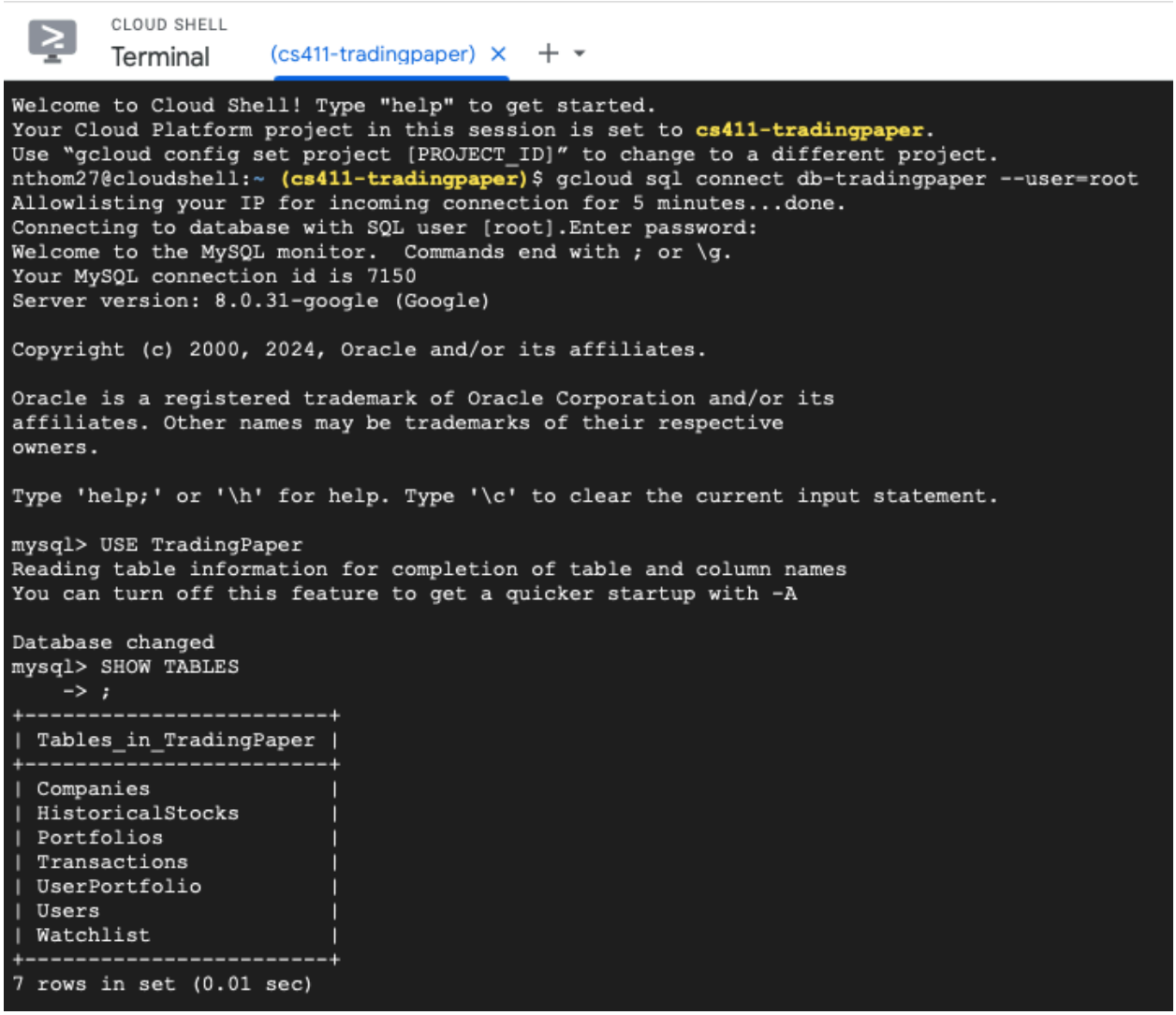
Database Implementation

Proof of Implementation on GCP



DDL Commands + COUNT(*)

CREATE TABLE Users(
UserID VARCHAR(255) NOT NULL,
Password VARCHAR(255) DEFAULT NULL,
FullName VARCHAR(255) DEFAULT NULL,
Address VARCHAR(255) DEFAULT NULL,
PhoneNumber VARCHAR(255) DEFAULT NULL,

```
        PRIMARY KEY (UserID)
);
```

```
mysql> select count(*) from Users
    -> ;
+----------+
| count(*) |
+----------+
|    10000 |
+----------+
1 row in set (0.35 sec)
```
*Table has at least 1000 rows

```
CREATE TABLE Portfolios(
        PortfolioID INT NOT NULL AUTO_INCREMENT,
        PortfolioType VARCHAR(255) DEFAULT NULL,
        PortfolioBalance DECIMAL(9,2) DEFAULT NULL,
        PRIMARY KEY (PortfolioID)
);
```

```
mysql> select count(*) from Portfolios;
+----------+
| count(*) |
+----------+
|    20044 |
+----------+
1 row in set (0.24 sec)
```
*Table has at least 1000 rows

**********<Junction Table>**********

```
CREATE TABLE UserPortfolio(
        UsersUserID VARCHAR(255) NOT NULL,
        PortfoliosPortfolioID INT NOT NULL,
        PRIMARY KEY(UsersUserID, PortfoliosPortfolioID),
        FOREIGN KEY (UsersUserID) REFERENCES Users(UserID)
                ON DELETE CASCADE,
        FOREIGN KEY (PortfoliosPortfolioID) REFERENCES Portfolios(PortfolioID)
                ON DELETE CASCADE
);
```

```
mysql> select count(*) from UserPortfolio;
+----------+
| count(*) |
+----------+
|    20044 |
+----------+
1 row in set (0.36 sec)
```

```
CREATE TABLE Transactions(
        TransactionID INT NOT NULL AUTO_INCREMENT,
```

```
    CurrentlyActive TINYINT(1) DEFAULT NULL,
    StockSymbol VARCHAR(10) DEFAULT NULL,
    NumShares INT DEFAULT NULL,
    PurchasePrice DECIMAL(9,2) DEFAULT NULL,
    `DateTime` DATETIME DEFAULT NULL,
    PortfoliosPortfolioID INT NOT NULL,
    PRIMARY KEY(TransactionID),
    FOREIGN KEY(PortfoliosPortfolioID) REFERENCES Portfolios(PortfolioID)
        ON DELETE CASCADE
);
```

```
mysql> select count(*) from Transactions;
+----------+
| count(*) |
+----------+
|   104005 |
+----------+
1 row in set (0.66 sec)
```

*Table has at least 1000 rows

```
CREATE TABLE Companies(
    StockSymbol VARCHAR(10) NOT NULL,
    Name VARCHAR(255) DEFAULT NULL,
    Sector VARCHAR(255) DEFAULT NULL,
    Industry VARCHAR(255) DEFAULT NULL,
    MarketCap DECIMAL(20,2) DEFAULT NULL,
    RevenueGrowth DECIMAL(14,2) DEFAULT NULL,
    City VARCHAR(255) DEFAULT NULL,
    State VARCHAR(255) DEFAULT NULL,
    Country VARCHAR(255) DEFAULT NULL,
    PRIMARY KEY(StockSymbol)
);
```

```
mysql> select count(*) from Companies;
+----------+
| count(*) |
+----------+
|      503 |
+----------+
1 row in set (0.18 sec)
```

**********<Junction Table>**********

```
CREATE TABLE Watchlist(
    StockSymbol VARCHAR(10) NOT NULL,
    PortfolioID INT NOT NULL,
```

```
    PRIMARY KEY (StockSymbol, PortfolioID),
    FOREIGN KEY (StockSymbol) REFERENCES Companies(StockSymbol)
        ON DELETE CASCADE,
    FOREIGN KEY (PortfolioID) REFERENCES Portfolios(PortfolioID)
        ON DELETE CASCADE
);
```

```
mysql> select count(*) from Watchlist;
+----------+
| count(*) |
+----------+
|    80335 |
+----------+
1 row in set (0.58 sec)
```

```
CREATE TABLE HistoricalStocks(
    `Date` DATE NOT NULL,
    ClosePrice DECIMAL(14,2) DEFAULT NULL,
    StockSymbol VARCHAR(10) NOT NULL,
    PRIMARY KEY(Date, StockSymbol),
    FOREIGN KEY (StockSymbol) REFERENCES Companies(StockSymbol)
        ON DELETE CASCADE
);
```

```
mysql> select count(*)from HistoricalStocks;
+----------+
| count(*) |
+----------+
|  1744089 |
+----------+
1 row in set (2.63 sec)
```

*Table has at least 1000 rows

Indexing Cost Analysis:

1. High and Low Close Prices on Watchlist Stocks:

Query Syntax and Output:

