Team 20 Code Sample doc

Environment setup

1. Spark installation: MAC:
https://github.com/charlie-ph/BigDataAnalytics/blob/master/Installations-HowTos/How-To-Install-Spark-On-MACOS.md

Windows:
https://github.com/charlie-ph/BigDataAnalytics/blob/master/Installations-HowTos/How-To-Install-Spark-On-Windows.md

2. PostgreSQL installation:
https://www.postgresql.org/docs/current/tutorial-install.html

How to run the code

Part 1: PostgreSQL
1) run code/Postgre/create_file.sql to create Data-warehouse in Postgre
2) import all csv files in /data into your database
3) run code/Postgre/SQL_basic_search.sql to get SQL results

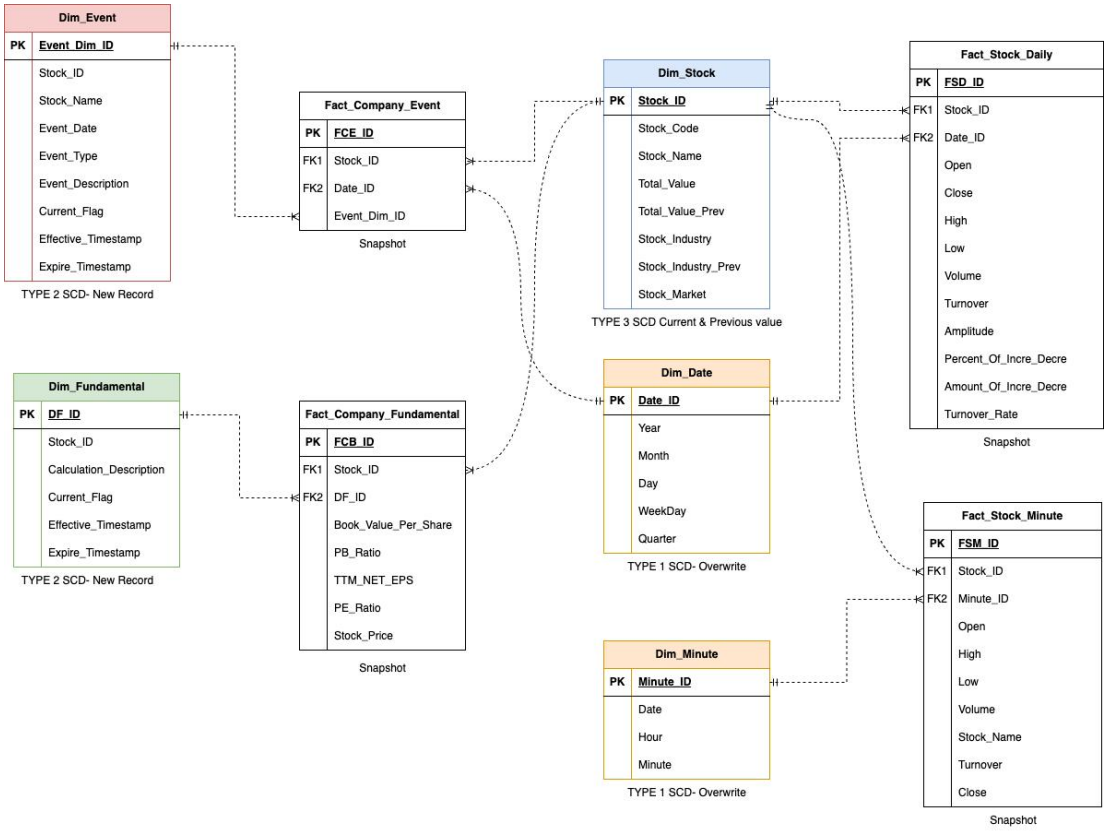
Part2: Spark
1) run code/Spark/Example.ipynb in your notebook.
2) make sure you have correct address to access all csv files.

Dataset Explanation

I designed this database according to some basic requirements of my own trading system.

Req1 Data for all Instruments	Minute and day stock data, including open, close, high, low, etc.
Req 2 Company Basic Info	Basic information about the company including market capitalization, the company's industry, etc.
Req 3 Significant events occurring in the company(CN Market)	Significant events occurring in the company include asset reorganization, external guarantees, share pledges, asset acquisitions, etc.
Req 4 Company Earnings Data(US Market)	Earnings data released by the company each quarter, including PB, PE, etc.

Database ERD:



For each table, we provide one notebook to finish the ETL process(code/Database/*.ipynb). All the data is from this API: akshare(https://akshare.akfamily.xyz/)

Results of running the code with data

Seq1: select 10 stocks from Shanghai Stock Exchange(SSE)

PostgreSQL:

```sql
-- select any stock from Shanghai Stock Exchange(SSE)
SELECT ds.stock_id, ds.total_value from "Trading".Dim_Stock ds
where stock_id>600000
order by stock_id
limit 10;
```

Data Output | Messages | Explain | Notifications

	stock_id [PK] integer	total_value numeric (20,2)
1	600004	25820896467.53
2	600006	12820000000.00
3	600007	21102569087.30
4	600008	20113218454.98
5	600009	91352149991.40
6	600010	71739808751.84
7	600011	116165890856.60
8	600012	17150027400.00
9	600015	90715092267.60
10	600016	167686662862.66

Spark:

```
dim_stock_df = spark.read.csv('Dim_Stock.csv', header=True, inferSchema=True)
sse_stocks_df = dim_stock_df.filter(col("stock_id") > 600000).orderBy("stock_id").limit(10)
sse_stocks_df.select('Stock_ID', 'Total_Value').show()
```

