IN ('1-URGENT','2-HIGH') THEN 1 ELSE 0 END) AS low_line_count
  FROM orders,
       lineitem
 WHERE o_orderkey = l_orderkey
   AND l_shipmode IN ('MAIL','SHIP')
   AND l_commitdate < l_receiptdate
   AND l_shipdate < l_commitdate
   AND l_receiptdate >= DATE '1993-01-01'
   AND l_receiptdate < DATE '1994-01-01'
 GROUP BY l_shipmode
 ORDER BY l_shipmode;
100% [REDACTED]

```

l_shipmode	high_line_count	low_line_count
MAIL	373287	560880
SHIP	373948	560693

```

Run Time (s): real 33.383 user 174.846766 sys 24.451699
D |

```

Q12 TPC-h pada Parquet files

```

Run Time (s): real 1.456 user 17.409175 sys 2.626334
D | SELECT
      l_shipmode,
      SUM(CASE WHEN o_orderpriority IN ('1-URGENT','2-HIGH') THEN 1 ELSE 0 END) AS high_line_count,
      SUM(CASE WHEN o_orderpriority NOT IN ('1-URGENT','2-HIGH') THEN 1 ELSE 0 END) AS low_line_count
  FROM orders,
       lineitem
 WHERE o_orderkey = l_orderkey
   AND l_shipmode IN ('MAIL','SHIP')
   AND l_commitdate < l_receiptdate
   AND l_shipdate < l_commitdate
   AND l_receiptdate >= DATE '1993-01-01'
   AND l_receiptdate < DATE '1994-01-01'
 GROUP BY l_shipmode
 ORDER BY l_shipmode;

```

l_shipmode	high_line_count	low_line_count
MAIL	373287	560880
SHIP	373948	560693

```

Run Time (s): real 1.064 user 10.501783 sys 1.913964
D |

```

Berikut merupakan hasil perbandingan query ke CSV dan Parquet.

Query	CSV	Parquet	Selisih wall-time	Speedup Parquet terhadap CSV
Q1	Real 32.751	Real 3.450	29.301	9.49x

	User 181.917723 Sys 26.256410	User 33.945318 Sys 3.986566		
Q2	Real 9.723 User 33.563440 Sys 9.398698	Real 0.787 User 4.604903 Sys 1.852002	8.936	12.35x
Q3	Real 36.517 User 175.463326 Sys 28.325994	Real 1.588 User 16.173434 Sys 3.622199	34.929	23.00x
Q5	Real 37.332 User 168.295563 Sys 29.200941	Real 1.456 User 17.409175 Sys 2.626334	35.876	25.65x
Q12	Real 33.383 User 174.846766 Sys 24.451699	Real 1.064 User 10.501783 Sys 1.913964	32.319	31.38x

Berdasarkan hasil *benchmark* dengan kelima *query* di atas, percepatan yang dihasilkan dengan menggunakan Parquet dibandingkan CSV berkisar dari 9 hingga 31 kali lipat. Apabila ditinjau dari waktu *real* (*wall-time*), *user*, dan *sys*, sebenarnya baik CSV maupun Parquet memungkinkan *multi-threading*. Buktinya, kedua format memiliki *wall-time* yang lebih pendek dibandingkan waktu *user* dan *sys*. Namun, *baseline* waktu dari Parquet sendiri memang lebih rendah sehingga hasil akhirnya jauh lebih cepat dibandingkan CSV.

CSV merupakan format lama berupa *row-based plain text* yang dipisahkan oleh koma, *hence the name comma-separated*. Meskipun CSV memiliki keunggulan berupa bersifat *human-readable*, tetapi pemrosesannya sangat tidak efisien. Program harus membaca tiap baris, melakukan *parsing*, dan membagi berdasarkan delimiternya. Belum lagi jika ada kasus khusus

seperti *quoted comma*. Alih-alih menyimpan dalam bentuk *plain text*, Parquet merupakan *binary blobs* yang bersifat *columnar* sehingga unggul untuk data besar yang memiliki banyak kolom dan terkadang *sparse*. Sebagai *binary-format*, Parquet didesain dengan efisiensi pemrosesan sebagai prioritas utama. Parquet mendukung *predicate pushdown* sehingga DuckDB dapat melewati data yang dianggap tidak perlu pada *filtering*. Terakhir, ukuran berkas Parquet juga jauh lebih kecil dibandingkan CSV sehingga mengurangi waktu yang dibutuhkan untuk I/O (penyebab *bottleneck* utama di komputasi modern).

Sebagai klarifikasi, kami tidak melakukan perbandingan antara CSV dengan Parquet yang telah *di-load* menjadi tabel Iceberg karena menggunakan *hosted service* (Snowflake) sehingga sukar untuk dibandingkan dengan adil. Faktor perbedaan *hardware* dapat memengaruhi hasil secara signifikan.

3) Isu kompatibilitas pada format Parquet.

Seperti yang telah dijelaskan sebelumnya, kami membuat *file* Parquet secara lokal menggunakan *library* PyArrow sebelum *di-upload* ke S3 dan dibaca oleh Spark pada AWS Glue Notebook. Hampir semua *file* dapat dibaca dengan baik, tetapi ada beberapa *file* yang tidak dapat terbaca karena terdapat perbedaan cara menyimpan data *timestamp* pada PyArrow dan Spark (sumber: <https://stackoverflow.com/a/77371160>).

The screenshot shows a Glue Notebook interface with a single cell containing PySpark code. The code reads a Parquet file from an S3 bucket and attempts to show its contents. However, it fails with a Java exception:

