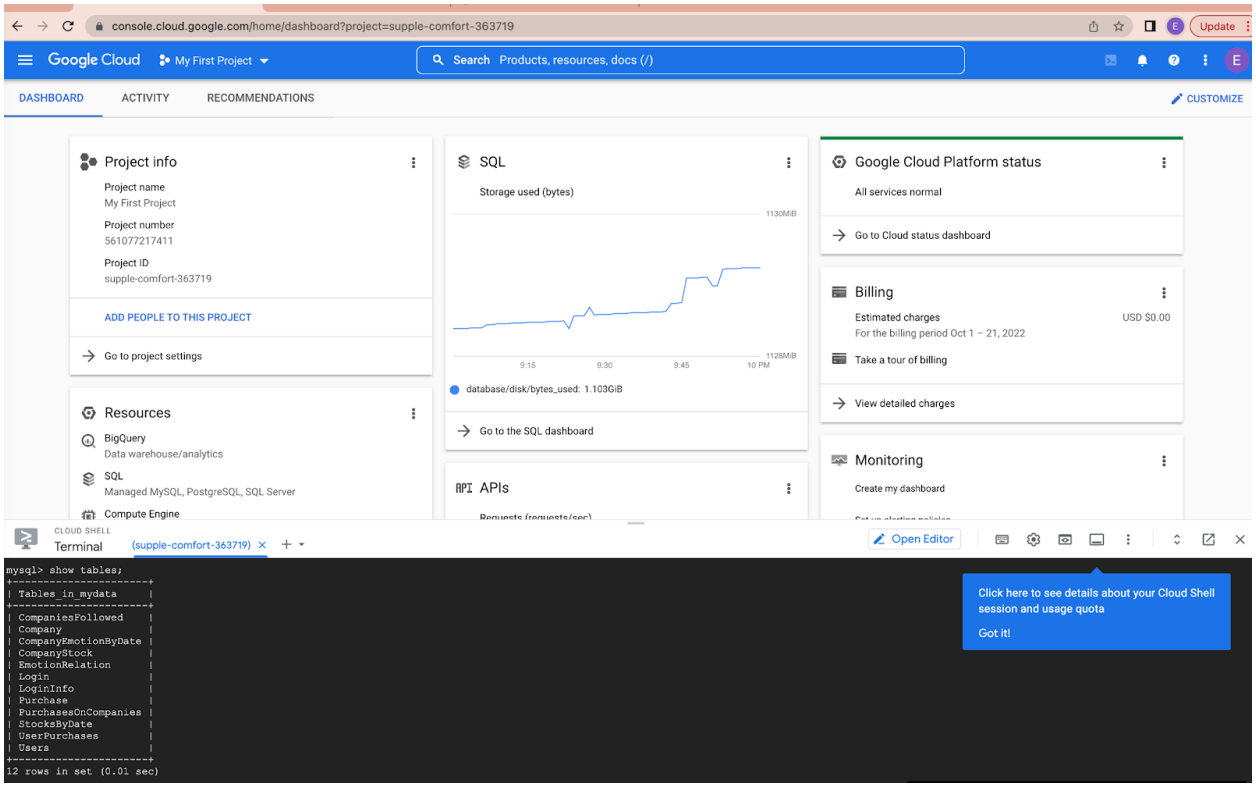
Stage3: Database Design

Database Implementation:



Entities:

```
CREATE TABLE Company(
        CompanyID VARCHAR(255) Primary Key,
        CompanyName VARCHAR(255),
        Followers INT,
        CurrentStockPrice REAL,
        CurrentGrowthRate REAL,
        PredictedStockPrice REAL,
        PredictedGrowthRate REAL
);


CREATE TABLE CompanyEmotionByDate (
        CompanyID varchar(255),
        Date varchar(255),
        Emotion varchar(255),
        Frequency REAL,
```

```
	Primary Key (CompanyId, Date, Emotion)
);

CREATE TABLE LoginInfo (
	UserID VARCHAR(255) Primary Key,
	PassWord_ VARCHAR(50),
	PhoneNumber LONG,
	Email VARCHAR(50)
);

CREATE TABLE Purchase (
	PurchaseId varchar(255) Primary Key,
	UserId varchar(255),
	CompanyID varchar(255),
	Date varchar(255),
	ChangeInNumberOfStocks REAL,
	ChangeInPrice REAL
);

CREATE TABLE  StocksByDate(
	StockId varchar(255) Primary Key,
	CompanyID varchar(255),
	Date varchar(255),
	StockPrice REAL,
	GrowthRate REAL
);

CREATE TABLE Users (
	UserId VARCHAR(255) Primary Key,
	AccountName VARCHAR(50)
);
```