Select Cell Language Mode

4] ✓ 0.3s Python

```
+--------+--------------------+
|Stock_ID|         Total_Value|
+--------+--------------------+
|  600004|   2.582089646753E10|
|  600006|            1.282E10|
|  600007|    2.11025690873E10|
|  600008|2.011321845498000...|
|  600009|9.135214999140001E10|
|  600010|   7.173980875184E10|
|  600011|   1.161658908566E11|
|  600012|       1.71500274E10|
|  600015|    9.07150922676E10|
|  600016|  1.6768666286266E11|
+--------+--------------------+
```

Seq2: select all stocks that rise over 9.9% on 2024-03-11

PostgreSQL:

```sql
--select all stocks that rise over 9.9% on 2024-03-11
SELECT * from "Trading".Fact_Stock_Daily
where Date_ID = 20240311 and percent_of_incre_decre>9.9
order by percent_of_incre_decre desc;

--select infomation from 20231120 9:30-9:40 on stock SH600004
```

Data Output | Messages | Explain | Notifications

	fsd_id [PK] bigint	stock_id integer	date_id integer	open numeric (10,2)	high numeric (10,2)	low numeric (10,2)	volume integer	turn
1	20240311300317	300317	20240311	4.04	4.84	4.04	972902	
2	20240311300324	300324	20240311	2.98	3.29	2.88	1842843	
3	20240311300530	300530	20240311	17.80	21.40	17.80	94640	
4	20240311300438	300438	20240311	22.01	25.91	22.01	425121	
5	20240311301202	301202	20240311	28.70	34.42	27.66	136343	
6	20240311301292	301292	20240311	14.16	16.85	14.16	50492	
7	20240311300890	300890	20240311	23.37	27.95	23.37	76936	
8	20240311301349	301349	20240311	28.89	34.66	28.89	30457	
9	20240311300712	300712	20240311	21.48	25.14	21.37	92134	
10	20240311688063	688063	20240311	85.00	99.60	85.00	136891	
11	20240311301205	301205	20240311	91.00	116.62	91.00	133827	

Spark:

```python
fact_stock_daily_df = spark.read.csv('Fact_Stock_Daily_2024.csv', header=True, inferSchema=True)
rising_stocks_df = fact_stock_daily_df.filter((col("Date_ID") == 20240311) & (col("percent_of_incre_decre") > 9.9))\
                   .orderBy(col("percent_of_incre_decre").desc())
rising_stocks_df.show()
```

0.9s Python

```
+--------------+--------+--------+-----+-----+-----+-------+--------------+---------+--------------------+--------------------+
|        FSD_ID|Stock_ID| Date_ID| Open| High|  Low| Volume|      Turnover|Amplitude|Percent_Of_Incre_Decre|Amount_Of_Incre_Decre|
+--------------+--------+--------+-----+-----+-----+-------+--------------+---------+--------------------+--------------------+
|20240311300317|  300317|20240311| 4.04| 4.84| 4.04| 972902|  4.60648926E8|    19.85|                20.1|                0.81|
|20240311300324|  300324|20240311| 2.98| 3.29| 2.88|1842843|  5.68963029E8|    14.96|               20.07|                0.59|
|20240311300530|  300530|20240311| 17.8| 21.4| 17.8|  94640|  1.91813994E8|    20.19|               20.02|                3.57|
|20240311301292|  301292|20240311|14.16|16.85|14.16|  50492|   8.4292834E7|    19.16|               20.01|                2.81|
|20240311301202|  301202|20240311| 28.7|34.42|27.66| 136343|  4.30519547E8|    23.57|               20.01|                5.74|
|20240311300890|  300890|20240311|23.37|27.95|23.37|  76936|  2.03484159E8|    19.67|               20.01|                4.66|
|20240311301349|  301349|20240311|28.89|34.66|28.89|  30457|  1.01086693E8|    19.98|               20.01|                5.78|
|20240311300438|  300438|20240311|22.01|25.91|22.01| 425121| 1.021499595E9|    18.06|               20.01|                4.32|
|20240311688063|  688063|20240311| 85.0| 99.6| 85.0| 136891|  1.26584976E9|    17.59|                20.0|                16.6|
|20240311300712|  300712|20240311|21.48|25.14|21.37|  92134|  2.19739943E8|     18.0|                20.0|                4.19|
|20240311301205|  301205|20240311| 91.0|116.62| 91.0| 133827| 1.369576675E9|    26.36|               17.31|                16.8|
|20240311300025|  300025|20240311| 8.54| 10.08| 8.38| 306597|  2.85149119E8|    20.09|               17.14|                1.49|
|20240311300769|  300769|20240311|41.01|46.89|41.01| 302334| 1.342825861E9|    14.71|               16.59|                6.63|
|20240311300953|  300953|20240311| 45.8|  53.0| 45.8|  31804|  1.59406988E8|    15.72|               15.31|                7.01|
|20240311688248|  688248|20240311|27.78|31.31|27.78| 171501|  5.19394678E8|    13.07|               15.07|                4.07|
|20240311300750|  300750|20240311|165.0|181.65|164.1| 763350|1.3248182793E10|    11.11|               14.46|               22.85|
|20240311301236|  301236|20240311|46.95| 55.88| 46.8|1141301|  5.880365894E9|    18.92|                14.4|                6.91|
|20240311300709|  300709|20240311| 38.8| 44.98|36.89| 446070|  1.808201963E9|     21.3|               14.01|                5.32|
|20240311300014|  300014|20240311| 38.6|42.76|38.55| 648947|  2.651262664E9|    11.19|                13.0|                4.89|
|20240311300411|  300411|20240311| 8.12|  9.49| 8.12| 962847|  8.43633186E8|    17.19|               12.92|                1.03|
+--------------+--------+--------+-----+-----+-----+-------+--------------+---------+--------------------+--------------------+
only showing top 20 rows
```