Uses: Join multiple relations and Aggregation via GROUP BY

```
SELECT u.FullName, w.StockSymbol,
    MAX(hs.ClosePrice) AS HighPrice,
    MIN(hs.ClosePrice) AS LowPrice
FROM Users u
JOIN UserPortfolio up ON u.UserID = up.UsersUserID
JOIN Watchlist w ON up.PortfoliosPortfolioID = w.PortfolioID
JOIN HistoricalStocks hs ON w.StockSymbol = hs.StockSymbol
WHERE u.UserID = 'JOHNWALTERS' OR u.UserID = 'ADAM82'
GROUP BY u.UserID, w.StockSymbol;
```

```
mysql> SELECT u.FullName, w.StockSymbol,
    ->       MAX(hs.ClosePrice) AS HighPrice,
    ->       MIN(hs.ClosePrice) AS LowPrice
    -> FROM Users u
    -> JOIN UserPortfolio up ON u.UserID = up.UsersUserID
    -> JOIN Watchlist w ON up.PortfoliosPortfolioID = w.PortfolioID
    -> JOIN HistoricalStocks hs ON w.StockSymbol = hs.StockSymbol
    -> WHERE u.UserID = 'JOHNWALTERS' OR u.UserID = 'ADAM82'
    -> GROUP BY u.UserID, w.StockSymbol
    -> LIMIT 15;

+-------------+-------------+-----------+----------+
| FullName    | StockSymbol | HighPrice | LowPrice |
+-------------+-------------+-----------+----------+
| Eric Miller | AMD         |    211.38 |     1.62 |
| Eric Miller | MSFT        |    467.56 |    23.01 |
| Eric Miller | UNH         |    555.15 |    27.85 |
| Eric Miller | MA          |    488.64 |    19.20 |
| Eric Miller | MCD         |    300.53 |    61.45 |
| Eric Miller | WMT         |     70.41 |    16.00 |
| Eric Miller | ABBV        |    182.10 |    33.71 |
| Eric Miller | GOOG        |    192.66 |    10.86 |
| Eric Miller | VZ          |     62.07 |    25.26 |
| Eric Miller | XOM         |    122.20 |    31.45 |
| Erika Poole | AVGO        |    182.89 |     1.68 |
| Erika Poole | COST        |    886.85 |    53.61 |
| Erika Poole | CRM         |    316.88 |    15.52 |
| Erika Poole | QCOM        |    227.09 |    31.96 |
| Erika Poole | WMT         |     70.41 |    16.00 |
+-------------+-------------+-----------+----------+
15 rows in set (13.91 sec)
```

Baseline:

```
| -> Table scan on <temporary>  (actual time=14272.603..14272.606 rows=18 loops=1)
    -> Aggregate using temporary table  (actual time=14272.598..14272.598 rows=18 loops=1)
        -> Nested loop inner join  (cost=54290.05 rows=56718) (actual time=345.112..14175.458 rows=65054 loops=1)
            -> Nested loop inner join  (cost=10.77 rows=17) (actual time=15.927..177.052 rows=18 loops=1)
                -> Nested loop inner join  (cost=5.19 rows=4) (actual time=0.096..93.475 rows=5 loops=1)
                    -> Filter: ((u.UserID = 'JOHNWALTERS') or (u.UserID = 'ADAM82'))  (cost=2.37 rows=2) (actual time=0.072..43.627 rows=2 loops=1)
                        -> Index range scan on u using PRIMARY over (UserID = 'ADAM82') OR (UserID = 'JOHNWALTERS')  (cost=2.37 rows=2) (actual time=0.062..43.605 rows=2 loops=1)
                    -> Covering index lookup on up using PRIMARY (UsersUserID=u.UserID)  (cost=1.31 rows=2) (actual time=0.669..24.919 rows=2 loops=2)
                -> Covering index lookup on w using PortfolioID (PortfolioID=up.PortfoliosPortfolioID)  (cost=1.12 rows=4) (actual time=8.881..16.713 rows=4 loops=5)
            -> Index lookup on hs using StockSymbol (StockSymbol=w.StockSymbol)  (cost=2948.55 rows=3417) (actual time=85.873..777.331 rows=3614 loops=18)
 |
```

Indexing on HistoricalStocks.ClosePrice:

```
| -> Table scan on <temporary>  (actual time=15481.310..15481.314 rows=18 loops=1)
    -> Aggregate using temporary table  (actual time=15481.305..15481.305 rows=18 loops=1)
        -> Nested loop inner join  (cost=62389.43 rows=56718) (actual time=510.726..15384.480 rows=65054 loops=1)
            -> Nested loop inner join  (cost=10.79 rows=17) (actual time=46.925..266.293 rows=18 loops=1)
                -> Nested loop inner join  (cost=5.21 rows=4) (actual time=44.243..131.822 rows=5 loops=1)
                    -> Filter: ((u.UserID = 'JOHNWALTERS') or (u.UserID = 'ADAM82'))  (cost=2.39 rows=2) (actual time=44.213..63.278 rows=2 loops=1)
                        -> Index range scan on u using PRIMARY over (UserID = 'ADAM82') OR (UserID = 'JOHNWALTERS')  (cost=2.39 rows=2) (actual time=44.204..63.257 rows=2 loops=1)
                    -> Covering index lookup on up using PRIMARY (UsersUserID=u.UserID)  (cost=1.31 rows=2) (actual time=0.712..34.266 rows=2 loops=2)
                -> Covering index lookup on w using PortfolioID (PortfolioID=up.PortfoliosPortfolioID)  (cost=1.12 rows=4) (actual time=12.038..26.891 rows=4 loops=5)
            -> Index lookup on hs using StockSymbol (StockSymbol=w.StockSymbol)  (cost=3436.43 rows=3417) (actual time=108.584..839.617 rows=3614 loops=18)
 |
```

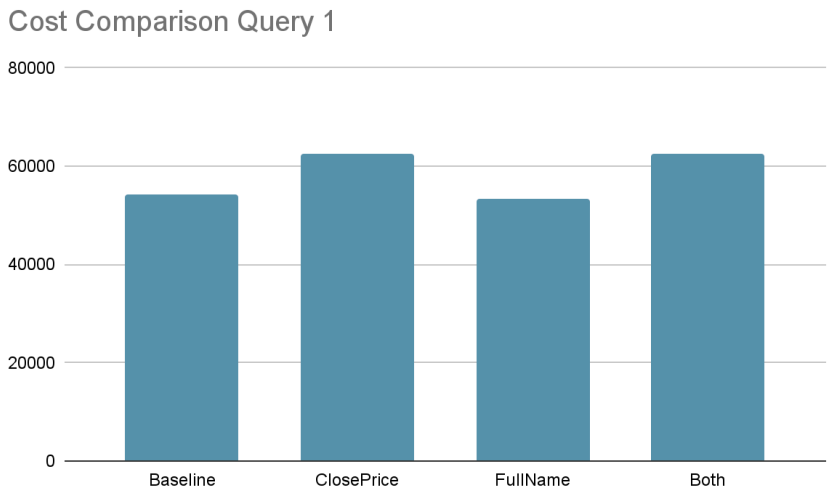
Indexing on Users.FullName:

```
| -> Table scan on <temporary>  (actual time=15884.971..15884.974 rows=18 loops=1)
    -> Aggregate using temporary table  (actual time=15884.965..15884.965 rows=18 loops=1)
        -> Nested loop inner join  (cost=53450.64 rows=56718) (actual time=433.406..15786.906 rows=65054 loops=1)
            -> Nested loop inner join  (cost=10.79 rows=17) (actual time=36.353..231.001 rows=18 loops=1)
                -> Nested loop inner join  (cost=5.21 rows=4) (actual time=14.008..76.371 rows=5 loops=1)
                    -> Filter: ((u.UserID = 'JOHNWALTERS') or (u.UserID = 'ADAM82'))  (cost=2.39 rows=2) (actual time=13.975..17.978 rows=2 loops=1)
                        -> Index range scan on u using PRIMARY over (UserID = 'ADAM82') OR (UserID = 'JOHNWALTERS')  (cost=2.39 rows=2) (actual time=13.965..17.956 rows=2 loops=1)
                    -> Covering index lookup on up using PRIMARY (UsersUserID=u.UserID)  (cost=1.31 rows=2) (actual time=0.617..29.191 rows=2 loops=2)
                -> Covering index lookup on w using PortfolioID (PortfolioID=up.PortfoliosPortfolioID)  (cost=1.12 rows=4) (actual time=11.195..30.923 rows=4 loops=5)
            -> Index lookup on hs using StockSymbol (StockSymbol=w.StockSymbol)  (cost=2897.98 rows=3417) (actual time=96.587..863.921 rows=3614 loops=18)
 |
```

