DDL Implementation and Index Analyzation

We were able to establish the MySQL database through the Google Cloud Platform cloud shell. While we created the tables using the DDL implementation via the shell, we were able to make minor edits and import data by using cloud-sql-proxy in combination with dbeaver. We were able to import datasets with over 1000 rows for both CPU and GPU, and we were able to fill the Users relation with generated data. The Carts relation and the Permissions relation were manually filled with test data in order to verify the functionality of the relations. From there, we were able to test our queries and analyze the run cost using the cloud shell.

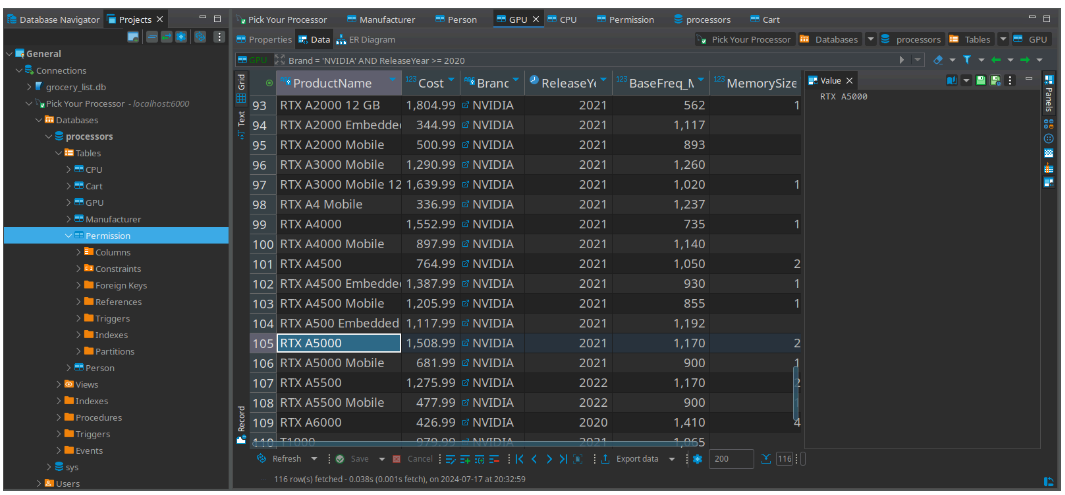
```
jetgeronimo@cloudshell:~ (cs411-db-429518)$ gcloud sql connect pt1 --user=root --quiet
Allowlisting your IP for incoming connection for 5 minutes...done.
Connecting to database with SQL user [root].Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 45859
Server version: 8.0.31-google (Google)

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```




```
mysql> SELECT COUNT(*) FROM Person;
+----------+
| COUNT(*) |
+----------+
|     6001 |
+----------+
1 row in set (0.00 sec)
```

```
mysql> SELECT COUNT(*) FROM CPU;
+----------+
| COUNT(*) |
+----------+
|     1026 |
+----------+
1 row in set (0.00 sec)
```

```
mysql> SELECT COUNT(*) FROM GPU;
+----------+
| COUNT(*) |
+----------+
|     2585 |
+----------+
1 row in set (0.01 sec)
```

```
mysql> SELECT COUNT(*) FROM Permission;
+----------+
| COUNT(*) |
+----------+
|        3 |
+----------+
1 row in set (0.04 sec)
```

```
mysql> SELECT COUNT(*) FROM Manufacturer;
+----------+
| COUNT(*) |
+----------+
|        8 |
+----------+
1 row in set (0.00 sec)
```

DDL

```
CREATE TABLE Person (
	Username VARCHAR(64),
	Budget DECIMAL(19,4),
	DateRegistered DATE,
	BirthDate DATE,
	PRIMARY KEY (Username)
);

CREATE TABLE Manufacturer (
	BrandName VARCHAR(64),
	CountryCode CHAR(2),
	YearEstablished YEAR,
	PRIMARY KEY (BrandName)
);