Seq3: select information from 20231120 9:30-9:40 on stock SH600004

PostgreSQL:

```
19  --select infomation from 20231120 9:30-9:40 on stock SH600004
20  SELECT * from "Trading".Fact_Stock_Minute
21  where Minute_ID>202311200930 and Minute_ID<202311200940 and stock_id = 600004;
22
23  --show total minutes in 2024-03-11
```

Data Output | Messages | Explain | Notifications

	fsm_id [PK] bigint	stock_id integer	minute_id bigint	open numeric (10,2)	high numeric (10,2)	low numeric (10,2)	volume integer
1	600004202311200931	600004	202311200931	0.00	10.92	10.89	647
2	600004202311200932	600004	202311200932	0.00	10.90	10.89	401
3	600004202311200933	600004	202311200933	0.00	10.91	10.86	2568
4	600004202311200934	600004	202311200934	0.00	10.91	10.90	310
5	600004202311200935	600004	202311200935	0.00	10.95	10.91	1727
6	600004202311200936	600004	202311200936	0.00	10.94	10.92	202
7	600004202311200937	600004	202311200937	0.00	10.94	10.93	1230
8	600004202311200938	600004	202311200938	0.00	10.93	10.93	211
9	600004202311200939	600004	202311200939	0.00	10.94	10.93	86

Time Cost for PostgreSQL:

#	Node	Timings	
		Exclusive	Inclusive
1.	→ Bitmap Heap Scan on Trading.fact_stock… **Recheck Cond**: ((fact_stock_minute.stock_id = **Heap Blocks**: exact=2	0.489 ms	7.586 ms
2.	→ Bitmap AND (cost=644.38..644.38 r…	0.004 ms	7.098 ms
3.	→ Bitmap Index Scan using idx_fs… **Index Cond**: (fact_stock_minute.stoc	2.269 ms	2.269 ms
4.	→ Bitmap Index Scan using idx_fs… **Index Cond**: ((fact_stock_minute.min	4.825 ms	4.825 ms

Spark:

```
import time
start_time=  time.time()
fact_stock_minute_df = spark.read.csv('Fact_Stock_Minute.csv', header=True, inferSchema=True)
stock_600004_info_df = fact_stock_minute_df.filter((col("Minute_ID") > 202311200930) \
                                                   & (col("Minute_ID") < 202311200940) \
                                                    & (col("Stock_ID") == 600004))
stock_600004_info_df.show()
print(f"Time used: {time.time() - start_time}")
```

✓ 4.8s Python

```
[Stage 29:>                                                         (0 + 3) / 3]
+-----------------+--------+-------------+----+-----+-----+------+----------+--------+-----+
|           FSM_ID|Stock_ID|   Minute_ID|Open| High|  Low|Volume|Stock_Name| Turnover|Close|
+-----------------+--------+-------------+----+-----+-----+------+----------+--------+-----+
|600004202311200931|  600004|202311200931| 0.0|10.92|10.89|   647|   ▯▯▯2▯▯▯| 705561.0|10.91|
|600004202311200932|  600004|202311200932| 0.0| 10.9|10.89|   401|   ▯▯▯2▯▯▯| 436909.0|10.89|
|600004202311200933|  600004|202311200933| 0.0|10.91|10.86|  2568|   ▯▯▯2▯▯▯|2794153.0| 10.9|
|600004202311200934|  600004|202311200934| 0.0|10.91| 10.9|   310|   ▯▯▯2▯▯▯| 338083.0| 10.9|
|600004202311200935|  600004|202311200935| 0.0|10.95|10.91|  1727|   ▯▯▯2▯▯▯|1886825.0|10.94|
|600004202311200936|  600004|202311200936| 0.0|10.94|10.92|   202|   ▯▯▯2▯▯▯| 220870.0|10.93|
|600004202311200937|  600004|202311200937| 0.0|10.94|10.93|  1230|   ▯▯▯2▯▯▯|1345150.0|10.93|
|600004202311200938|  600004|202311200938| 0.0|10.93|10.93|   211|   ▯▯▯2▯▯▯| 230657.0|10.93|
|600004202311200939|  600004|202311200939| 0.0|10.94|10.93|    86|   ▯▯▯2▯▯▯|  94037.0|10.94|
+-----------------+--------+-------------+----+-----+-----+------+----------+--------+-----+

Time used: 4.820951700210571
```

Seq4: show total minutes in 2024-03-11

PostgreSQL:

```
23  --show total minutes in 2024-03-11
24  SELECT count(Minute_ID) AS Total_Minutes
25  from "Trading".Dim_Minute
26  where DATE_TRUNC('day',CAST(Date AS timestamp))='2024-03-11';
27
```

Data Output | Messages | Explain ✕ | Notifications

	total_minutes bigint
1	241

Spark:

```
dim_minute_df = spark.read.csv('Dim_Minute_2024.csv', header=True, inferSchema=True)
total_minutes_df = dim_minute_df.filter(col("Date").cast("date") == "2024-03-11")\
                                .agg({"Minute_ID": "count"})\
                                .withColumnRenamed("count(Minute_ID)", "Total_Minutes")
total_minutes_df.show()
```

✓ 0.9s Python

```
+-------------+
|Total_Minutes|
+-------------+
|          241|
+-------------+
```