Indexing on Both HistoricalStocks.ClosePrice and Users.FullName:

```
| -> Table scan on <temporary>  (actual time=13782.185..13782.188 rows=18 loops=1)
    -> Aggregate using temporary table  (actual time=13782.180..13782.180 rows=18 loops=1)
        -> Nested loop inner join  (cost=62389.43 rows=56718) (actual time=490.788..13686.974 rows=65054 loops=1)
            -> Nested loop inner join  (cost=10.79 rows=17) (actual time=76.180..323.367 rows=18 loops=1)
                -> Nested loop inner join  (cost=5.21 rows=4) (actual time=25.171..104.445 rows=5 loops=1)
                    -> Filter: ((u.UserID = 'JOHNWALTERS') or (u.UserID = 'ADAM82'))  (cost=2.39 rows=2) (actual time=25.138..28.040 rows=2 loops=1)
                        -> Index range scan on u using PRIMARY over (UserID = 'ADAM82') OR (UserID = 'JOHNWALTERS')  (cost=2.39 rows=2) (actual time=25.127..28.019 rows=2 loops=1)
                    -> Covering index lookup on up using PRIMARY (UsersUserID=u.UserID)  (cost=1.31 rows=2) (actual time=0.464..38.198 rows=2 loops=2)
                -> Covering index lookup on w using PortfolioID (PortfolioID=up.PortfoliosPortfolioID)  (cost=1.12 rows=4) (actual time=16.878..43.782 rows=4 loops=5)
            -> Index lookup on hs using StockSymbol (StockSymbol=w.StockSymbol)  (cost=3436.43 rows=3417) (actual time=95.931..742.113 rows=3614 loops=18)
 |
```

Hi/Lo Query Conclusion:



Comparing all of the costs for this query, we decided to keep the index on Users.FullName but scratch the index on HistoricalStocks.ClosePrice. This is because it yielded the lowest overall cost as compared to all other indexes and index combinations. Looking closer at the EXPLAIN ANALYZE reveals that adding any indices increases the cost slightly on the nested loop inner join as well as the covering index lookup steps, however, it creates enough of a decrease in cost on the index lookup step that it is able to decrease the overall cost to fall lower than the baseline overall cost, hence we kept this index. Adding an index on HistoricalStocks.ClosePrice actually increased both the nested loop inner join/covering index lookup steps as well as the final index lookup step, causing the overall cost to go up, both when using this index alone as well as in combination with ClosePrice. This is likely due to the fact that HistoricalStocks.ClosePrice has many more entries than does Users.FullName meaning it needs read through more entries when indexing this attribute.

2. People Who Own the Top 5 Most Valuable Stocks:

Query Syntax and Output:

Uses: Join multiple relations and Subqueries that cannot be easily replaced by a join

```
mysql> SELECT u.FullName, t.StockSymbol, tc.MarketCap
    -> FROM Users u
    -> JOIN UserPortfolio up ON u.UserID = up.UsersUserID
    -> JOIN Portfolios p ON up.PortfoliosPortfolioID = p.PortfolioID
    -> JOIN Transactions t ON p.PortfolioID = t.PortfoliosPortfolioID
    -> JOIN (SELECT c.StockSymbol, c.MarketCap
    -> FROM Companies c
    -> ORDER BY c.MarketCap DESC
    -> LIMIT 5 ) AS tc ON t.StockSymbol = tc.StockSymbol
    -> ORDER BY tc.MarketCap
    -> LIMIT 15;
+-----------------------+-------------+------------------+
| FullName              | StockSymbol | MarketCap        |
+-----------------------+-------------+------------------+
| Brett Cruz            | GOOG        | 2314040508416.00 |
| Scott Hull            | GOOG        | 2314040508416.00 |
| Jonathan Harrell      | GOOG        | 2314040508416.00 |
| Anthony Williams      | GOOG        | 2314040508416.00 |
| Elizabeth Leonard     | GOOG        | 2314040508416.00 |
| Christopher Henry     | GOOG        | 2314040508416.00 |
| Mrs. Michelle Hoover  | GOOG        | 2314040508416.00 |
| Brett Cruz            | GOOG        | 2314040508416.00 |
| Jeffery Oconnor       | GOOG        | 2314040508416.00 |
| Gary Ford             | GOOG        | 2314040508416.00 |
| Tracey Williams       | GOOG        | 2314040508416.00 |
| Katherine Dixon       | GOOG        | 2314040508416.00 |
| Christopher Porter Jr. | GOOG        | 2314040508416.00 |
| Eric Miller           | GOOG        | 2314040508416.00 |
| Kimberly Lyons        | GOOG        | 2314040508416.00 |
+-----------------------+-------------+------------------+
15 rows in set (1.01 sec)
```

Baseline:

```
| -> Sort: tc.MarketCap  (actual time=1980.527..1981.173 rows=10651 loops=1)
    -> Stream results  (cost=139202.39 rows=52050) (actual time=233.157..1960.590 rows=10651 loops=1)
        -> Nested loop inner join  (cost=139202.39 rows=52050) (actual time=233.142..1956.519 rows=10651 loops=1)
            -> Nested loop inner join  (cost=123726.81 rows=52050) (actual time=211.637..1630.074 rows=10651 loops=1)
                -> Nested loop inner join  (cost=108214.54 rows=52050) (actual time=182.577..920.632 rows=10651 loops=1)
                    -> Inner hash join (t.StockSymbol = tc.StockSymbol)  (cost=52460.98 rows=52050) (actual time=138.108..576.810 rows=10651 loops=1)
                        -> Table scan on t  (cost=278.70 rows=104100) (actual time=81.138..502.556 rows=104005 loops=1)
                        -> Hash
                            -> Table scan on tc  (cost=56.41..58.46 rows=5) (actual time=56.777..56.779 rows=5 loops=1)
                                -> Materialize  (cost=55.90..55.90 rows=5) (actual time=56.775..56.775 rows=5 loops=1)
                                    -> Limit: 5 row(s)  (cost=55.40 rows=5) (actual time=56.740..56.740 rows=5 loops=1)
                                        -> Sort: c.MarketCap DESC, limit input to 5 row(s) per chunk  (cost=55.40 rows=503) (actual time=56.739..56.739 rows=5 loops=1)
                                            -> Table scan on c  (cost=55.40 rows=503) (actual time=52.203..56.607 rows=503 loops=1)
                    -> Covering index lookup on up using PortfoliosPortfolioID (PortfoliosPortfolioID=t.PortfoliosPortfolioID)  (cost=0.97 rows=1) (actual time=0.032..0.032 rows=1 loops
=10651)
                -> Single-row index lookup on u using PRIMARY (UserID=up.UsersUserID)  (cost=0.20 rows=1) (actual time=0.066..0.066 rows=1 loops=10651)
            -> Single-row covering index lookup on p using PRIMARY (PortfolioID=t.PortfoliosPortfolioID)  (cost=0.20 rows=1) (actual time=0.030..0.030 rows=1 loops=10651)
 |
```

Indexing on Users.FullName:

```
| -> Sort: tc.MarketCap  (actual time=1203.116..1203.832 rows=10651 loops=1)
    -> Stream results  (cost=107042.17 rows=52050) (actual time=173.507..1194.704 rows=10651 loops=1)
        -> Nested loop inner join  (cost=107042.17 rows=52050) (actual time=173.495..1191.065 rows=10651 loops=1)
            -> Nested loop inner join  (cost=93379.05 rows=52050) (actual time=151.066..1021.599 rows=10651 loops=1)
                -> Nested loop inner join  (cost=85571.55 rows=52050) (actual time=151.025..890.378 rows=10651 loops=1)
                    -> Inner hash join (t.StockSymbol = tc.StockSymbol)  (cost=52339.62 rows=52050) (actual time=136.812..737.189 rows=10651 loops=1)
                        -> Table scan on t  (cost=254.43 rows=104100) (actual time=28.636..611.421 rows=104005 loops=1)
                        -> Hash
                            -> Table scan on tc  (cost=56.41..58.46 rows=5) (actual time=108.114..108.115 rows=5 loops=1)
                                -> Materialize  (cost=55.90..55.90 rows=5) (actual time=108.112..108.112 rows=5 loops=1)
                                    -> Limit: 5 row(s)  (cost=55.40 rows=5) (actual time=108.091..108.092 rows=5 loops=1)
                                        -> Sort: c.MarketCap DESC, limit input to 5 row(s) per chunk  (cost=55.40 rows=503) (actual time=108.090..108.091 rows=5 loops=1)
                                            -> Table scan on c  (cost=55.40 rows=503) (actual time=103.527..107.964 rows=503 loops=1)
                    -> Covering index lookup on up using PortfoliosPortfolioID (PortfoliosPortfolioID=t.PortfoliosPortfolioID)  (cost=0.54 rows=1) (actual time=0.014..0.014 rows=1 loops
=10651)
                -> Single-row index lookup on u using PRIMARY (UserID=up.UsersUserID)  (cost=0.05 rows=1) (actual time=0.012..0.012 rows=1 loops=10651)
            -> Single-row covering index lookup on p using PRIMARY (PortfolioID=t.PortfoliosPortfolioID)  (cost=0.16 rows=1) (actual time=0.016..0.016 rows=1 loops=10651)
 |
```