```
[15]: df2 = spark.read.parquet("s3://if4044-big-data-kol-4/tpc-h-5gb-parquet/0BDERS/orders.parquet")
df2.show(5)

Py4JJavaError: An error occurred while calling doAll.parquet.
: org.apache.spark.SparkException: Job aborted due to stage failure: Task 0 in stage 15.0 failed 4 times, most recent failure: Lost task 0.3 in stage 15.0 (TID 27) (172.34.95.156 executor 1): org.apache.spark.sql.AnalysisException: Input Parquet type: INT32 (TRIMMED) is not supported.
```

This error occurs because the schema of the Parquet file is not compatible with Spark's requirements. Specifically, it's trying to read an INT32 column as TRIMMED, which is not supported.

Gambar 2.7 Error saat membaca *file* Parquet dari AWS Glue Notebook

Terdapat dua alternatif untuk menangani ini, yaitu:

- memperbaiki *script* `tbl_to_parquet.py` yang telah dibuat sebelumnya dan menjalankan ulang *script* tersebut pada beberapa *file* yang tidak bisa dibaca oleh Spark
- membuat *script* baru (`fix_parquet_schema.py`) untuk membaca *file* Parquet, mengubah skema agar kompatibel dengan Spark, dan menyimpannya kembali.

Kami memilih pilihan b, karena jika menjalankan ulang *script* pertama, maka *script* akan membaca kembali *file* TBL, sedangkan jika menjalankan *script* kedua, maka *script* akan membaca *file* Parquet, sehingga lebih cepat. Kami menduga di dunia nyata, banyak kasus di mana membuat *script* baru untuk memperbaiki format data akan lebih cepat daripada memperbaiki dan menjalankan ulang *script* lama. Meskipun begitu, tidak ada salahnya memperbaiki *script* lama agar dapat digunakan kembali lain waktu.

4) *Reproducibility* pada teknologi big data.

Kami sering menemukan kasus di mana ketika menjalankan suatu *command* pertama kalinya, *command* tersebut berhasil, namun ketika dijalankan

ulang, *command* tersebut mengeluarkan *error*. Beberapa kemungkinan penyebabnya adalah sebagai berikut.

- *Command* yang tidak idempotent (misal CREATE TABLE vs CREATE TABLE IF NOT EXISTS).
- Jupyter Notebook menggunakan *stateful kernel*, sehingga dapat menyimpan variabel lama yang sebenarnya sudah di-reassign tetapi *code block* untuk me-reassign variabel tersebut belum dijalankan.
- Ketika ingin mengulangi pembuatan tabel, terdapat beberapa residu dari tabel lama yang belum terhapus, misalnya *metadata*, *lock file*, dsb.