Seq5: show total trading days in October

PostgreSQL:

```
28  --show total trading days in Feburary 2024
29  SELECT count(Date_ID) AS Total_Trading_Days
30  from "Trading".Dim_Date
31  where year = 2024 and month = 2;
32
```

Data Output | Messages | Explain ✕ | Notifications

	total_trading_days bigint
1	16

Spark:

```
dim_date_df = spark.read.csv('Dim_Date_2024.csv', header=True, inferSchema=True)
total_trading_days_df = dim_date_df.filter((col("year") == 2024) & (col("month") == 2))\
    .agg({"Date_ID": "count"})\
    .withColumnRenamed("count(Date_ID)", "Total_Trading_Days")
total_trading_days_df.show()
```

✓ 0.4s Python

```
+------------------+
|Total_Trading_Days|
+------------------+
|                16|
+------------------+
```

Seq6: recent company events from September

PostgreSQL:

```
--show recent company events from September
SELECT de.Stock_ID, de.Event_Date, de.Event_Type, de.Current_Flag,
       de.Effective_Timestamp, de.Expire_Timestamp
FROM "Trading".Fact_Company_Event fce
JOIN "Trading".Dim_Event de
ON fce.Event_Dim_ID = de.Event_Dim_ID
where fce.Date_ID>20230901 and fce.Stock_ID>600000 and fce.Stock_ID<600010;

--select Minute Level Data from 2024-03-11 9:30 to 2024-03-11 9:40
```

Data Output | Messages | Explain | Notifications

stock_id integer	event_date date	event_type character varying (50)	current_flag boolean	effective_timestamp timestamp without time zone	expire_timestamp timestamp without
600006	2023-09-15	Asset Restructuring	true	2023-09-15 00:00:00	[null]
600008	2023-11-14	Asset Restructuring	true	2023-11-14 00:00:00	[null]

Spark:

```
fact_company_event_df = spark.read.csv('Fact_Company_Event.csv', header=True, inferSchema=True).withColumnRenamed("Stock_ID","FCE
dim_event_df = spark.read.csv('Dim_Event.csv', header=True, inferSchema=True).withColumnRenamed("Stock_ID","DE_Stock_ID")

recent_events_df = fact_company_event_df.join(dim_event_df, fact_company_event_df.Event_Dim_ID == dim_event_df.Event_Dim_ID, "inn
                   .select("FCE_ID","FCE_Stock_ID","Date_ID","Event_Type","Current_Flag","Effective_Timestamp","Expire_Timestamp
                   .filter((col("Date_ID") > 20230901) & (col("FCE_Stock_ID") > 600000) & (col("FCE_Stock_ID") < 600010))

recent_events_df.show()
```

✓ 0.4s Python

```
+------------------+------------+--------+------------------+------------+-------------------+----------------+
|            FCE_ID|FCE_Stock_ID| Date_ID|        Event_Type|Current_Flag|Effective_Timestamp|Expire_Timestamp|
+------------------+------------+--------+------------------+------------+-------------------+----------------+
|20230915008260006|      600006|20230915|Asset Restructuring|           1|         2023-09-15|            NULL|
|20231114006360008|      600008|20231114|Asset Restructuring|           1|         2023-11-14|            NULL|
+------------------+------------+--------+------------------+------------+-------------------+----------------+
```

Seq7 select Minute Level Data from 2023-11-21 9:30 to 2023-11-21 9:40 where the company is in CN market and its Total_Value is in top 10

PostgreSQL:

```sql
SELECT fsm.FSM_ID, fsm.Stock_ID, fsm.Minute_ID, fsm.Close, fsm.High, fsm.Low, fsm.volu
FROM "Trading".Fact_Stock_Minute fsm
WHERE fsm.Stock_ID in (
    SELECT Stock_ID
    FROM "Trading".Dim_Stock
    WHERE Stock_Market = 'CN'
    ORDER BY Total_Value DESC
    LIMIT 10
)
AND fsm.Minute_ID >= 202311210930 AND fsm.Minute_ID <= 202311210940;
```

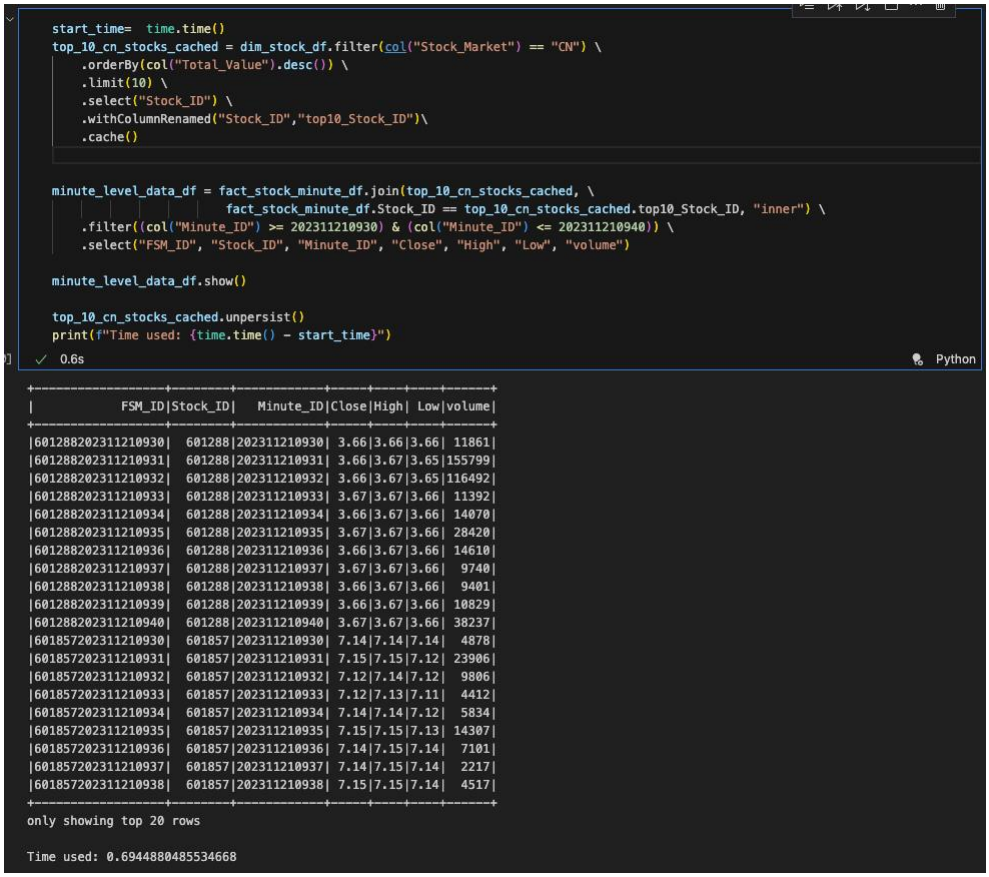
Data Output | Messages | Explain | Notifications