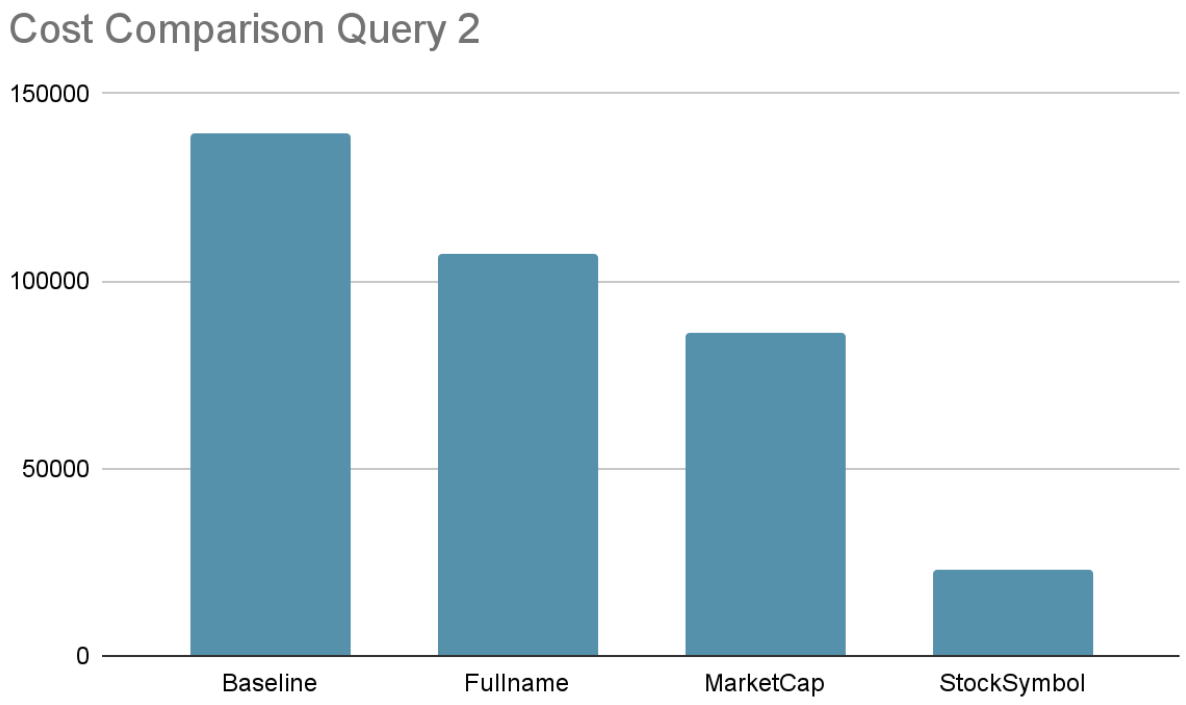
Indexing on Companies.MarketCap:

```
| -> Sort: tc.MarketCap  (actual time=86.545..87.298 rows=10651 loops=1)
    -> Stream results  (cost=85973.92 rows=52050) (actual time=0.177..80.136 rows=10651 loops=1)
        -> Nested loop inner join  (cost=85973.92 rows=52050) (actual time=0.172..77.445 rows=10651 loops=1)
            -> Nested loop inner join  (cost=78166.42 rows=52050) (actual time=0.165..65.817 rows=10651 loops=1)
                -> Nested loop inner join  (cost=70358.92 rows=52050) (actual time=0.145..52.755 rows=10651 loops=1)
                    -> Inner hash join (t.StockSymbol = tc.StockSymbol)  (cost=52141.42 rows=52050) (actual time=0.131..35.450 rows=10651 loops=1)
                        -> Table scan on t  (cost=225.87 rows=104100) (actual time=0.029..23.791 rows=104005 loops=1)
                        -> Hash
                            -> Table scan on tc  (cost=1.03..3.08 rows=5) (actual time=0.080..0.081 rows=5 loops=1)
                                -> Materialize  (cost=0.52..0.52 rows=5) (actual time=0.079..0.079 rows=5 loops=1)
                                    -> Limit: 5 row(s)  (cost=0.02 rows=5) (actual time=0.061..0.063 rows=5 loops=1)
                                        -> Covering index scan on c using comp_cap_idx (reverse)  (cost=0.02 rows=5) (actual time=0.016..0.017 rows=5 loops=1)
                    -> Covering index lookup on up using PortfoliosPortfolioID (PortfoliosPortfolioID=t.PortfoliosPortfolioID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops
=10651)
                -> Single-row index lookup on u using PRIMARY (UserID=up.UsersUserID)  (cost=0.05 rows=1) (actual time=0.001..0.001 rows=1 loops=10651)
            -> Single-row covering index lookup on p using PRIMARY (PortfolioID=t.PortfoliosPortfolioID)  (cost=0.05 rows=1) (actual time=0.001..0.001 rows=1 loops=10651)
 |
```

Indexing on Transactions.StockSymbol:

```
| -> Nested loop inner join  (cost=23104.57 rows=12393) (actual time=49.098..337.305 rows=10651 loops=1)
    -> Nested loop inner join  (cost=11298.16 rows=12393) (actual time=19.882..113.137 rows=10651 loops=1)
        -> Nested loop inner join  (cost=6960.66 rows=12393) (actual time=19.868..93.745 rows=10651 loops=1)
            -> Nested loop inner join  (cost=2623.16 rows=12393) (actual time=19.849..74.233 rows=10651 loops=1)
                -> Sort: tc.MarketCap  (cost=60.12..60.12 rows=5) (actual time=18.204..18.213 rows=5 loops=1)
                    -> Table scan on tc  (cost=56.41..58.46 rows=5) (actual time=18.192..18.193 rows=5 loops=1)
                        -> Materialize  (cost=55.90..55.90 rows=5) (actual time=18.190..18.190 rows=5 loops=1)
                            -> Limit: 5 row(s)  (cost=55.40 rows=5) (actual time=18.167..18.168 rows=5 loops=1)
                                -> Sort: c.MarketCap DESC, limit input to 5 row(s) per chunk  (cost=55.40 rows=503) (actual time=18.167..18.167 rows=5 loops=1)
                                    -> Table scan on c  (cost=55.40 rows=503) (actual time=14.500..18.050 rows=503 loops=1)
                -> Index lookup on t using trx_sym_idx (StockSymbol=tc.StockSymbol)  (cost=314.32 rows=2479) (actual time=0.717..11.074 rows=2130 loops=5)
            -> Covering index lookup on up using PortfoliosPortfolioID (PortfoliosPortfolioID=t.PortfoliosPortfolioID)  (cost=0.25 rows=1) (actual time=0.001..0.002 rows=1 loops=10651)
        -> Single-row index lookup on u using PRIMARY (UserID=up.UsersUserID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=10651)
    -> Single-row covering index lookup on p using PRIMARY (PortfolioID=t.PortfoliosPortfolioID)  (cost=0.85 rows=1) (actual time=0.021..0.021 rows=1 loops=10651)
 |
```

Top 5 Stocks Query Conclusion:



For this query we chose to leave the index on Transactions.StockSymbol as opposed to the other attributes as it provided a significant drop in cost when compared to the previous costs. We can observe a steady decrease in the Stream results step as we move from the baseline indexing, to adding an index on Users.FullName, then finally to an index on Companies.MarketCap. This steady decrease in intermediate cost is reflected in the overall cost. However, when switching to an index on Transactions.StockSymbol we seem to avoid having to perform the top two “Sort” and “Stream” lines and skip right to the “Nested loop inner join” lines. This is likely the cause of such a noticeable dropoff in cost as we compare an index on Transactions.StockSymbol to the alternatives.

3. List of Covid Best Performers:

Query Syntax and Output:

Uses: Join multiple relations and Aggregation via GROUP BY

```
mysql> SELECT hs.StockSymbol, c.Name, MAX(hs.ClosePrice) / MIN(hs.ClosePrice) AS Covid_Best_Performers
    -> FROM HistoricalStocks hs
    -> JOIN Companies c USING(StockSymbol)
    -> WHERE hs.`Date` BETWEEN '2020-03-14' AND '2020-12-31'
    -> GROUP BY hs.StockSymbol
    -> HAVING MAX(hs.ClosePrice) / MIN(hs.ClosePrice) > 5
    -> ORDER BY Covid_Best_Performers DESC
    -> LIMIT 15;
+-------------+----------------------------+----------------------+
| StockSymbol | Name                       | Covid_Best_Performers |
+-------------+----------------------------+----------------------+
| CZR         | Caesars Entertainment, Inc. |             10.853521 |
| TSLA        | Tesla, Inc.                |              9.768272 |
| ENPH        | Enphase Energy, Inc.       |              7.561067 |
| CRWD        | CrowdStrike Holdings, Inc. |              6.813087 |
| MRNA        | Moderna, Inc.              |              6.578621 |
| ETSY        | Etsy, Inc.                 |              6.228463 |
| TRGP        | Targa Resources Corp.      |              5.978858 |
+-------------+----------------------------+----------------------+
7 rows in set (0.65 sec)
```