CREATE TABLE GPU (
	ProductName VARCHAR(64),
	Cost DECIMAL(19, 4),
	Brand VARCHAR(64),
	ReleaseYear YEAR,
	BaseFreq_MHz INT,
	MemorySize_GB DECIMAL(14, 6),
	MemClock_MHz INT,
	PRIMARY KEY (ProductName, Brand),
	FOREIGN KEY (Brand) REFERENCES Manufacturer(BrandName)
);
CREATE TABLE CPU (
	ProductName VARCHAR(64),
	Cost DECIMAL(19, 4),
	Brand VARCHAR(64),
	ReleaseYear YEAR,
	BaseFreq_GHz DECIMAL(2, 1),
	TurboFreq_GHz DECIMAL(2, 1),
	Cores INT,
	Threads INT,
	Cache_MB INT,
	TDP_W INT,

	PRIMARY KEY (ProductName, Brand),

	FOREIGN KEY (Brand) REFERENCES Manufacturer(BrandName)
);
```

```sql
CREATE TABLE Cart (
	CartId INT,
	GPUName VARCHAR(64),
	CPUName VARCHAR(64),
	Creator VARCHAR(64),
	CreationDate DATE,
	LastModified DATETIME,

	PRIMARY KEY (CartId),

	FOREIGN KEY (GPUName) REFERENCES GPU(ProductName),
	FOREIGN KEY(CPUName) REFERENCES CPU(ProductName)
);

CREATE TABLE Permission (
	CartId INT,
	Username VARCHAR(64),
	Permission ENUM('read', 'write', 'manage'),
	TimeAdded DATETIME,

	PRIMARY KEY (CartId, Username),

	FOREIGN KEY (CartId) REFERENCES Cart(CartId),
	FOREIGN KEY (Username) REFERENCES Person(Username)
);
```

Query 1:

Retrieve Top 15 Most Expensive Products from CPU and GPU Tables That Released After 2023

```
SELECT p.ProductName, p.Brand, p.Cost, p.ReleaseYear
FROM (
    SELECT ProductName, Brand, Cost, ReleaseYear
    FROM CPU
    UNION
    SELECT ProductName, Brand, Cost, ReleaseYear
    FROM GPU
) AS p
WHERE
     p.ReleaseYear >= 2023
ORDER BY p.Cost DESC
LIMIT 15;
```

ProductName	Brand	Cost	ReleaseYear
GeForce RTX 4050	NVIDIA	1373.9900	2023
Core i9-13900F	Intel	996.9900	2023
Core i5-13500HX	Intel	990.9900	2023
Core i7-1370PRE	Intel	978.9900	2023
Core i5-14600T	Intel	976.9900	2024
Core i5-13450HX	Intel	973.9900	2023
Core i9-14900KS	Intel	962.9900	2024
Core i7-13700TE	Intel	955.9900	2023
Core i5-14600KF	Intel	954.9900	2023
Core i9-13900E	Intel	954.9900	2023
Core i9-14900KF	Intel	953.9900	2023
Core i9-14900T	Intel	922.9900	2024
Core i5-13500T	Intel	918.9900	2023
Core i3-14100	Intel	908.9900	2024
Core i9-14900	Intel	906.9900	2024

15 rows in set (0.00 sec)

Non-indexed Cost

```
| -> Limit: 15 row(s)  (cost=633.96..633.96 rows=15) (actual time=3.178..3.181 rows=15 loops=1)
    -> Sort: p.Cost DESC, limit input to 15 row(s) per chunk  (cost=633.96..633.96 rows=15) (actual time=3.177..3.179 rows=15 loops=1)
        -> Table scan on p  (cost=490.22..507.74 rows=1204) (actual time=3.084..3.110 rows=116 loops=1)
            -> Union materialize with deduplication  (cost=490.20..490.20 rows=1204) (actual time=3.081..3.081 rows=116 loops=1)
                -> Filter: (`CPU`.ReleaseYear >= 2023)  (cost=104.60 rows=342) (actual time=0.105..0.902 rows=115 loops=1)
                    -> Table scan on CPU  (cost=104.60 rows=1026) (actual time=0.093..0.805 rows=1026 loops=1)
                -> Filter: (GPU.ReleaseYear >= 2023)  (cost=265.25 rows=862) (actual time=0.733..2.050 rows=1 loops=1)
                    -> Table scan on GPU  (cost=265.25 rows=2585) (actual time=0.082..1.824 rows=2585 loops=1)
 |
```

Cost: 633.96

INDEX ON GPU.Cost