Relations:

```
CREATE TABLE CompaniesFollowed (
	UserId varchar(255) references Users(UserId) ON DELETE CASCADE ON UPDATE
CASCADE,
	CompanyId varchar(255) references Company(CompanyID) ON DELETE CASCADE
ON UPDATE CASCADE,
	PRIMARY KEY (UserId, CompanyId)
);

CREATE TABLE Login(
```

```
	UserId varchar(255) references LoginIfo(UserId) ON DELETE CASCADE ON UPDATE
CASCADE,
	UserID varchar(255) references Users(UserId) ON DELETE CASCADE ON UPDATE
CASCADE
);

CREATE TABLE CompanyStock (
	StockId varchar(255) references StocksByDate(StockId) ON DELETE CASCADE ON
UPDATE CASCADE,
	CompanyId varchar(255) references Company(CompanyId) ON DELETE CASCADE ON
UPDATE CASCADE,
   PRIMARY KEY (StockId, CompanyId)
);

CREATE TABLE UserPurchases (
	UserId varchar(255) references Users(UserId) ON DELETE CASCADE ON UPDATE
CASCADE,
	PurchaseId varchar(255) references Purchase(PurchaseId) ON DELETE CASCADE ON
UPDATE CASCADE,
	PRIMARY KEY (UserId, PurchaseId)
);

CREATE TABLE PurchasesOnCompanies (
	CompanyId varchar(255) references Company(CompanyId) ON DELETE CASCADE ON
UPDATE CASCADE,
	PurchaseId varchar(255) references Purchase(PurchaseId) ON DELETE CASCADE ON
UPDATE CASCADE,
   PRIMARY KEY (CompanyId, PurchaseId)

);

Create TABLE EmotionRelation (
	Id1 varchar(50) references Company(CompanyId) ON DELETE CASCADE ON UPDATE
CASCADE,
	Id2 varchar(50) references CompanyEmotionsByDate(CompanyId) ON DELETE
CASCADE ON UPDATE CASCADE,
	Date varchar(255) references CompanyEmotionsByDate(Date) ON DELETE CASCADE
ON UPDATE CASCADE,
	Emotion varchar(255) references CompanyEmotionsByDate(Emotion) ON DELETE
CASCADE ON UPDATE CASCADE,
	Primary Key(Id1,Id2, Date, Emotion)
)
```

Three Tables Having >1000 Rows

```
SELECT COUNT(PurchaseId)
FROM Purchase;
```

COUNT(PurchaseId)
1001

```
SELECT COUNT(StockId)
FROM StocksByDate;
```

COUNT(StockId)
1328

```
Use mydata;
SELECT COUNT(CompanyId)
FROM CompanyEmotionByDate;
```

COUNT(CompanyId)
103416

Queries:

1. Find the top 15 users holding the highest total worth of all stock under the current price.

```
USE mydata;
SELECT UserId, sum FROM
(SELECT sum(stock_change*StockPrice) sum, UserId FROM
(SELECT ChangeInNumberOfStocks as stock_change, UserId, CompanyID FROM Purchase) a
NATURAL JOIN StocksByDate GROUP BY UserId) b
ORDER BY sum DESC, UserId
LIMIT 15;
```

UserId	sum
2040895809	13514887.587600002
9417629999	13514887.587600002
6222864198	5317146.281600001
2226872272	3139505.882
1136498583	1349732.7684
4863664559	1349732.7684
4889627251	1349732.7684
6285202079	1349732.7684
9691781697	1349732.7684
9945055399	751845.0681
791940187	346708.99240000005
9881632838	335994.3101
3023397821	278369.5333
8655517933	273131.9905
3641874726	193564.80159999998

2. Find the top 15 companies whose stocks were sold for the highest total values, and find out how many followers these companies have.

```
USE mydata;
SELECT c.CompanyName, t.sum_, c.Followers
FROM (SELECT SUM(pur.ChangeInPrice) as sum_, pur.CompanyID
 FROM Purchase pur
    GROUP BY CompanyID) as t NATURAL JOIN Company c
WHERE (t.sum_)> 100000
ORDER BY t.sum_ DESC, c.Followers DESC
LIMIT 15;
```