	fsm_id [PK] bigint	stock_id integer	minute_id bigint	close numeric (10,2)	high numeric (10,2)	low numeric (10,2)	volume integer
1	600938202311210930	600938	202311210930	19.12	19.12	19.12	
2	600938202311210931	600938	202311210931	19.06	19.14	19.05	4
3	600938202311210932	600938	202311210932	19.05	19.09	19.04	3
4	600938202311210933	600938	202311210933	19.01	19.05	19.00	3
5	600938202311210934	600938	202311210934	19.04	19.04	18.99	3
6	600938202311210935	600938	202311210935	19.05	19.05	19.04	3
7	600938202311210936	600938	202311210936	19.01	19.04	19.01	2
8	600938202311210937	600938	202311210937	19.02	19.02	19.01	2
9	600938202311210938	600938	202311210938	19.02	19.03	19.01	1
10	600938202311210939	600938	202311210939	19.02	19.03	19.01	1

Time Cost: 119.978ms

#	Node	Timings		Rows		
		Exclusive	Inclusive	Rows X	Actual	Plan
1.	→ Nested Loop Inner Join (cost=1371.75...	95.093 ms	119.978 ms	↓ 1.05	110	105
2.	→ Aggregate (cost=581.38..581.48 r... Buckets: Batches: Memory Usage: 24 k	0.157 ms	14.365 ms	↑ 1	10	10
3.	→ Limit (cost=581.23..581.25 r...	0.001 ms	14.209 ms	↑ 1	10	10
4.	→ Sort (cost=581.23..594....	1.151 ms	14.208 ms	↑ 530.9	10	5309
5.	→ Seq Scan on Tradin... Filter: ((dim_stock.stoc Rows Removed by Filte	13.057 ms	13.057 ms	↑ 1	5309	5309

Spark:



```
start_time=  time.time()
top_10_cn_stocks_cached = dim_stock_df.filter(col("Stock_Market") == "CN") \
    .orderBy(col("Total_Value").desc()) \
    .limit(10) \
    .select("Stock_ID") \
    .withColumnRenamed("Stock_ID","top10_Stock_ID")\
    .cache()

minute_level_data_df = fact_stock_minute_df.join(top_10_cn_stocks_cached, \
                        fact_stock_minute_df.Stock_ID == top_10_cn_stocks_cached.top10_Stock_ID, "inner") \
    .filter((col("Minute_ID") >= 202311210930) & (col("Minute_ID") <= 202311210940)) \
    .select("FSM_ID", "Stock_ID", "Minute_ID", "Close", "High", "Low", "volume")

minute_level_data_df.show()

top_10_cn_stocks_cached.unpersist()
print(f"Time used: {time.time() - start_time}")
```

✓ 0.6s Python

```
+---------------+--------+-----------+-----+-----+-----+------+
|         FSM_ID|Stock_ID|  Minute_ID|Close| High|  Low|volume|
+---------------+--------+-----------+-----+-----+-----+------+
|601288202311210930|  601288|202311210930| 3.66| 3.66| 3.66| 11861|
|601288202311210931|  601288|202311210931| 3.66| 3.67| 3.65|155799|
|601288202311210932|  601288|202311210932| 3.66| 3.67| 3.65|116492|
|601288202311210933|  601288|202311210933| 3.67| 3.67| 3.66| 11392|
|601288202311210934|  601288|202311210934| 3.66| 3.67| 3.66| 14070|
|601288202311210935|  601288|202311210935| 3.67| 3.67| 3.66| 28420|
|601288202311210936|  601288|202311210936| 3.66| 3.67| 3.66| 14610|
|601288202311210937|  601288|202311210937| 3.67| 3.67| 3.66|  9740|
|601288202311210938|  601288|202311210938| 3.66| 3.67| 3.66|  9401|
|601288202311210939|  601288|202311210939| 3.66| 3.67| 3.66| 10829|
|601288202311210940|  601288|202311210940| 3.67| 3.67| 3.66| 38237|
|601857202311210930|  601857|202311210930| 7.14| 7.14| 7.14|  4878|
|601857202311210931|  601857|202311210931| 7.15| 7.15| 7.12| 23906|
|601857202311210932|  601857|202311210932| 7.12| 7.14| 7.12|  9806|
|601857202311210933|  601857|202311210933| 7.12| 7.13| 7.11|  4412|
|601857202311210934|  601857|202311210934| 7.14| 7.14| 7.12|  5834|
|601857202311210935|  601857|202311210935| 7.15| 7.15| 7.13| 14307|
|601857202311210936|  601857|202311210936| 7.14| 7.15| 7.14|  7101|
|601857202311210937|  601857|202311210937| 7.14| 7.15| 7.14|  2217|
|601857202311210938|  601857|202311210938| 7.15| 7.15| 7.14|  4517|
+---------------+--------+-----------+-----+-----+-----+------+
only showing top 20 rows

Time used: 0.6944880485534668
```