```
| -> Limit: 15 row(s)  (cost=633.96..633.96 rows=15) (actual time=1.794..1.796 rows=15 loops=1)
    -> Sort: p.Cost DESC, limit input to 15 row(s) per chunk  (cost=633.96..633.96 rows=15) (actual time=1.794..1.795 rows=15 loops=1)
        -> Table scan on p  (cost=490.22..507.74 rows=1204) (actual time=1.733..1.750 rows=116 loops=1)
            -> Union materialize with deduplication  (cost=490.20..490.20 rows=1204) (actual time=1.731..1.731 rows=116 loops=1)
                -> Filter: (`CPU`.ReleaseYear >= 2023)  (cost=104.60 rows=342) (actual time=0.126..0.521 rows=115 loops=1)
                    -> Table scan on CPU  (cost=104.60 rows=1026) (actual time=0.118..0.452 rows=1026 loops=1)
                -> Filter: (GPU.ReleaseYear >= 2023)  (cost=265.25 rows=862) (actual time=0.380..1.134 rows=1 loops=1)
                    -> Table scan on GPU  (cost=265.25 rows=2585) (actual time=0.053..0.973 rows=2585 loops=1)
 |
```

Cost: 633.96

INDEX ON CPU.Cost

```
| -> Limit: 15 row(s)  (cost=633.96..633.96 rows=15) (actual time=1.841..1.843 rows=15 loops=1)
    -> Sort: p.Cost DESC, limit input to 15 row(s) per chunk  (cost=633.96..633.96 rows=15) (actual time=1.840..1.842 rows=15 loops=1)
        -> Table scan on p  (cost=490.22..507.74 rows=1204) (actual time=1.774..1.791 rows=116 loops=1)
            -> Union materialize with deduplication  (cost=490.20..490.20 rows=1204) (actual time=1.772..1.772 rows=116 loops=1)
                -> Filter: (`CPU`.ReleaseYear >= 2023)  (cost=104.60 rows=342) (actual time=0.077..0.584 rows=115 loops=1)
                    -> Table scan on CPU  (cost=104.60 rows=1026) (actual time=0.069..0.513 rows=1026 loops=1)
                -> Filter: (GPU.ReleaseYear >= 2023)  (cost=265.25 rows=862) (actual time=0.402..1.104 rows=1 loops=1)
                    -> Table scan on GPU  (cost=265.25 rows=2585) (actual time=0.072..0.950 rows=2585 loops=1)
 |
```

Cost: 633.96

INDEX ON CPU.Cost AND GPU.Cost

```
| -> Limit: 15 row(s)  (cost=633.96..633.96 rows=15) (actual time=2.196..2.198 rows=15 loops=1)
    -> Sort: p.Cost DESC, limit input to 15 row(s) per chunk  (cost=633.96..633.96 rows=15) (actual time=2.196..2.197 rows=15 loops=1)
        -> Table scan on p  (cost=490.22..507.74 rows=1204) (actual time=2.133..2.150 rows=116 loops=1)
            -> Union materialize with deduplication  (cost=490.20..490.20 rows=1204) (actual time=2.129..2.129 rows=116 loops=1)
                -> Filter: (`CPU`.ReleaseYear >= 2023)  (cost=104.60 rows=342) (actual time=0.077..0.459 rows=115 loops=1)
                    -> Table scan on CPU  (cost=104.60 rows=1026) (actual time=0.069..0.394 rows=1026 loops=1)
                -> Filter: (GPU.ReleaseYear >= 2023)  (cost=265.25 rows=862) (actual time=0.522..1.570 rows=1 loops=1)
                    -> Table scan on GPU  (cost=265.25 rows=2585) (actual time=0.049..1.384 rows=2585 loops=1)
 |
```

Cost: 633.96

INDEX ON CPU.Cost AND GPU.Cost AND CPU.ReleaseYear AND GPU.ReleaseYear

```
| -> Limit: 15 row(s)  (cost=85.73..85.73 rows=15) (actual time=0.619..0.621 rows=15 loops=1)
    -> Sort: p.Cost DESC, limit input to 15 row(s) per chunk  (cost=85.73..85.73 rows=15) (actual time=0.618..0.619 rows=15 loops=1)
        -> Table scan on p  (cost=64.35..68.27 rows=116) (actual time=0.552..0.569 rows=116 loops=1)
            -> Union materialize with deduplication  (cost=64.32..64.32 rows=116) (actual time=0.551..0.551 rows=116 loops=1)
                -> Index range scan on CPU using idx_releaseyear_cpu over (2023 <= ReleaseYear), with index condition: (`CPU`.ReleaseYear >= 2023)  (cost=52.01 rows=115) (actual time=0.047..0.348 rows=115 loops=1)
                -> Index range scan on GPU using idx_releaseyear_gpu over (2023 <= ReleaseYear), with index condition: (GPU.ReleaseYear >= 2023)  (cost=0.71 rows=1) (actual time=0.015..0.016 rows=1 loops=1)
 |
```

Cost: 85.73

INDEX ON CPU.ReleaseYear AND GPU.ReleaseYear