100% 10:8

Result Grid Filter Rows: Search Export: Fetch rows:

CompanyName	sum_	Followers
LifeStance Health Group Inc. Common Stock	235133	24
McKesson Corporation Common Stock	234882	35
Lamar Advertising Company Class A Common...	182759	37
Church & Dwight Company Inc. Common Stock	178811	76
L3Harris Technologies Inc. Common Stock	163319	85
Quaker Houghton Common Stock	160096	96
Mid-America Apartment Communities Inc. Com...	159996	61
Medpace Holdings Inc. Common Stock	148855	50
Laboratory Corporation of America Holdings Co...	146060	70
Moody's Corporation Common Stock	138339	90
McGrath RentCorp Common Stock	134648	80
Matson Inc. Common Stock	132830	50
Saia Inc. Common Stock	132145	58
Masimo Corporation Common Stock	126069	51
Southwest Airlines Company Common Stock	117524	74

Indexing:

Query 1:
When there is no index, the aggregate cost time was 3.970s, and the scan cost time for StocksByDate and Purchase tables were 0.729s and 0.528s. But when we applied index by both UserId and CompanyId, the aggregate cost time became 3.970s, and the scan cost time for StocksByDate and Purchase tables were 0.481s and 0.441s. The cost time for table scans are halved. Since each userId may purchase a variety of companies' stock and each company may also have many followers, when we use index with these two attributes, we can reduce the cost time for searching.

1. EXPLAIN ANALYZE with no index

```
-> Limit: 15 row(s)  (cost=0.00..0.00 rows=0) (actual time=4.659..4.662 rows=15 loops=1)
    -> Sort: b.sum DESC, b.UserId, limit input to 15 row(s) per chunk  (actual time=0.378..0.379
rows=15 loops=1)
        -> Table scan on b  (cost=14957.35 rows=132932) (actual time=0.001..0.092 rows=888
loops=1)
            -> Materialize  (cost=0.00..0.00 rows=0) (actual time=4.658..4.661 rows=888 loops=1)
                -> Table scan on <temporary>  (actual time=0.001..0.093 rows=888 loops=1)
                    -> Aggregate using temporary table  (actual time=3.803..3.970 rows=888 loops=1)
                        -> Filter: (StocksByDate.CompanyID = Purchase.CompanyID)  (cost=133055.04
rows=132933) (actual time=1.046..2.601 rows=1001 loops=1)
                            -> Inner hash join
(<hash>(StocksByDate.CompanyID)=<hash>(Purchase.CompanyID))  (cost=133055.04
rows=132933) (actual time=1.042..2.285 rows=1001 loops=1)
                                -> Table scan on StocksByDate  (cost=0.03 rows=1328) (actual
time=0.046..0.729 rows=1328 loops=1)
                                -> Hash
                                    -> Table scan on Purchase  (cost=102.35 rows=1001) (actual
time=0.050..0.528 rows=1001 loops=1)
```

2. CREATE INDEX user_id ON Purchase (UserId);

```
-> Limit: 15 row(s)  (cost=0.00..0.00 rows=0) (actual time=3.853..3.855 rows=15 loops=1)
    -> Sort: b.sum DESC, b.UserId, limit input to 15 row(s) per chunk  (actual time=0.260..0.261
rows=15 loops=1)
        -> Table scan on b  (cost=14957.35 rows=132932) (actual time=0.001..0.067 rows=888
loops=1)
            -> Materialize  (cost=0.00..0.00 rows=0) (actual time=3.851..3.853 rows=888 loops=1)
                -> Table scan on <temporary>  (actual time=0.001..0.083 rows=888 loops=1)
                    -> Aggregate using temporary table  (actual time=3.085..3.266 rows=888 loops=1)
                        -> Filter: (StocksByDate.CompanyID = Purchase.CompanyID)  (cost=133055.04
rows=132933) (actual time=1.034..2.138 rows=1001 loops=1)
                            -> Inner hash join
(<hash>(StocksByDate.CompanyID)=<hash>(Purchase.CompanyID))  (cost=133055.04
rows=132933) (actual time=1.030..1.912 rows=1001 loops=1)
                                -> Table scan on StocksByDate  (cost=0.03 rows=1328) (actual
time=0.069..0.516 rows=1328 loops=1)
                                -> Hash
                                    -> Table scan on Purchase  (cost=102.35 rows=1001) (actual
time=0.037..0.428 rows=1001 loops=1)
```