Time Cost: 0.69ms

PostgreSQL Performance Tuning Example1

select Daily Situations for every Monday in November of company SZ300796;

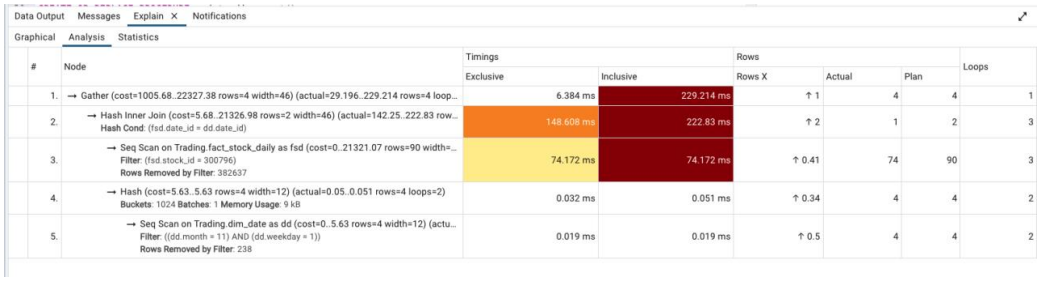
version 1 SELECT dd.Month, dd.WeekDay, fsd.Stock_ID, fsd.Date_ID,

fsd.Open,fsd.High,fsd.Low, fsd.Volume, fsd.Turnover

FROM "Trading".Fact_Stock_Daily fsd

JOIN "Trading".Dim_Date dd ON fsd.Date_ID = dd.Date_ID

WHERE dd.Month = 11 and dd.WeekDay=1 and fsd.Stock_ID = 300796;



Data Output | Messages | Explain | Notifications

Graphical | Analysis | Statistics

#	Node	Timings		Rows			Loops
		Exclusive	Inclusive	Rows X	Actual	Plan	
1.	→ Gather (cost=1005.68..22327.38 rows=4 width=46) (actual=29.196..229.214 rows=4 loop...	6.384 ms	229.214 ms	↑ 1	4	4	1
2.	→ Hash Inner Join (cost=5.68..21326.98 rows=2 width=46) (actual=142.25..222.83 row... Hash Cond: (fsd.date_id = dd.date_id)	148.608 ms	222.83 ms	↑ 2	1	2	3
3.	→ Seq Scan on Trading.fact_stock_daily as fsd (cost=0..21321.07 rows=90 width=... Filter: (fsd.stock_id = 300796) Rows Removed by Filter: 382637	74.172 ms	74.172 ms	↑ 0.41	74	90	3
4.	→ Hash (cost=5.63..5.63 rows=4 width=12) (actual=0.05..0.051 rows=4 loops=2) Buckets: 1024 Batches: 1 Memory Usage: 9 kB	0.032 ms	0.051 ms	↑ 0.34	4	4	2
5.	→ Seq Scan on Trading.dim_date as dd (cost=0..5.63 rows=4 width=12) (actu... Filter: ((dd.month = 11) AND (dd.weekday = 1)) Rows Removed by Filter: 238	0.019 ms	0.019 ms	↑ 0.5	4	4	2

version 2 SELECT dd.Month, dd.WeekDay, fsd.Stock_ID, fsd.Date_ID, fsd.Open,

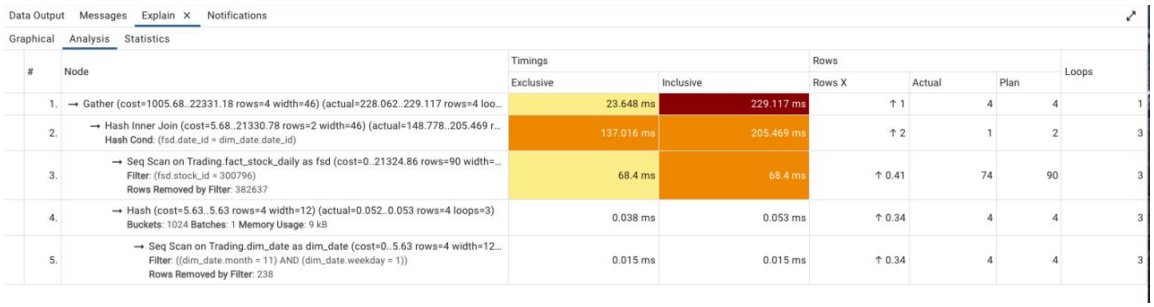
fsd.High,fsd.Low, fsd.Volume, fsd.Turnover

FROM "Trading".Fact_Stock_Daily fsd

JOIN (SELECT Date_ID, Month, WeekDay FROM "Trading".Dim_Date

WHERE Month = 11 AND WeekDay = 1) dd ON fsd.Date_ID = dd.Date_ID

WHERE fsd.Stock_ID = 300796;