Baseline:

```
| -> Sort: Covid_Best_Performers DESC  (actual time=618.477..618.478 rows=7 loops=1)
   -> Filter: ((max(hs.ClosePrice) / min(hs.ClosePrice)) > 5)  (actual time=618.325..618.447 rows=7 loops=1)
      -> Table scan on <temporary>  (actual time=618.280..618.340 rows=496 loops=1)
         -> Aggregate using temporary table  (actual time=618.278..618.278 rows=496 loops=1)
            -> Nested loop inner join  (cost=237467.31 rows=206368) (actual time=4.233..521.346 rows=100494 loops=1)
               -> Filter: (hs.`Date` between '2020-03-14' and '2020-12-31')  (cost=41417.71 rows=206368) (actual time=1.521..376.275 rows=100494 loops=1)
                  -> Index range scan on hs using PRIMARY over ('2020-03-14' <= Date <= '2020-12-31')  (cost=41417.71 rows=206368) (actual time=0.053..345.079 rows=100494 loops=1)
               -> Single-row index lookup on c using PRIMARY (StockSymbol=hs.StockSymbol)  (cost=0.85 rows=1) (actual time=0.001..0.001 rows=1 loops=100494)
 |
```

Index on Company.Name:

```
| -> Sort: Covid_Best_Performers DESC  (actual time=770.607..770.608 rows=7 loops=1)
   -> Filter: ((max(hs.ClosePrice) / min(hs.ClosePrice)) > 5)  (actual time=770.439..770.579 rows=7 loops=1)
      -> Table scan on <temporary>  (actual time=770.394..770.461 rows=496 loops=1)
         -> Aggregate using temporary table  (actual time=770.391..770.391 rows=496 loops=1)
            -> Nested loop inner join  (cost=113666.25 rows=206368) (actual time=0.063..666.486 rows=100494 loops=1)
               -> Filter: (hs.`Date` between '2020-03-14' and '2020-12-31')  (cost=41437.45 rows=206368) (actual time=0.040..536.801 rows=100494 loops=1)
                  -> Index range scan on hs using PRIMARY over ('2020-03-14' <= Date <= '2020-12-31')  (cost=41437.45 rows=206368) (actual time=0.034..506.078 rows=100494 loops=1)
               -> Single-row index lookup on c using PRIMARY (StockSymbol=hs.StockSymbol)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=100494)
 |
```

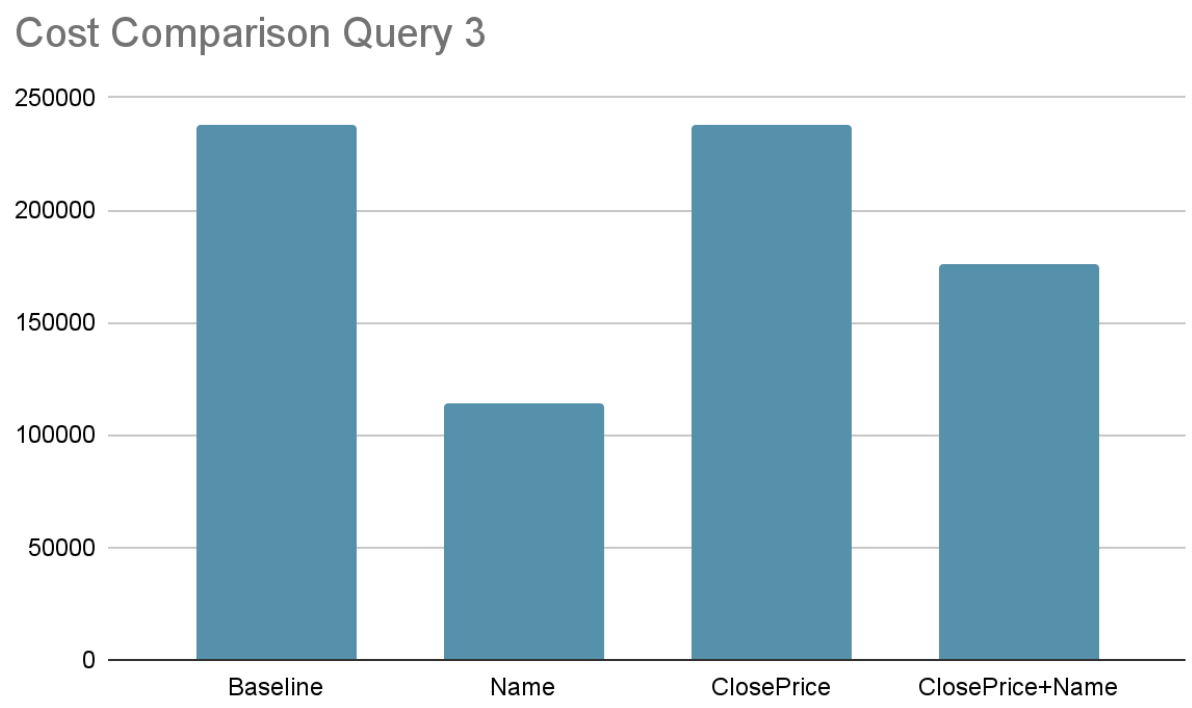
Index on HistoricalStocks.ClosePrice:

```
| -> Sort: Covid_Best_Performers DESC  (actual time=668.652..668.652 rows=7 loops=1)
   -> Filter: ((max(hs.ClosePrice) / min(hs.ClosePrice)) > 5)  (actual time=668.506..668.626 rows=7 loops=1)
      -> Table scan on <temporary>  (actual time=668.460..668.519 rows=496 loops=1)
         -> Aggregate using temporary table  (actual time=668.457..668.457 rows=496 loops=1)
            -> Nested loop inner join  (cost=237516.33 rows=206368) (actual time=36.326..567.205 rows=100494 loops=1)
               -> Filter: (hs.`Date` between '2020-03-14' and '2020-12-31')  (cost=41466.73 rows=206368) (actual time=0.051..375.787 rows=100494 loops=1)
                  -> Index range scan on hs using PRIMARY over ('2020-03-14' <= Date <= '2020-12-31')  (cost=41466.73 rows=206368) (actual time=0.044..345.131 rows=100494 loops=1)
               -> Single-row index lookup on c using PRIMARY (StockSymbol=hs.StockSymbol)  (cost=0.85 rows=1) (actual time=0.002..0.002 rows=1 loops=100494)
|
```

Index on HistoricalStocks.ClosePrice and Company.Name:

```
| -> Sort: Covid_Best_Performers DESC  (actual time=372.260..372.261 rows=7 loops=1)
   -> Filter: ((max(hs.ClosePrice) / min(hs.ClosePrice)) > 5)  (actual time=372.107..372.234 rows=7 loops=1)
      -> Table scan on <temporary>  (actual time=372.062..372.127 rows=496 loops=1)
         -> Aggregate using temporary table  (actual time=372.060..372.060 rows=496 loops=1)
            -> Nested loop inner join  (cost=175598.09 rows=206368) (actual time=0.107..275.801 rows=100494 loops=1)
               -> Filter: (hs.`Date` between '2020-03-14' and '2020-12-31')  (cost=41458.89 rows=206368) (actual time=0.044..129.570 rows=100494 loops=1)
                  -> Index range scan on hs using PRIMARY over ('2020-03-14' <= Date <= '2020-12-31')  (cost=41458.89 rows=206368) (actual time=0.039..100.025 rows=100494 loops=1)
               -> Single-row index lookup on c using PRIMARY (StockSymbol=hs.StockSymbol)  (cost=0.55 rows=1) (actual time=0.001..0.001 rows=1 loops=100494)
|
```

Covid Query Conclusion:



For this query we decided to keep the index on Company.Name, but drop the index on HistoricalStocks.ClosePrice. Unlike in the first query, this time the combination of both indices actually resulted in a cost that fell in the approximate middle of the other two. The main difference we noted was that, although all three indexing trials increased the cost of the “Filter” and “Index range scan” steps, putting an index on Company.Name seemed to decrease the cost of the “Single-row index lookup” step. Although this seemed contradictory at first, it does make sense as this step shows that it loops 100494 times (whereas the previously mentioned step loops only once over several records), creating a big impact on the overall cost with each repetition. Putting an index only on Company.Name decreased the cost of this step down from the baseline: 0.85, to the lowest of the 4 runs: 0.25 resulting in the significantly smaller overall cost. In the case of an index only on HistoricalStocks.ClosePrice, it did not change this inner cost value and it remained at 0.85. When combining the two indices, this value hit around an average of the two at 0.55, therefore leading to an overall cost that also fell around the average of the other two.