3. CREATE INDEX change_in_num ON Purchase (ChangeInNumberOfStocks);

-> Limit: 15 row(s) (cost=0.00..0.00 rows=0) (actual time=6.234..6.237 rows=15 loops=1)
    -> Sort: b.sum DESC, b.UserId, limit input to 15 row(s) per chunk (actual time=0.488..0.489 rows=15 loops=1)
        -> Table scan on b (cost=14957.35 rows=132932) (actual time=0.001..0.080 rows=888 loops=1)
            -> Materialize (cost=0.00..0.00 rows=0) (actual time=6.234..6.235 rows=888 loops=1)
                -> Table scan on <temporary> (actual time=0.001..0.071 rows=888 loops=1)
                    -> Aggregate using temporary table (actual time=5.027..5.156 rows=888 loops=1)
                        -> Filter: (StocksByDate.CompanyID = Purchase.CompanyID) (cost=133055.04 rows=132933) (actual time=2.241..3.607 rows=1001 loops=1)
                            -> Inner hash join (<hash>(StocksByDate.CompanyID)=<hash>(Purchase.CompanyID)) (cost=133055.04 rows=132933) (actual time=2.237..3.399 rows=1001 loops=1)
                                -> Table scan on StocksByDate (cost=0.03 rows=1328) (actual time=0.703..1.491 rows=1328 loops=1)
                                -> Hash
                                    -> Table scan on Purchase (cost=102.35 rows=1001) (actual time=0.550..0.907 rows=1001 loops=1)

4. CREATE INDEX company_id ON Purchase (CompanyID);
-> Limit: 15 row(s) (cost=0.00..0.00 rows=0) (actual time=16.462..16.467 rows=15 loops=1)
    -> Sort: b.sum DESC, b.UserId, limit input to 15 row(s) per chunk (actual time=0.610..0.612 rows=15 loops=1)
        -> Table scan on b (cost=312.66 rows=2757) (actual time=0.002..0.078 rows=888 loops=1)
            -> Materialize (cost=0.00..0.00 rows=0) (actual time=16.461..16.465 rows=888 loops=1)
                -> Table scan on <temporary> (actual time=0.001..0.080 rows=888 loops=1)
                    -> Aggregate using temporary table (actual time=15.439..15.573 rows=888 loops=1)
                        -> Nested loop inner join (cost=1100.33 rows=2758) (actual time=0.129..14.310 rows=1001 loops=1)
                            -> Filter: (StocksByDate.CompanyID is not null) (cost=135.05 rows=1328) (actual time=0.045..0.710 rows=1328 loops=1)
                                -> Table scan on StocksByDate (cost=135.05 rows=1328) (actual time=0.044..0.592 rows=1328 loops=1)
                            -> Index lookup on Purchase using company_id (CompanyID=StocksByDate.CompanyID) (cost=0.52 rows=2) (actual time=0.010..0.010 rows=1 loops=1328)

5. CREATE INDEX company_id ON Purchase (CompanyID);
-> Limit: 15 row(s) (cost=0.00..0.00 rows=0) (actual time=3.755..3.757 rows=15 loops=1)
    -> Sort: b.sum DESC, b.UserId, limit input to 15 row(s) per chunk (actual time=0.270..0.271 rows=15 loops=1)

```
    -> Table scan on b  (cost=14957.35 rows=132932) (actual time=0.001..0.064 rows=888
loops=1)
        -> Materialize  (cost=0.00..0.00 rows=0) (actual time=3.754..3.755 rows=888 loops=1)
            -> Table scan on <temporary>  (actual time=0.001..0.080 rows=888 loops=1)
                -> Aggregate using temporary table  (actual time=3.127..3.261 rows=888 loops=1)
                    -> Filter: (StocksByDate.CompanyID = Purchase.CompanyID)  (cost=133055.04
rows=132933) (actual time=1.419..2.359 rows=1001 loops=1)
                        -> Inner hash join
(<hash>(StocksByDate.CompanyID)=<hash>(Purchase.CompanyID))  (cost=133055.04
rows=132933) (actual time=1.415..2.161 rows=1001 loops=1)
                            -> Table scan on StocksByDate  (cost=0.03 rows=1328) (actual
time=0.546..0.959 rows=1328 loops=1)
                            -> Hash
                                -> Table scan on Purchase  (cost=102.35 rows=1001) (actual
time=0.047..0.385 rows=1001 loops=1)
```