Data Output | Messages | Explain | Notifications

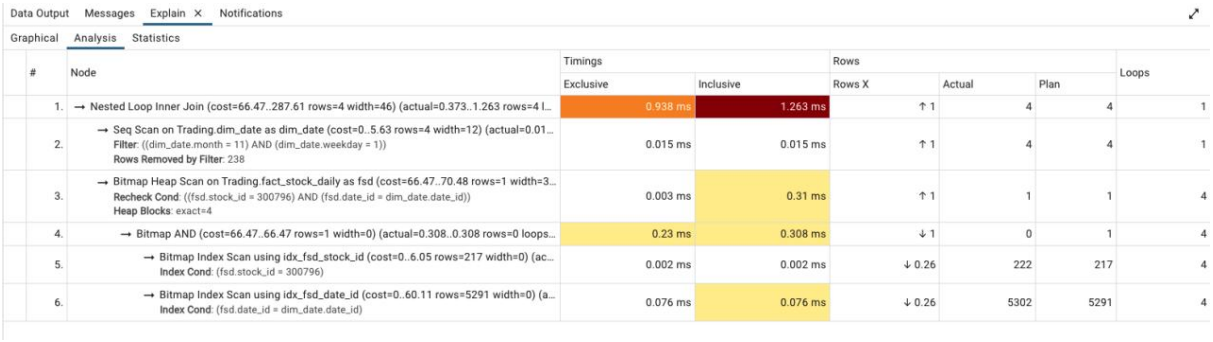
Graphical | Analysis | Statistics

#	Node	Timings		Rows			Loops
		Exclusive	Inclusive	Rows X	Actual	Plan	
1.	→ Gather (cost=1005.68..22331.18 rows=4 width=46) (actual=228.062..229.117 rows=4 loo...	23.648 ms	229.117 ms	↑ 1	4	4	1
2.	→ Hash Inner Join (cost=5.68..21330.78 rows=2 width=46) (actual=148.778..205.469 r... Hash Cond: (fsd.date_id = dim_date.date_id)	137.016 ms	205.469 ms	↑ 2	1	2	3
3.	→ Seq Scan on Trading.fact_stock_daily as fsd (cost=0..21324.86 rows=90 width=... Filter: (fsd.stock_id = 300796) Rows Removed by Filter: 382637	68.4 ms	68.4 ms	↑ 0.41	74	90	3
4.	→ Hash (cost=5.63..5.63 rows=4 width=12) (actual=0.052..0.053 rows=4 loops=3) Buckets: 1024 Batches: 1 Memory Usage: 9 kB	0.038 ms	0.053 ms	↑ 0.34	4	4	3
5.	→ Seq Scan on Trading.dim_date as dim_date (cost=0..5.63 rows=4 width=12... Filter: ((dim_date.month = 11) AND (dim_date.weekday = 1)) Rows Removed by Filter: 238	0.015 ms	0.015 ms	↑ 0.34	4	4	3

version 3 Add index on FK

CREATE INDEX idx_fsd_stock_id ON "Trading".Fact_Stock_Daily(Stock_ID);

CREATE INDEX idx_fsd_date_id ON "Trading".Fact_Stock_Daily(Date_ID);



Data Output | Messages | Explain | Notifications

Graphical | Analysis | Statistics

#	Node	Timings		Rows			Loops
		Exclusive	Inclusive	Rows X	Actual	Plan	
1.	→ Nested Loop Inner Join (cost=66.47..287.61 rows=4 width=46) (actual=0.373..1.263 rows=4 l...	0.938 ms	1.263 ms	↑ 1	4	4	1
2.	→ Seq Scan on Trading.dim_date as dim_date (cost=0..5.63 rows=4 width=12) (actual=0.01... Filter: ((dim_date.month = 11) AND (dim_date.weekday = 1)) Rows Removed by Filter: 238	0.015 ms	0.015 ms	↑ 1	4	4	1
3.	→ Bitmap Heap Scan on Trading.fact_stock_daily as fsd (cost=66.47..70.48 rows=1 width=3... Recheck Cond: ((fsd.stock_id = 300796) AND (fsd.date_id = dim_date.date_id)) Heap Blocks: exact=4	0.003 ms	0.31 ms	↑ 1	1	1	4
4.	→ Bitmap AND (cost=66.47..66.47 rows=1 width=0) (actual=0.308..0.308 rows=0 loops...	0.23 ms	0.308 ms	↓ 1	0	1	4
5.	→ Bitmap Index Scan using idx_fsd_stock_id (cost=0..6.05 rows=217 width=0) (ac... Index Cond: (fsd.stock_id = 300796)	0.002 ms	0.002 ms	↓ 0.26	222	217	4
6.	→ Bitmap Index Scan using idx_fsd_date_id (cost=0..60.11 rows=5291 width=0) (a... Index Cond: (fsd.date_id = dim_date.date_id)	0.076 ms	0.076 ms	↓ 0.26	5302	5291	4

Performance tuning skills:

Use subquery and Add index on FKs

PostgreSQL Performance Tuning Example2

Query: In event, we first filter out the latest events from 2023–11–20 to 2023–11–24, and from these events, we check the stock price changes of the corresponding companies on the next day, and from the companies that have increased by more than 3%, we categorize the companies according to the time and the type of the event, and finally, we sort them according to the date, and the average growth of the stock price.