4. Tech-Heavy Portfolios:

Query Syntax and Output:

Uses: Join multiple relations, SET Operators, Aggregation via GROUP BY, and Subqueries that cannot be easily replaced by a join.

```
mysql> SELECT
    ->     p.PortfolioID,
    ->     t.StockSymbol,
    ->     c.Name AS CompanyName,
    ->     c.Sector
    -> FROM Portfolios p
    -> JOIN Transactions t ON p.PortfolioID = t.PortfoliosPortfolioID
    -> JOIN Companies c ON t.StockSymbol = c.StockSymbol
    -> JOIN (
    -> (SELECT p.PortfolioID, COUNT(*)
    -> FROM Portfolios p
    -> JOIN Transactions t ON p.PortfolioID = t.PortfoliosPortfolioID
    -> JOIN Companies c ON t.StockSymbol = c.StockSymbol
    -> GROUP BY p.PortfolioID
    -> HAVING COUNT(*) > 3
    -> ) INTERSECT (
    -> SELECT p.PortfolioID, COUNT(*)
    -> FROM Portfolios p
    -> JOIN Transactions t ON p.PortfolioID = t.PortfoliosPortfolioID
    -> JOIN Companies c ON t.StockSymbol = c.StockSymbol
    -> WHERE c.Sector = 'Technology'
    -> GROUP BY p.PortfolioID
    -> HAVING COUNT(*) > 3)
    -> ) tt ON tt.PortfolioID = p.PortfolioID
    -> ORDER BY p.PortfolioID
    -> LIMIT 15;
+-------------+-------------+-----------------------------------------------+------------+
| PortfolioID | StockSymbol | CompanyName                                   | Sector     |
+-------------+-------------+-----------------------------------------------+------------+
|        1424 | IBM         | International Business Machines Corporation | Technology |
|        1424 | ADBE        | Adobe Inc.                                    | Technology |
|        1424 | ORCL        | Oracle Corporation                            | Technology |
|        1424 | ACN         | Accenture plc                                 | Technology |
|        2018 | INTU        | Intuit Inc.                                   | Technology |
|        2018 | QCOM        | QUALCOMM Incorporated                         | Technology |
|        2018 | NVDA        | NVIDIA Corporation                            | Technology |
|        2018 | IBM         | International Business Machines Corporation | Technology |
|        6515 | QCOM        | QUALCOMM Incorporated                         | Technology |
|        6515 | MSFT        | Microsoft Corporation                         | Technology |
|        6515 | AAPL        | Apple Inc.                                    | Technology |
|        6515 | AMAT        | Applied Materials, Inc.                       | Technology |
|        6708 | AAPL        | Apple Inc.                                    | Technology |
|        6708 | AMD         | Advanced Micro Devices, Inc.                  | Technology |
|        6708 | TXN         | Texas Instruments Incorporated                | Technology |
+-------------+-------------+-----------------------------------------------+------------+
```

Baseline:

```
| -> Sort: p.PortfolioID  (actual time=1302.170..1302.175 rows=70 loops=1)
    -> Stream results  (cost=1156128.70 rows=53151) (actual time=1301.270..1302.123 rows=70 loops=1)
        -> Nested loop inner join  (cost=1156128.70 rows=53151) (actual time=1301.266..1302.090 rows=70 loops=1)
            -> Nested loop inner join  (cost=699027.50 rows=53151) (actual time=1301.258..1301.905 rows=70 loops=1)
                -> Nested loop inner join  (cost=400635.10 rows=10225) (actual time=1301.238..1301.513 rows=17 loops=1)
                    -> Table scan on tt  (cost=343793.13..343923.42 rows=10225) (actual time=1301.218..1301.409 rows=17 loops=1)
                        -> Intersect materialize with deduplication  (cost=343793.12..343793.12 rows=10225) (actual time=1301.201..1301.201 rows=13722 loops=1)
                            -> Filter: (count(0) > 3)  (cost=175986.52 rows=102249) (actual time=199.225..975.770 rows=13722 loops=1)
                                -> Group aggregate: count(0), count(0)  (cost=175986.52 rows=102249) (actual time=198.005..971.566 rows=20044 loops=1)
                                    -> Nested loop inner join  (cost=165761.62 rows=102249) (actual time=194.134..958.540 rows=104005 loops=1)
                                        -> Nested loop inner join  (cost=68625.04 rows=102249) (actual time=128.469..752.484 rows=104005 loops=1)
                                            -> Covering index scan on p using PRIMARY  (cost=2019.83 rows=19670) (actual time=30.342..90.489 rows=20044 loops=1)
                                            -> Filter: (t.StockSymbol is not null)  (cost=2.87 rows=5) (actual time=0.030..0.033 rows=5 loops=20044)
                                                -> Index lookup on t using PortfoliosPortfolioID (PortfoliosPortfolioID=p.PortfolioID)  (cost=2.87 rows=5) (actual time=0.030..0.032 rows
=5 loops=20044)
                                        -> Single-row covering index lookup on c using PRIMARY (StockSymbol=t.StockSymbol)  (cost=0.85 rows=1) (actual time=0.002..0.002 rows=1 loops=104
005)
                            -> Filter: (count(0) > 3)  (cost=166784.11 rows=10225) (actual time=0.284..311.637 rows=1625 loops=1)
                                -> Group aggregate: count(0), count(0)  (cost=166784.11 rows=10225) (actual time=0.171..310.330 rows=15552 loops=1)
                                    -> Nested loop inner join  (cost=165761.62 rows=10225) (actual time=0.145..303.363 rows=31743 loops=1)
                                        -> Nested loop inner join  (cost=68625.04 rows=102249) (actual time=0.129..161.537 rows=104005 loops=1)
                                            -> Covering index scan on p using PRIMARY  (cost=2019.83 rows=19670) (actual time=0.114..3.551 rows=20044 loops=1)
                                            -> Filter: (t.StockSymbol is not null)  (cost=2.87 rows=5) (actual time=0.006..0.007 rows=5 loops=20044)
                                                -> Index lookup on t using PortfoliosPortfolioID (PortfoliosPortfolioID=p.PortfolioID)  (cost=2.87 rows=5) (actual time=0.006..0.007 rows
=5 loops=20044)
                                        -> Filter: (c.Sector = 'Technology')  (cost=0.85 rows=0.1) (actual time=0.001..0.001 rows=0 loops=104005)
                                            -> Single-row index lookup on c using PRIMARY (StockSymbol=t.StockSymbol)  (cost=0.85 rows=1) (actual time=0.001..0.001 rows=1 loops=104005)
                -> Single-row covering index lookup on p using PRIMARY (PortfolioID=tt.PortfolioID)  (cost=0.54 rows=1) (actual time=0.006..0.006 rows=1 loops=17)
            -> Filter: (t.StockSymbol is not null)  (cost=2.87 rows=5) (actual time=0.021..0.023 rows=4 loops=17)
                -> Index lookup on t using PortfoliosPortfolioID (PortfoliosPortfolioID=tt.PortfolioID)  (cost=2.87 rows=5) (actual time=0.021..0.022 rows=4 loops=17)
        -> Single-row index lookup on c using PRIMARY (StockSymbol=t.StockSymbol)  (cost=0.85 rows=1) (actual time=0.002..0.002 rows=1 loops=70)
 |
```

Indexing Transactions.StockSymbol:

```
| -> Sort: p.PortfolioID  (actual time=806.205..806.209 rows=70 loops=1)
    -> Stream results  (cost=503142.29 rows=531513) (actual time=805.666..806.175 rows=70 loops=1)
        -> Nested loop inner join  (cost=503142.29 rows=531513) (actual time=805.660..806.149 rows=70 loops=1)
            -> Nested loop inner join  (cost=317112.71 rows=531513) (actual time=805.649..806.060 rows=70 loops=1)
                -> Nested loop inner join  (cost=131083.14 rows=102249) (actual time=805.615..805.838 rows=17 loops=1)
                    -> Table scan on tt  (cost=94015.39..95295.99 rows=102249) (actual time=805.599..805.769 rows=17 loops=1)
                        -> Intersect materialize with deduplication  (cost=94015.38..94015.38 rows=102249) (actual time=805.585..805.585 rows=13722 loops=1)
                            -> Filter: (count(0) > 3)  (cost=83790.47 rows=102249) (actual time=19.068..568.951 rows=13722 loops=1)
                                -> Group aggregate: count(0), count(0)  (cost=83790.47 rows=102249) (actual time=19.064..566.399 rows=20044 loops=1)
                                    -> Nested loop inner join  (cost=73565.57 rows=102249) (actual time=19.022..554.298 rows=104005 loops=1)
                                        -> Nested loop inner join  (cost=37778.41 rows=102249) (actual time=18.990..435.061 rows=104005 loops=1)
                                            -> Covering index scan on p using PRIMARY  (cost=1991.25 rows=19670) (actual time=0.043..4.019 rows=20044 loops=1)
                                            -> Filter: (t.StockSymbol is not null)  (cost=1.30 rows=5) (actual time=0.014..0.021 rows=5 loops=20044)
                                                -> Index lookup on t using PortfoliosPortfolioID (PortfoliosPortfolioID=p.PortfolioID)  (cost=1.30 rows=5) (actual time=0.014..0.021 rows
=5 loops=20044)
                                        -> Single-row covering index lookup on c using PRIMARY (StockSymbol=t.StockSymbol)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=104
005)
                            -> Filter: (count(0) > 3)  (actual time=223.551..225.750 rows=1625 loops=1)
                                -> Table scan on <temporary>  (actual time=223.547..225.025 rows=15552 loops=1)
                                    -> Aggregate using temporary table  (actual time=223.545..223.545 rows=15552 loops=1)
                                        -> Nested loop inner join  (cost=50568.99 rows=10408) (actual time=0.061..211.099 rows=31743 loops=1)
                                            -> Nested loop inner join  (cost=46926.05 rows=10408) (actual time=0.051..169.174 rows=31743 loops=1)
                                                -> Filter: (t.StockSymbol is not null)  (cost=10496.65 rows=104084) (actual time=0.036..31.070 rows=104005 loops=1)
                                                    -> Table scan on t  (cost=10496.65 rows=104084) (actual time=0.034..24.338 rows=104005 loops=1)
                                                -> Filter: (c.Sector = 'Technology')  (cost=0.25 rows=0.1) (actual time=0.001..0.001 rows=0 loops=104005)
                                                    -> Single-row index lookup on c using PRIMARY (StockSymbol=t.StockSymbol)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=
104005)
                                            -> Single-row covering index lookup on p using PRIMARY (PortfolioID=t.PortfoliosPortfolioID)  (cost=0.25 rows=1) (actual time=0.001..0.001 ro
ws=1 loops=31743)
                -> Single-row covering index lookup on p using PRIMARY (PortfolioID=tt.PortfolioID)  (cost=0.25 rows=1) (actual time=0.004..0.004 rows=1 loops=17)
            -> Filter: (t.StockSymbol is not null)  (cost=1.30 rows=5) (actual time=0.012..0.013 rows=4 loops=17)
                -> Index lookup on t using PortfoliosPortfolioID (PortfoliosPortfolioID=tt.PortfolioID)  (cost=1.30 rows=5) (actual time=0.011..0.012 rows=4 loops=17)
        -> Single-row index lookup on c using PRIMARY (StockSymbol=t.StockSymbol)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=70)
 |
```