6. CREATE INDEX user_company_id ON Purchase (UserId, CompanyID);

```
-> Limit: 15 row(s)  (cost=0.00..0.00 rows=0) (actual time=3.518..3.520 rows=15 loops=1)
    -> Sort: b.sum DESC, b.UserId, limit input to 15 row(s) per chunk  (actual time=0.244..0.244
rows=15 loops=1)
    -> Table scan on b  (cost=14957.35 rows=132932) (actual time=0.001..0.069 rows=888
loops=1)
        -> Materialize  (cost=0.00..0.00 rows=0) (actual time=3.517..3.519 rows=888 loops=1)
            -> Table scan on <temporary>  (actual time=0.001..0.081 rows=888 loops=1)
                -> Aggregate using temporary table  (actual time=2.869..3.033 rows=888 loops=1)
                    -> Filter: (StocksByDate.CompanyID = Purchase.CompanyID)  (cost=133055.04
rows=132933) (actual time=0.994..2.034 rows=1001 loops=1)
                        -> Inner hash join
(<hash>(StocksByDate.CompanyID)=<hash>(Purchase.CompanyID))  (cost=133055.04
rows=132933) (actual time=0.991..1.820 rows=1001 loops=1)
                            -> Table scan on StocksByDate  (cost=0.03 rows=1328) (actual
time=0.031..0.481 rows=1328 loops=1)
                            -> Hash
                                -> Table scan on Purchase  (cost=102.35 rows=1001) (actual
time=0.055..0.441 rows=1001 loops=1)
```

For query 2:

When there is no index, the aggregate cost is 2.100, and the table scan on the subquery, temp table and Purchase takes 0.003, 0.039 and 0.399, respectively. When we add an index on Companies (Followers), the aggregate cost decreases to 1.349, but the table scans would take a little longer. This is probably because Followers is an auxiliary attribute of the table and indexing on it does not help much. Then we add an index on Company (CompanyId), which also speeds up the aggregate time. The cost for table scans is similar and shows no significant improvement, probably because our current Purchase table is relatively small. After that, we try adding an index on PurchaseId but the situation is also similar. Finally, we try adding combinations of any two indexes out of the three indexes(Followers, CompanyId, PurchaseId) , but it doesn't bring a better effect either. Since neither index or combination of indices reduced the cost, we decided to not include an index for this query. Since there isn't any repetition among our data, specifically on followers, companyId, and PurchaseId, it wouldn't be necessary to index such that the cost would be reduced.

1. EXPLAIN ANALYZE with no index

```
-> Limit: 15 row(s)  (actual time=2.366..2.368 rows=15 loops=1)
    -> Sort: t.sum_ DESC, c.Followers DESC, limit input to 15 row(s) per chunk  (actual time=2.365..2.367 rows=15 loops=1)
    -> Stream results  (cost=465.46 rows=1001) (actual time=2.209..2.336 rows=23 loops=1)
    -> Nested loop inner join  (cost=465.46 rows=1001) (actual time=2.207..2.327 rows=23 loops=1)
        -> Filter: (t.CompanyID is not null)  (cost=0.11..115.11 rows=1001) (actual time=2.154..2.160 rows=23 loops=1)
        -> Table scan on t  (cost=2.50..2.50 rows=0) (actual time=0.001..0.003 rows=23 loops=1)
        -> Materialize  (cost=2.50..2.50 rows=0) (actual time=2.153..2.157 rows=23 loops=1)
            -> Filter: (sum(pur.ChangeInPrice) > 100000)  (actual time=2.041..2.132 rows=23 loops=1)
            -> Table scan on <temporary>  (actual time=0.001..0.039 rows=482 loops=1)
            -> Aggregate using temporary table  (actual time=2.035..2.100 rows=482 loops=1)
                -> Table scan on pur  (cost=102.35 rows=1001) (actual time=0.646..1.399 rows=1001 loops=1)
        -> Single-row index lookup on c using PRIMARY (CompanyID=t.CompanyID) (cost=0.25 rows=1) (actual time=0.007..0.007 rows=1 loops=23)
```

2. Create index f on Companies (Followers)

```
-> Limit: 15 row(s)  (actual time=1.736..1.738 rows=15 loops=1)
    -> Sort: t.sum_ DESC, c.Followers DESC, limit input to 15 row(s) per chunk  (actual time=1.735..1.737 rows=15 loops=1)
    -> Stream results  (cost=465.46 rows=1001) (actual time=1.562..1.705 rows=23 loops=1)
```