Version 1:

```
with event_stock as(

select distinct ds.stock_id, fce.Date_ID, de.Event_Type

from "Trading".Fact_Company_Event fce

join "Trading".dim_event de on fce.Event_Dim_ID = de.Event_Dim_ID

join "Trading".Dim_Stock ds on ds.Stock_ID = fce.Stock_ID
```

```
where de.Expire_Timestamp isnull

and fce.Date_ID between 20231120 and 20231124

order by fce.Date_ID)

select distinct tmp.event_type, tmp.date_id, tmp.avg_incre

from ( select

fsd.stock_id, fsd.date_id, es.event_type,fsd.percent_of_incre_decre,

AVG(fsd.percent_of_incre_decre) OVER(PARTITION BY es.event_type) avg_incre

        from "Trading".Fact_Stock_Daily as fsd

        join event_stock es on  fsd.Stock_ID = es.stock_id and

        fsd.Date_ID = es.Date_ID + 1 and fsd.percent_of_incre_decre>3.0

group by fsd.stock_id,fsd.date_id, es.event_type,fsd.percent_of_incre_decre

) AS tmp

group by date_id,event_type,tmp.avg_incre

order by tmp.date_id,tmp.avg_incre
```

#	Node	Exclusive	Inclusive
1.	→ Unique (cost=5661.31..5661.45 rows=8 width=54) (actual=227.707..227.8…	0.003 ms	227.889 ms
2.	→ Group (cost=5661.31..5661.39 rows=8 width=54) (actual=227.706..2…	0.004 ms	227.887 ms
3.	→ Sort (cost=5661.31..5661.33 rows=8 width=54) (actual=227.706…	0.01 ms	227.884 ms
4.	→ Subquery Scan (cost=5660.97..5661.19 rows=8 width=54) (…	0.002 ms	227.874 ms
5.	→ Window Aggregate (cost=5660.97..5661.11 rows=8 wi…	0.176 ms	227.873 ms
6.	→ Sort (cost=5660.97..5660.99 rows=8 width=32) (a…	0.581 ms	227.697 ms
7.	→ Group (cost=5660.75..5660.85 rows=8 width=…	0.007 ms	227.116 ms

Version2

```
CREATE INDEX IF NOT EXISTS idx_fsm_stock_id ON "Trading".Fact_Stock_Minute(Stock_ID);

CREATE INDEX IF NOT EXISTS idx_fsm_minute_id ON "Trading".Fact_Stock_Minute(Minute_ID);

CREATE INDEX IF NOT EXISTS idx_fce_stock_id ON "Trading".Fact_Company_Event(Stock_ID);

CREATE INDEX IF NOT EXISTS idx_fce_date_id ON "Trading".Fact_Company_Event(Date_ID);

WITH event_stock AS (
   SELECT distinct ds.stock_id, fce.Date_ID, de.Event_Type
   FROM "Trading".Fact_Company_Event fce
   JOIN "Trading".dim_event de ON fce.Event_Dim_ID = de.Event_Dim_ID
   JOIN "Trading".Dim_Stock ds ON ds.Stock_ID = fce.Stock_ID
   WHERE de.Expire_Timestamp isnull
   AND fce.Date_ID between 20231120 and 20231124
),
tmp AS (
   SELECT
      fsd.stock_id, fsd.date_id, es.event_type,
      AVG(fsd.percent_of_incre_decre) OVER(PARTITION BY es.event_type) avg_incre
   FROM "Trading".Fact_Stock_Daily as fsd
```

```
  JOIN event_stock es ON
    fsd.Stock_ID = es.stock_id and
    fsd.Date_ID = es.Date_ID + 1 and
    fsd.percent_of_incre_decre>3.0
  GROUP BY fsd.stock_id,fsd.date_id, es.event_type,fsd.percent_of_incre_decre
)
SELECT distinct
tmp.event_type,
tmp.date_id,
tmp.avg_incre
FROM tmp
GROUP BY date_id,event_type,tmp.avg_incre
ORDER BY tmp.date_id,tmp.avg_incre
```

		Exclusive	Inclusive	R
1.	→ Unique (cost=5662.26..5662.4 rows=8 width=54) (actual=62.855..62.889 r...	0.003 ms	62.889 ms	
2.	→ Group (cost=5662.26..5662.34 rows=8 width=54) (actual=62.855..62....	0.004 ms	62.887 ms	
3.	→ Sort (cost=5662.26..5662.28 rows=8 width=54) (actual=62.854.....	0.016 ms	62.883 ms	
4.	→ Subquery Scan (cost=5661.92..5662.14 rows=8 width=54) (...	0.004 ms	62.868 ms	
5.	→ Window Aggregate (cost=5661.92..5662.06 rows=8 wi...	0.304 ms	62.865 ms	
6.	→ Sort (cost=5661.92..5661.94 rows=8 width=32) (a...	0.525 ms	62.562 ms	
7.	→ Group (cost=1686.93..5661.8 rows=8 width=3...	0.006 ms	62.037 ms	
8.	→ Incremental Sort (cost=1686.93..5661.7...	0.029 ms	62.032 ms	

Explanation:

First of all, add index to all foreign keys, and put all fk's into block buffer cache to improve cache hit ratio. meanwhile, adding the attribute of index can avoid full table scan when joining.

Use the WITH clause to create a temporary table instead of duplicating the event_stock and Fact_Stock_Daily tables in the SELECT clause. This avoids double counting and improves query speed and makes the size of the index scan during a join operation significantly smaller

Instead of using a global sort, I use the PARTITION BY clause to group data by date or other dimension. This reduces the overhead of sorting operations and improves query efficiency.

Instead of using multiple subqueries, I use the AVG() function to calculate the average increment for each date. This simplifies the query logic and improves query performance.