Indexing Companies.Name:

```
| -> Sort: p.PortfolioID  (actual time=1390.800..1390.805 rows=70 loops=1)
    -> Stream results  (cost=643339.61 rows=53151) (actual time=1383.625..1390.752 rows=70 loops=1)
        -> Nested loop inner join  (cost=643339.61 rows=53151) (actual time=1383.619..1390.713 rows=70 loops=1)
            -> Nested loop inner join  (cost=505146.22 rows=53151) (actual time=1383.594..1390.540 rows=70 loops=1)
                -> Nested loop inner join  (cost=255186.08 rows=10225) (actual time=1380.198..1382.610 rows=17 loops=1)
                    -> Table scan on tt  (cost=202452.33..202582.62 rows=10225) (actual time=1351.304..1351.611 rows=17 loops=1)
                        -> Intersect materialize with deduplication  (cost=202452.32..202452.32 rows=10225) (actual time=1351.289..1351.289 rows=13722 loops=1)
                            -> Filter: (count(0) > 3)  (cost=105316.12 rows=102249) (actual time=131.899..1001.956 rows=13722 loops=1)
                                -> Group aggregate: count(0), count(0)  (cost=105316.12 rows=102249) (actual time=131.895..999.093 rows=20044 loops=1)
                                    -> Nested loop inner join  (cost=95091.22 rows=102249) (actual time=131.851..986.185 rows=104005 loops=1)
                                        -> Nested loop inner join  (cost=59304.06 rows=102249) (actual time=131.757..861.421 rows=104005 loops=1)
                                            -> Covering index scan on p using PRIMARY  (cost=2015.93 rows=19670) (actual time=67.126..162.036 rows=20044 loops=1)
                                            -> Filter: (t.StockSymbol is not null)  (cost=2.39 rows=5) (actual time=0.029..0.034 rows=5 loops=20044)
                                                -> Index lookup on t using PortfoliosPortfolioID (PortfoliosPortfolioID=p.PortfolioID)  (cost=2.39 rows=5) (actual time=0.029..0.034 rows
=5 loops=20044)
                                        -> Single-row covering index lookup on c using PRIMARY (StockSymbol=t.StockSymbol)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=104
005)
                            -> Filter: (count(0) > 3)  (cost=96113.71 rows=10225) (actual time=0.246..333.209 rows=1625 loops=1)
                                -> Group aggregate: count(0), count(0)  (cost=96113.71 rows=10225) (actual time=0.119..331.856 rows=15552 loops=1)
                                    -> Nested loop inner join  (cost=95091.22 rows=10225) (actual time=0.090..324.639 rows=31743 loops=1)
                                        -> Nested loop inner join  (cost=59304.06 rows=102249) (actual time=0.073..184.705 rows=104005 loops=1)
                                            -> Covering index scan on p using PRIMARY  (cost=2015.93 rows=19670) (actual time=0.045..3.642 rows=20044 loops=1)
                                            -> Filter: (t.StockSymbol is not null)  (cost=2.39 rows=5) (actual time=0.007..0.009 rows=5 loops=20044)
                                                -> Index lookup on t using PortfoliosPortfolioID (PortfoliosPortfolioID=p.PortfolioID)  (cost=2.39 rows=5) (actual time=0.007..0.008 rows
=5 loops=20044)
                                        -> Filter: (c.Sector = 'Technology')  (cost=0.25 rows=0.1) (actual time=0.001..0.001 rows=0 loops=104005)
                                            -> Single-row index lookup on c using PRIMARY (StockSymbol=t.StockSymbol)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=104005)
                -> Single-row covering index lookup on p using PRIMARY (PortfolioID=tt.PortfolioID)  (cost=0.50 rows=1) (actual time=1.823..1.823 rows=1 loops=17)
            -> Filter: (t.StockSymbol is not null)  (cost=2.39 rows=5) (actual time=0.464..0.466 rows=4 loops=17)
                -> Index lookup on t using PortfoliosPortfolioID (PortfoliosPortfolioID=tt.PortfolioID)  (cost=2.39 rows=5) (actual time=0.464..0.465 rows=4 loops=17)
        -> Single-row index lookup on c using PRIMARY (StockSymbol=t.StockSymbol)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=70)
 |
```