```
| -> Limit: 15 row(s)  (cost=85.73..85.73 rows=15) (actual time=0.459..0.462 rows=15 loops=1)
    -> Sort: p.Cost DESC, limit input to 15 row(s) per chunk  (cost=85.73..85.73 rows=15) (actual time=0.459..0.460 rows=15 loops=1)
        -> Table scan on p  (cost=64.35..68.27 rows=116) (actual time=0.405..0.421 rows=116 loops=1)
            -> Union materialize with deduplication  (cost=64.32..64.32 rows=116) (actual time=0.404..0.404 rows=116 loops=1)
                -> Index range scan on CPU using idx_cpu_releaseyear over (2023 <= ReleaseYear), with index condition: (`CPU`.ReleaseYear >= 2023)  (cost=52.01 rows=115) (actual time=0.028..0.316 rows=115 loops=1)
                -> Index range scan on GPU using idx_gpu_releaseyear over (2023 <= ReleaseYear), with index condition: (GPU.ReleaseYear >= 2023)  (cost=0.71 rows=1) (actual time=0.009..0.010 rows=1 loops=1)
 |
```

Cost: 85.73

Justification:

The indexing would ideally create a copy of the column associated with cost for GPU and CPU and keep it in a sorted order. This should allow for faster access to the ordered data, and that's what happens here. By having a copied ordering of the year column we were able to reduce our cost from 634 to 86. Indexing by ReleaseYear cuts the cost of the query substantially since the query can isolate the condition (ReleaseYear >= 2023) on which it searches for entries. Our indexing reduced our cost by over 7 times, we initially used cost but because we don't filter or aggregate by the CPU or GPU cost this had no effect.

Query 2:

Find users with the most expensive carts

```
SELECT
   c.Creator,
   SUM(g.Cost + cp.Cost) AS TotalCartCost
FROM
   Cart AS c
INNER JOIN
   GPU g ON c.GPUName = g.ProductName
INNER JOIN
   CPU cp ON c.CPUName = cp.ProductName
GROUP BY
   c.Creator
ORDER BY
   TotalCartCost DESC
LIMIT 15;
```

```
+-----------------+--------------+
| Creator         | TotalCartCost |
+-----------------+--------------+
| EthanFlores216  |     2417.9800 |
| jet             |     1643.9800 |
| KevinKennedy626 |     1498.9800 |
+-----------------+--------------+
3 rows in set (0.00 sec)
```

Non-indexed

```
| -> Limit: 15 row(s)  (actual time=0.181..0.182 rows=3 loops=1)
    -> Sort: TotalCartCost DESC, limit input to 15 row(s) per chunk  (actual time=0.181..0.181 rows=3 loops=1)
        -> Table scan on <temporary>  (actual time=0.168..0.169 rows=3 loops=1)
            -> Aggregate using temporary table  (actual time=0.168..0.168 rows=3 loops=1)
                -> Nested loop inner join  (cost=1.85 rows=2) (actual time=0.109..0.145 rows=3 loops=1)
                    -> Nested loop inner join  (cost=1.15 rows=2) (actual time=0.095..0.114 rows=3 loops=1)
                        -> Filter: ((c.CPUName is not null) and (c.GPUName is not null))  (cost=0.45 rows=2) (actual time=0.063..
0.066 rows=3 loops=1)
                            -> Table scan on c  (cost=0.45 rows=2) (actual time=0.062..0.064 rows=3 loops=1)
                        -> Index lookup on cp using PRIMARY (ProductName=c.CPUName)  (cost=0.30 rows=1) (actual time=0.014..0.015
 rows=1 loops=3)
                    -> Index lookup on g using PRIMARY (ProductName=c.GPUName)  (cost=0.30 rows=1) (actual time=0.008..0.010 rows
=1 loops=3)
```

Cost:1.85

After INDEX ON CPU.Cart AND GPU.Cart