- Belum familiar dengan teknologi yang digunakan.

5) Isu ketika menggunakan teknologi *cloud*

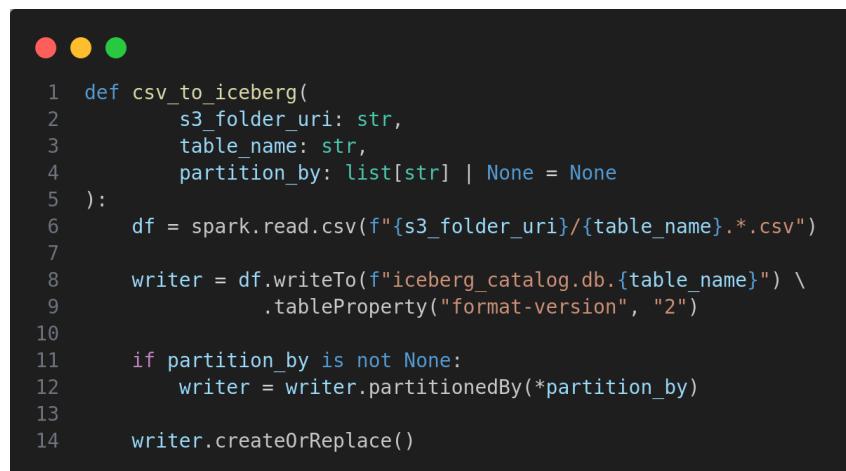
- *Billing* membengkak, karena biaya AWS Glue dihitung per jam *compute unit*, dan kami membuka sesi AWS Glue Notebook berjam-jam karena perlu beberapa kali *trial and error* sebelum berhasil memasukkan data ke Iceberg. Itupun setelah berhasil, saat dicoba dijalankan ulang terkadang malah gagal lagi 😊 (mungkin kak Rendy bisa memberikan tips agar dapat leluasa *trial and error* di *cloud* tanpa harus khawatir *billing* membengkak. Awalnya kami sebenarnya sudah mencoba di lokal, tetapi ada beberapa perbedaan ketika menjalankan di *cloud*, sehingga kami memerlukan *trial and error* yang cukup banyak di lingkungan *cloud*).
- *Setup* IAM Role dan Policy yang cukup rumit, terutama untuk integrasi dengan *third-party app* seperti Snowflake.
- Meskipun kami sudah cukup familiar dengan teknologi *object storage*, khususnya yang berbasis API s3, selama ini kami lebih berfokus pada mengunggah berkas berukuran kecil hingga sedang tetapi dengan kuantitas yang banyak. Sementara itu, pada tugas ini, kami harus mengunggah berkas CSV yang berukuran sangat besar,

mencapai puluhan GB. Hal ini mengharuskan program yang kami gunakan, yakni rclone untuk memecah berkas (*chunking*). Namun, ketika kami mengunggah berkas, terkadang ditemukan *error* akibat informasi otentikasi yang berubah ketika berganti *chunk*. Singkat cerita, ternyata rclone yang kami gunakan merupakan versi lama (diunduh dari *package manager* Ubuntu). Akhirnya, kami perlu menggunakan versi terbaru (masih beta) dari rclone yang tersedia melalui situs resmi mereka. Hal ini menarik karena Cloudflare R2 merupakan layanan yang cukup populer, tetapi dukungannya secara resmi dari rclone baru ada di versi beta.

- Kami sempat ketiduran dan lupa mematikan *instance VM* sehingga terkena *billing* dengan sia-sia 🤦‍♂️💰💸.

6) Beberapa *improvement* yang dapat dilakukan:

- a) Sebenarnya *file CSV* tidak perlu dikonversi ke Parquet secara lokal sebelum *di-upload* ke S3, karena konversi seharusnya dapat dilakukan oleh Spark secara otomatis saat memasukkan data ke tabel Iceberg. Oleh karena itu, fungsi untuk memasukkan data ke tabel Iceberg dapat diubah sebagai berikut. (Hal ini tidak kami lakukan karena baru disadari di akhir).