-> Nested loop inner join (cost=465.46 rows=1001) (actual time=1.559..1.692 rows=23 loops=1)
-> Filter: (t.CompanyID is not null) (cost=0.11..115.11 rows=1001) (actual time=1.530..1.537 rows=23 loops=1)
-> Table scan on t (cost=2.50..2.50 rows=0) (actual time=0.001..0.005 rows=23 loops=1)
-> Materialize (cost=2.50..2.50 rows=0) (actual time=1.528..1.533 rows=23 loops=1)
-> Filter: (sum(pur.ChangeInPrice) > 100000) (actual time=1.265..1.502 rows=23 loops=1)
-> Table scan on <temporary> (actual time=0.001..0.058 rows=482 loops=1)
-> Aggregate using temporary table (actual time=1.258..1.349 rows=482 loops=1)
-> Table scan on pur (cost=102.35 rows=1001) (actual time=0.063..0.449 rows=1001 loops=1)
-> Single-row index lookup on c using PRIMARY (CompanyID=t.CompanyID) (cost=0.25 rows=1) (actual time=0.006..0.006 rows=1 loops=23)

3. Create index cid on Company (CompanyId):

-> Limit: 15 row(s) (actual time=2.657..2.659 rows=15 loops=1)
-> Sort: t.sum_ DESC, c.Followers DESC, limit input to 15 row(s) per chunk (actual time=2.657..2.658 rows=15 loops=1)
-> Stream results (cost=465.46 rows=1001) (actual time=1.304..2.627 rows=23 loops=1)
-> Nested loop inner join (cost=465.46 rows=1001) (actual time=1.301..2.613 rows=23 loops=1)
-> Filter: (t.CompanyID is not null) (cost=0.11..115.11 rows=1001) (actual time=1.273..1.284 rows=23 loops=1)
-> Table scan on t (cost=2.50..2.50 rows=0) (actual time=0.000..0.006 rows=23 loops=1)
-> Materialize (cost=2.50..2.50 rows=0) (actual time=1.272..1.279 rows=23 loops=1)
-> Filter: (sum(pur.ChangeInPrice) > 100000) (actual time=1.156..1.251 rows=23 loops=1)
-> Table scan on <temporary> (actual time=0.001..0.036 rows=482 loops=1)
-> Aggregate using temporary table (actual time=1.150..1.212 rows=482 loops=1)
-> Table scan on pur (cost=102.35 rows=1001) (actual time=0.060..0.477 rows=1001 loops=1)
-> Single-row index lookup on c using PRIMARY (CompanyID=t.CompanyID) (cost=0.25 rows=1) (actual time=0.057..0.057 rows=1 loops=23)

4. Create index purid on Purchase (PurchaseId)

-> Limit: 15 row(s) (actual time=6.441..6.444 rows=15 loops=1)

```
    -> Sort: t.sum_ DESC, c.Followers DESC, limit input to 15 row(s) per chunk  (actual time=6.441..6.443 rows=15 loops=1)
    -> Stream results  (cost=465.46 rows=1001) (actual time=1.583..6.397 rows=23 loops=1)
    -> Nested loop inner join  (cost=465.46 rows=1001) (actual time=1.579..6.376 rows=23 loops=1)
        -> Filter: (t.CompanyID is not null)  (cost=0.11..115.11 rows=1001) (actual time=1.547..1.563 rows=23 loops=1)
        -> Table scan on t  (cost=2.50..2.50 rows=0) (actual time=0.001..0.010 rows=23 loops=1)
        -> Materialize  (cost=2.50..2.50 rows=0) (actual time=1.545..1.556 rows=23 loops=1)
            -> Filter: (sum(pur.ChangeInPrice) > 100000)  (actual time=1.422..1.521 rows=23 loops=1)
            -> Table scan on <temporary>  (actual time=0.001..0.045 rows=482 loops=1)
            -> Aggregate using temporary table  (actual time=1.413..1.484 rows=482 loops=1)
                -> Table scan on pur  (cost=102.35 rows=1001) (actual time=0.067..0.494 rows=1001 loops=1)
        -> Single-row index lookup on c using PRIMARY (CompanyID=t.CompanyID)  (cost=0.25 rows=1) (actual time=0.209..0.209 rows=1 loops=23)
```