```
| -> Limit: 15 row(s)  (actual time=0.157..0.158 rows=3 loops=1)
    -> Sort: TotalCartCost DESC, limit input to 15 row(s) per chunk  (actual time=0.157..0.157 rows=3 loops=1)
        -> Table scan on <temporary>  (actual time=0.144..0.144 rows=3 loops=1)
            -> Aggregate using temporary table  (actual time=0.143..0.143 rows=3 loops=1)
                -> Nested loop inner join  (cost=1.85 rows=2) (actual time=0.081..0.119 rows=3 loops=1)
                    -> Nested loop inner join  (cost=1.15 rows=2) (actual time=0.057..0.078 rows=3 loops=1)
                        -> Filter: ((c.CPUName is not null) and (c.GPUName is not null))  (cost=0.45 rows=2) (actual time=0.028..
0.031 rows=3 loops=1)
                            -> Table scan on c  (cost=0.45 rows=2) (actual time=0.026..0.029 rows=3 loops=1)
                        -> Index lookup on cp using PRIMARY (ProductName=c.CPUName)  (cost=0.30 rows=1) (actual time=0.014..0.015
 rows=1 loops=3)
                    -> Index lookup on g using PRIMARY (ProductName=c.GPUName)  (cost=0.30 rows=1) (actual time=0.011..0.013 rows
=1 loops=3)
 |
```

Cost: 1.85

After INDEX ON Cart.CPUName AND Cart.GPUName AND Cart.Creator

```
| -> Limit: 15 row(s)  (actual time=0.184..0.184 rows=3 loops=1)
    -> Sort: TotalCartCost DESC, limit input to 15 row(s) per chunk  (actual time=0.183..0.184 rows=3 loops=1)
        -> Table scan on <temporary>  (actual time=0.171..0.172 rows=3 loops=1)
            -> Aggregate using temporary table  (actual time=0.170..0.170 rows=3 loops=1)
                -> Nested loop inner join  (cost=1.85 rows=2) (actual time=0.064..0.148 rows=3 loops=1)
                    -> Nested loop inner join  (cost=1.15 rows=2) (actual time=0.052..0.115 rows=3 loops=1)
                        -> Filter: ((c.CPUName is not null) and (c.GPUName is not null))  (cost=0.45 rows=2) (actual time=0.026..0.029 rows=3 loops=1)
                            -> Table scan on c  (cost=0.45 rows=2) (actual time=0.024..0.027 rows=3 loops=1)
                        -> Index lookup on cp using PRIMARY (ProductName=c.CPUName)  (cost=0.30 rows=1) (actual time=0.015..0.028 rows=1 loops=3)
                    -> Index lookup on g using PRIMARY (ProductName=c.GPUName)  (cost=0.30 rows=1) (actual time=0.009..0.011 rows=1 loops=3)
 |
```

Cost: 1.85

After INDEX ON Cart.Creator

```
| -> Limit: 15 row(s)  (actual time=0.188..0.189 rows=3 loops=1)
    -> Sort: TotalCartCost DESC, limit input to 15 row(s) per chunk  (actual time=0.188..0.188 rows=3 loops=1)
        -> Table scan on <temporary>  (actual time=0.174..0.175 rows=3 loops=1)
            -> Aggregate using temporary table  (actual time=0.173..0.173 rows=3 loops=1)
                -> Nested loop inner join  (cost=1.85 rows=2) (actual time=0.090..0.146 rows=3 loops=1)
                    -> Nested loop inner join  (cost=1.15 rows=2) (actual time=0.068..0.097 rows=3 loops=1)
                        -> Filter: ((c.CPUName is not null) and (c.GPUName is not null))  (cost=0.45 rows=2) (actual time=0.031..0.035 rows=3 loops=1)
                            -> Table scan on c  (cost=0.45 rows=2) (actual time=0.029..0.033 rows=3 loops=1)
                        -> Index lookup on cp using PRIMARY (ProductName=c.CPUName)  (cost=0.30 rows=1) (actual time=0.019..0.020 rows=1 loops=3)
                    -> Index lookup on g using PRIMARY (ProductName=c.GPUName)  (cost=0.30 rows=1) (actual time=0.013..0.016 rows=1 loops=3)
 |
```

Cost: 1.85

Justification

We chose to create indices on GPUName, CPUName, and Creator since the names are being used for the inner joins and the cart creators are being used to aggregate the carts. However, using indices on these attributes did not appear to cut down our costs, likely due to a lack of entries in the Cart relation. Since the Cart relation contains only three columns, the query does not appear to be processing enough information to derive any benefit from the added indices. Testing the combination of product names and cart creators made no difference. If we are able to add significantly more entries to Carts, we will likely be able to gain a better perspective on the performance benefits that could be derived from indexing product names and cart creators.

Query 3:

Find the Brands with Products Above Average Cost For Either Type