```
 1 def csv_to_iceberg(
 2     s3_folder_uri: str,
 3     table_name: str,
 4     partition_by: list[str] | None = None
 5 ):
 6     df = spark.read.csv(f"{s3_folder_uri}/{table_name}.*.csv")
 7
 8     writer = df.writeTo(f"iceberg_catalog.db.{table_name}") \
 9         .tableProperty("format-version", "2")
10
11     if partition_by is not None:
12         writer = writer.partitionedBy(*partition_by)
13
14     writer.createOrReplace()
```

Gambar 3.2 Fungsi untuk memasukkan data dari CSV ke tabel Iceberg

- b) Melakukan partisi tabel Iceberg berdasarkan kolom tertentu agar *query* semakin efisien. Sebenarnya kami sudah berencana untuk melakukan ini (terdapat parameter `partition_by` pada fungsi `parquet_to_iceberg` pada Gambar 1.7, tetapi parameter tersebut tidak kami gunakan karena tidak sempat mencari kolom mana yang ideal untuk dijadikan `partition_by` pada tiap tabel)
- c) Seharusnya konversi data TPC-H dari DuckDB ke Parquet bisa dilakukan sejak awal. Pada tugas ini, awalnya kami melakukan *export* ke bentuk CSV, lalu dikonversi kembali menjadi Parquet. Namun, proses tersebut memakan waktu yang sangat-sangat lama karena *parsing* CSV berupa CPU-bound dan tidak dapat memanfaatkan multi-core (padahal kami sudah pakai *instance* dengan 16 vCPU). Pada waktu itu, kami tidak menyimpan berkas .db yang dihasilkan, sehingga pilihannya hanya *generate* dataset TPC-H dari nol, kemudian konversi ke CSV dan Parquet sekaligus atau mengonversi berkas CSV yang sudah ada. Akhirnya, kami memilih untuk mengonversi saja karena pengunggahan CSV ke R2 cukup lama meskipun sudah menggunakan *provider* VM yang koneksi internetnya cukup bagus (RunPod), kemungkinan di-throttle karena kebanyakan *upload*, hehe.

7) Penerapan Prinsip *Software Engineering* dengan dbt

Sebagai alat yang bekerja sebagai *data orchestrator*, dbt memiliki banyak fitur yang memungkinkan *developer* untuk menjaga agar kode atau *query* yang ditulis bersifat *readable*, *reusable*, dan *maintainable*. Namun, di luar fitur yang disediakan, ada pula hal-hal yang terkait dengan cara penggunaan fitur-fitur tersebut. Berikut adalah hal-hal yang dapat diterapkan untuk menjaga prinsip *software engineering* pada dbt.

- i) **Gunakan descriptive naming.** Berikan nama yang jelas pada model, kolom, dan *file* yang dibuat. Apabila terlalu panjang, pastikan

- singkatan/akronim yang digunakan lazim digunakan dan dicatat di tempat lain, misalnya istilah stg untuk *staging*.
- ii) **Tulis query SQL dengan rapi.** Pastikan query memiliki alur berpikir yang jelas dan mudah dipahami. Tambahkan komentar jika diperlukan. Hindari *subquery* bertumpuk, gunakan CTE. Apabila dibutuhkan, maka bisa coba gunakan *linter* seperti sqlfluff.
 - iii) **Tulis dokumentasi di file schema.yml.** Dokumentasi terkait deskripsi model, kolom, dsb sangat penting agar dokumentasi dari dbt dapat dibaca dan benar-benar bermanfaat.
 - iv) **Pecah model menjadi hierarki yang terstruktur.** Jangan buat semua model berada pada tingkatan yang sama. Misalnya, pada tugas ini, kami membuat model *staging* terkait *sales*. Penggunaan hierarki juga dapat membantu mengorganisasi dependensi model serta *reuse* model secara lebih efisien pada model lain. Contoh tingkatan yang dapat digunakan misalnya *staging*, *intermediate*, lalu *facts* dan *dimension*.
 - v) **Gunakan ref() untuk dependensi.** Alih-alih melakukan query ulang dari nol, gunakan sintaks *ref()* agar dbt dapat mengetahui dependensi/sumber data dari model tersebut. Dependensi ini juga dapat dilacak melalui graf dependensi dari dokumentasi dbt.
 - vi) **Buat test dengan dbt.** Penggunaan *dbt test* akan membantu menjaga kualitas data yang digunakan. Tes ini juga dapat diotomatisasi dengan CI/CD atau *scheduler* seperti Airflow.
 - vii) **Modularisasi logic dengan macros.** Penggunaan *macros* akan membuat kode bisa digunakan ulang di tempat lain serta mempersingkat penulisan *query-query* kompleks yang punya *logic* berulang di berbagai tempat.
 - viii) **Gunakan parameterisasi jinja.** Dengan menggunakan jinja, kita bisa mengubah parameter secara fleksibel, baik saat *runtime* maupun

- secara global melalui *file config* tertentu. Hal tersebut membuat kode lebih mudah dikostumisasi dibandingkan melakukan *hard code*.
- ix) **Manfaatkan dbt docs.** Fitur dokumentasi otomatis dari dbt dapat membantu tim memahami fungsi dan peran dari model maupun *macro* yang ada. Dokumentasi tersebut bahkan dapat menunjukkan *lineage* atau garis keturunan model sehingga mempermudah proses pelacakan sumber data.

4. Repository

<https://github.com/haziqam/if4044-data-lakehouse>