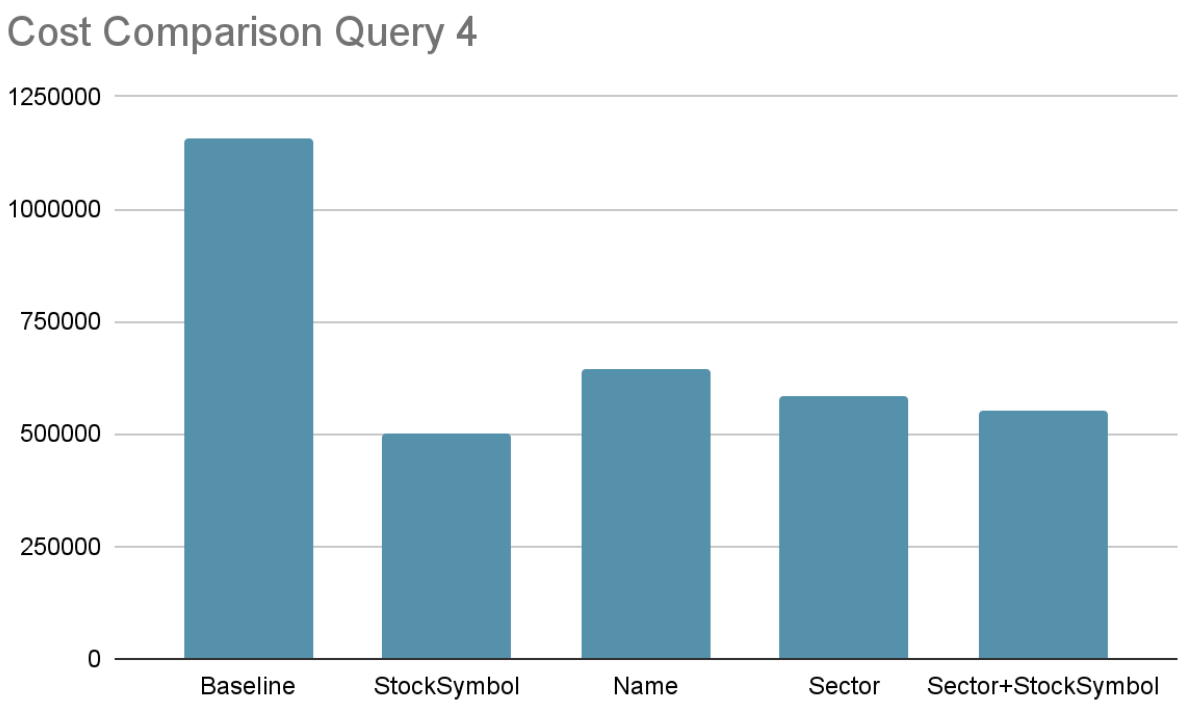
Indexing Companies.Sector:

```
| -> Sort: p.PortfolioID  (actual time=985.895..985.899 rows=70 loops=1)
    -> Stream results  (cost=584303.30 rows=85572) (actual time=985.239..985.857 rows=70 loops=1)
        -> Nested loop inner join  (cost=584303.30 rows=85572) (actual time=985.209..985.804 rows=70 loops=1)
            -> Nested loop inner join  (cost=437790.31 rows=85572) (actual time=985.199..985.702 rows=70 loops=1)
                -> Nested loop inner join  (cost=226811.98 rows=16151) (actual time=985.150..985.468 rows=17 loops=1)
                    -> Table scan on tt  (cost=187795.38..187999.74 rows=16151) (actual time=985.129..985.365 rows=17 loops=1)
                        -> Intersect materialize with deduplication  (cost=187795.36..187795.36 rows=16151) (actual time=985.105..985.105 rows=13722 loops=1)
                            -> Filter: (count(0) > 3)  (cost=97490.23 rows=104153) (actual time=0.206..608.468 rows=13722 loops=1)
                                -> Group aggregate: count(0), count(0)  (cost=97490.23 rows=104153) (actual time=0.203..605.524 rows=20044 loops=1)
                                    -> Nested loop inner join  (cost=87074.94 rows=104153) (actual time=0.168..592.213 rows=104005 loops=1)
                                        -> Nested loop inner join  (cost=50621.39 rows=104153) (actual time=0.150..461.660 rows=104005 loops=1)
                                            -> Covering index scan on p using PRIMARY  (cost=2000.44 rows=19658) (actual time=0.045..126.132 rows=20044 loops=1)
                                            -> Filter: (t.StockSymbol is not null)  (cost=1.94 rows=5) (actual time=0.013..0.016 rows=5 loops=20044)
                                                -> Index lookup on t using PortfoliosPortfolioID (PortfoliosPortfolioID=p.PortfolioID)  (cost=1.94 rows=5) (actual time=0.013..0.016 rows
=5 loops=20044)
                                        -> Single-row covering index lookup on c using PRIMARY (StockSymbol=t.StockSymbol)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=104
005)
                            -> Filter: (count(0) > 3)  (cost=88690.03 rows=16151) (actual time=0.291..361.355 rows=1625 loops=1)
                                -> Group aggregate: count(0), count(0)  (cost=88690.03 rows=16151) (actual time=0.107..359.722 rows=15552 loops=1)
                                    -> Nested loop inner join  (cost=87074.94 rows=16151) (actual time=0.065..351.677 rows=31743 loops=1)
                                        -> Nested loop inner join  (cost=50621.39 rows=104153) (actual time=0.044..196.363 rows=104005 loops=1)
                                            -> Covering index scan on p using PRIMARY  (cost=2000.44 rows=19658) (actual time=0.025..4.292 rows=20044 loops=1)
                                            -> Filter: (t.StockSymbol is not null)  (cost=1.94 rows=5) (actual time=0.007..0.009 rows=5 loops=20044)
                                                -> Index lookup on t using PortfoliosPortfolioID (PortfoliosPortfolioID=p.PortfolioID)  (cost=1.94 rows=5) (actual time=0.007..0.009 rows
=5 loops=20044)
                                        -> Filter: (c.Sector = 'Technology')  (cost=0.25 rows=0.2) (actual time=0.001..0.001 rows=0 loops=104005)
                                            -> Single-row index lookup on c using PRIMARY (StockSymbol=t.StockSymbol)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=104005)
                -> Single-row covering index lookup on p using PRIMARY (PortfolioID=tt.PortfolioID)  (cost=0.36 rows=1) (actual time=0.006..0.006 rows=1 loops=17)
            -> Filter: (t.StockSymbol is not null)  (cost=1.94 rows=5) (actual time=0.012..0.013 rows=4 loops=17)
                -> Index lookup on t using PortfoliosPortfolioID (PortfoliosPortfolioID=tt.PortfolioID)  (cost=1.94 rows=5) (actual time=0.012..0.013 rows=4 loops=17)
        -> Single-row index lookup on c using PRIMARY (StockSymbol=t.StockSymbol)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=70)
 |
```

Indexing Companies.Sector and Transactions.StockSymbol:

```
| -> Sort: p.PortfolioID  (actual time=1086.574..1086.578 rows=70 loops=1)
    -> Stream results  (cost=551875.17 rows=551828) (actual time=1086.032..1086.545 rows=70 loops=1)
        -> Nested loop inner join  (cost=551875.17 rows=551828) (actual time=1086.028..1086.503 rows=70 loops=1)
            -> Nested loop inner join  (cost=358736.51 rows=551828) (actual time=1086.019..1086.414 rows=70 loops=1)
                -> Nested loop inner join  (cost=165597.85 rows=104153) (actual time=1085.994..1086.197 rows=17 loops=1)
                    -> Table scan on tt  (cost=95757.63..97062.01 rows=104153) (actual time=1085.981..1086.155 rows=17 loops=1)
                        -> Intersect materialize with deduplication  (cost=95757.61..95757.61 rows=104153) (actual time=1085.962..1085.962 rows=13722 loops=1)
                            -> Filter: (count(0) > 3)  (cost=85342.32 rows=104153) (actual time=0.111..866.721 rows=13722 loops=1)
                                -> Group aggregate: count(0), count(0)  (cost=85342.32 rows=104153) (actual time=0.108..864.011 rows=20044 loops=1)
                                    -> Nested loop inner join  (cost=74927.02 rows=104153) (actual time=0.083..850.796 rows=104005 loops=1)
                                        -> Nested loop inner join  (cost=38473.47 rows=104153) (actual time=0.070..726.603 rows=104005 loops=1)
                                            -> Covering index scan on p using PRIMARY  (cost=2019.93 rows=19658) (actual time=0.034..229.498 rows=20044 loops=1)
                                            -> Filter: (t.StockSymbol is not null)  (cost=1.32 rows=5) (actual time=0.021..0.024 rows=5 loops=20044)
                                                -> Index lookup on t using PortfoliosPortfolioID (PortfoliosPortfolioID=p.PortfolioID)  (cost=1.32 rows=5) (actual time=0.021..0.024 rows
=5 loops=20044)
                                        -> Single-row covering index lookup on c using PRIMARY (StockSymbol=t.StockSymbol)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=104
005)
                            -> Filter: (count(0) > 3)  (actual time=204.110..206.341 rows=1625 loops=1)
                                -> Table scan on <temporary>  (actual time=204.105..205.559 rows=15552 loops=1)
                                    -> Aggregate using temporary table  (actual time=204.104..204.104 rows=15552 loops=1)
                                        -> Nested loop inner join  (cost=57555.75 rows=16143) (actual time=0.054..191.711 rows=31743 loops=1)
                                            -> Nested loop inner join  (cost=46933.25 rows=16143) (actual time=0.043..169.600 rows=31743 loops=1)
                                                -> Filter: (t.StockSymbol is not null)  (cost=10498.25 rows=104100) (actual time=0.029..29.990 rows=104005 loops=1)
                                                    -> Table scan on t  (cost=10498.25 rows=104100) (actual time=0.028..23.098 rows=104005 loops=1)
                                                -> Filter: (c.Sector = 'Technology')  (cost=0.25 rows=0.2) (actual time=0.001..0.001 rows=0 loops=104005)
                                                    -> Single-row index lookup on c using PRIMARY (StockSymbol=t.StockSymbol)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=
104005)
                                            -> Single-row covering index lookup on p using PRIMARY (PortfolioID=t.PortfoliosPortfolioID)  (cost=0.56 rows=1) (actual time=0.001..0.001 ro
ws=1 loops=31743)
                    -> Single-row covering index lookup on p using PRIMARY (PortfolioID=tt.PortfolioID)  (cost=0.56 rows=1) (actual time=0.002..0.002 rows=1 loops=17)
                -> Filter: (t.StockSymbol is not null)  (cost=1.32 rows=5) (actual time=0.011..0.012 rows=4 loops=17)
                    -> Index lookup on t using PortfoliosPortfolioID (PortfoliosPortfolioID=tt.PortfolioID)  (cost=1.32 rows=5) (actual time=0.011..0.012 rows=4 loops=17)
            -> Single-row index lookup on c using PRIMARY (StockSymbol=t.StockSymbol)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=70)
 |
```

Tech Heavy Query Conclusion:



Comparing all of the costs for this query, we decided to keep the index on Transactions.StockSymbol. This is because indexing on StockSymbol yielded the lowest overall

cost as compared to all other indexes and index combinations. Looking closer at the EXPLAIN ANALYZE data, we can see that, when using an index on Transactions.StockSymbol, the cost in the second “Nested loop inner join” is more than halved. This trend of getting a fraction of the cost is continued with each inner “Nested loop inner join” having even smaller fractions as they go. Looking further into the EXPLAIN ANALYZE output only confirms this trend continues for all inner loops and scans. This is likely due to how often we use the attribute “Transactions.StockSymbol” in our query, particularly when joining. It is essentially used just as often as Portfolios.PortfolioID, where PortfolioID is a primary key in the Portfolios table (and linked as a foreign key to the Transactions table) meaning it is already indexed. Therefore, it would make sense that adding an index to the Transactions.StockSymbol attribute (which isn’t directly linked to the Companies table where it is a primary key) would bring down the cost so drastically, as it was being used all throughout our query as if it was a primary key in our Transactions table.