```
SELECT
    Manufacturer.BrandName
FROM
    Manufacturer
INNER JOIN
    CPU
    ON Manufacturer.BrandName = CPU.Brand
INNER JOIN
    GPU
    ON Manufacturer.BrandName = GPU.Brand
WHERE
    CPU.Cost > (
    SELECT
    AVG(c.Cost)
    FROM
    CPU AS c
    )
    OR GPU.Cost > (
    SELECT
    AVG(g.Cost)
    FROM
    GPU AS g
    )
GROUP BY
    Manufacturer.BrandName
ORDER BY
    Manufacturer.YearEstablished DESC
LIMIT 15;
```

RESULT

```
+-----------+
| BrandName |
+-----------+
| Intel     |
+-----------+
1 row in set (0.35 sec)
```

No Indexing

```
| -> Limit: 15 row(s)  (actual time=302.532..302.533 rows=1 loops=1)
    -> Sort: Manufacturer.YearEstablished DESC, limit input to 15 row(s) per chunk  (actual time=302.532..302.532 rows=1 loops=1)
        -> Table scan on <temporary>  (cost=87545.46..91692.02 rows=331526) (actual time=302.516..302.516 rows=1 loops=1)
            -> Temporary table with deduplication  (cost=87545.45..87545.45 rows=331526) (actual time=302.514..302.514 rows=1 loops=1)
                -> Nested loop inner join  (cost=54392.82 rows=331526) (actual time=0.807..271.455 rows=98499 loops=1)
                    -> Nested loop inner join  (cost=463.70 rows=1026) (actual time=0.075..1.213 rows=1026 loops=1)
                        -> Table scan on CPU  (cost=104.60 rows=1026) (actual time=0.058..0.621 rows=1026 loops=1)
                        -> Single-row index lookup on Manufacturer using PRIMARY (BrandName=`CPU`.Brand)  (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=1026)
                    -> Filter: ((`CPU`.Cost > (select #2)) or (GPU.Cost > (select #3)))  (cost=20.28 rows=323) (actual time=0.173..0.256 rows=96 loops=1026)
                        -> Index lookup on GPU using Brand (Brand=`CPU`.Brand)  (cost=20.28 rows=323) (actual time=0.171..0.226 rows=122 loops=1026)
                        -> Select #2 (subquery in condition; run only once)
                            -> Aggregate: avg(c.Cost)  (cost=207.20 rows=1) (actual time=0.434..0.434 rows=1 loops=1)
                                -> Table scan on c  (cost=104.60 rows=1026) (actual time=0.032..0.317 rows=1026 loops=1)
                        -> Select #3 (subquery in condition; run only once)
                            -> Aggregate: avg(g.Cost)  (cost=523.75 rows=1) (actual time=0.958..0.958 rows=1 loops=1)
                                -> Table scan on g  (cost=265.25 rows=2585) (actual time=0.031..0.661 rows=2585 loops=1)
 |
```

COST = 87545.46

Index: CPU.Cost, GPU.Cost

```
| -> Limit: 15 row(s)  (actual time=322.262..322.262 rows=1 loops=1)
    -> Sort: Manufacturer.YearEstablished DESC, limit input to 15 row(s) per chunk  (actual time=322.261..322.261 rows=1 loops=1)
        -> Table scan on <temporary>  (cost=87545.46..91692.02 rows=331526) (actual time=322.240..322.240 rows=1 loops=1)
            -> Temporary table with deduplication  (cost=87545.45..87545.45 rows=331526) (actual time=322.236..322.236 rows=1 loops=1)
                -> Nested loop inner join  (cost=54392.82 rows=331526) (actual time=0.289..289.240 rows=98499 loops=1)
                    -> Nested loop inner join  (cost=463.70 rows=1026) (actual time=0.061..1.363 rows=1026 loops=1)
                        -> Covering index scan on CPU using CPUCostIndex  (cost=104.60 rows=1026) (actual time=0.042..0.625 rows=1026 loops=1)
                        -> Single-row index lookup on Manufacturer using PRIMARY (BrandName=`CPU`.Brand)  (cost=0.25 rows=1) (actual time=0.000..0.001 rows=1 loops=1026)
                    -> Filter: ((`CPU`.Cost > (select #2)) or (GPU.Cost > (select #3)))  (cost=20.28 rows=323) (actual time=0.182..0.273 rows=96 loops=1026)
                        -> Index lookup on GPU using Brand (Brand=`CPU`.Brand)  (cost=20.28 rows=323) (actual time=0.182..0.242 rows=122 loops=1026)
                        -> Select #2 (subquery in condition; run only once)
                            -> Aggregate: avg(c.Cost)  (cost=207.20 rows=1) (actual time=0.400..0.400 rows=1 loops=1)
                                -> Covering index scan on c using CPUCostIndex  (cost=104.60 rows=1026) (actual time=0.044..0.277 rows=1026 loops=1)
                        -> Select #3 (subquery in condition; run only once)
                            -> Aggregate: avg(g.Cost)  (cost=523.75 rows=1) (actual time=0.811..0.811 rows=1 loops=1)
                                -> Covering index scan on g using GPUCostIndex  (cost=265.25 rows=2585) (actual time=0.030..0.515 rows=2585 loops=1)
 |
```

COST = 87545.46

Index: CPU.Cost, GPU.Cost, Manufacturer.BrandName

```
| -> Limit: 15 row(s)  (actual time=307.748..307.749 rows=1 loops=1)
    -> Sort: Manufacturer.YearEstablished DESC, limit input to 15 row(s) per chunk  (actual time=307.748..307.748 rows=1 loops=1)
        -> Table scan on <temporary>  (cost=87545.46..91692.02 rows=331526) (actual time=307.729..307.730 rows=1 loops=1)
            -> Temporary table with deduplication  (cost=87545.45..87545.45 rows=331526) (actual time=307.726..307.726 rows=1 loops=1)
                -> Nested loop inner join  (cost=54392.82 rows=331526) (actual time=0.306..276.811 rows=98499 loops=1)
                    -> Nested loop inner join  (cost=463.70 rows=1026) (actual time=0.056..1.222 rows=1026 loops=1)
                        -> Covering index scan on CPU using CPUCostIndex  (cost=104.60 rows=1026) (actual time=0.039..0.573 rows=1026 loops=1)
                        -> Single-row index lookup on Manufacturer using PRIMARY (BrandName=`CPU`.Brand)  (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=1026)
                    -> Filter: ((`CPU`.Cost > (select #2)) or (GPU.Cost > (select #3)))  (cost=20.28 rows=323) (actual time=0.176..0.261 rows=96 loops=1026)
                        -> Index lookup on GPU using Brand (Brand=`CPU`.Brand)  (cost=20.28 rows=323) (actual time=0.175..0.232 rows=122 loops=1026)
                        -> Select #2 (subquery in condition; run only once)
                            -> Aggregate: avg(c.Cost)  (cost=207.20 rows=1) (actual time=0.381..0.381 rows=1 loops=1)
                                -> Covering index scan on c using CPUCostIndex  (cost=104.60 rows=1026) (actual time=0.047..0.250 rows=1026 loops=1)
                        -> Select #3 (subquery in condition; run only once)
                            -> Aggregate: avg(g.Cost)  (cost=523.75 rows=1) (actual time=0.893..0.893 rows=1 loops=1)
                                -> Covering index scan on g using GPUCostIndex  (cost=265.25 rows=2585) (actual time=0.028..0.592 rows=2585 loops=1)
 |
```

COST = 87545.46

Index: Manufacturer.BrandName

```
| -> Limit: 15 row(s)  (actual time=315.528..315.528 rows=1 loops=1)
    -> Sort: Manufacturer.YearEstablished DESC, limit input to 15 row(s) per chunk  (actual time=315.528..315.528 rows=1 loops=1)
        -> Table scan on <temporary>  (cost=87545.46..91692.02 rows=331526) (actual time=315.512..315.512 rows=1 loops=1)
            -> Temporary table with deduplication  (cost=87545.45..87545.45 rows=331526) (actual time=315.510..315.510 rows=1 loops=1)
                -> Nested loop inner join  (cost=54392.82 rows=331526) (actual time=0.762..284.008 rows=98499 loops=1)
                    -> Nested loop inner join  (cost=463.70 rows=1026) (actual time=0.075..1.302 rows=1026 loops=1)
                        -> Table scan on CPU  (cost=104.60 rows=1026) (actual time=0.058..0.659 rows=1026 loops=1)
                        -> Single-row index lookup on Manufacturer using PRIMARY (BrandName=`CPU`.Brand)  (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=1026)
                    -> Filter: ((`CPU`.Cost > (select #2)) or (GPU.Cost > (select #3)))  (cost=20.28 rows=323) (actual time=0.182..0.268 rows=96 loops=1026)
                        -> Index lookup on GPU using Brand (Brand=`CPU`.Brand)  (cost=20.28 rows=323) (actual time=0.180..0.237 rows=122 loops=1026)
                        -> Select #2 (subquery in condition; run only once)
                            -> Aggregate: avg(c.Cost)  (cost=207.20 rows=1) (actual time=0.449..0.449 rows=1 loops=1)
                                -> Table scan on c  (cost=104.60 rows=1026) (actual time=0.072..0.328 rows=1026 loops=1)
                        -> Select #3 (subquery in condition; run only once)
                            -> Aggregate: avg(g.Cost)  (cost=523.75 rows=1) (actual time=1.045..1.045 rows=1 loops=1)
                                -> Table scan on g  (cost=265.25 rows=2585) (actual time=0.032..0.715 rows=2585 loops=1)
 |
```

COST = 87545.56

Justification

When indexing on Manufacturer.BrandName, the performance boost is limited in reach. In the inner join clauses, Manufacturer.BrandName is joined onto CPU.BrandName and GPU.BrandName, but only the former is indexed, whereas the latter pair is not. Thus the time complexity and computational cost remain almost the same despite the index. In applying the indices on GPU.Cost and CPU.Cost, the performance is held back by the inner joins described previously, which are now at their worst performance because there is no index on Manufacturer.BrandName. Finally, when both groups of indices (the indices on Cost and the index on BrandName) are applied, the performance still suffers for the same reason described for the index on BrandName alone.

Query 4:

Select all users which own a cart with at least one US-based processor

```
(SELECT
        Person.Username
FROM
        Person
INNER JOIN
        Permission
        ON Person.Username = Permission.Username
INNER JOIN
        Cart
        ON Permission.CartId = Cart.CartId
INNER JOIN
        CPU
        ON Cart.CPUName = CPU.ProductName
INNER JOIN
        Manufacturer
        ON CPU.Brand = Manufacturer.BrandName
WHERE
        Manufacturer.CountryCode = 'US'
        AND Permission.Scope = 'manage'
LIMIT 15
) UNION (
```

```
SELECT
	Person.Username
FROM
	Person
INNER JOIN
	Permission
	ON Person.Username = Permission.Username
INNER JOIN
	Cart
	ON Permission.CartId = Cart.CartId
INNER JOIN
	GPU
	ON Cart.GPUName = GPU.ProductName
INNER JOIN
	Manufacturer
	ON GPU.Brand = Manufacturer.BrandName
WHERE
	Manufacturer.CountryCode = 'US'
	AND Permission.Scope = 'manage'
LIMIT 15
);
```

```
+------------------+
| Username         |
+------------------+
| jet              |
| KevinKennedy626  |
| EthanFlores216   |
+------------------+
```

Non-indexed Cost

```
| -> Table scan on <union temporary>  (cost=5.81..5.81 rows=0.3) (actual time=0.179..0.179 rows=3 loops=1)
    -> Union materialize with deduplication  (cost=3.31..3.31 rows=0.3) (actual time=0.178..0.178 rows=3 loops=1)
        -> Limit: 15 row(s)  (cost=1.64 rows=0.1) (actual time=0.074..0.113 rows=3 loops=1)
            -> Nested loop inner join  (cost=1.64 rows=0.1) (actual time=0.073..0.112 rows=3 loops=1)
                -> Nested loop inner join  (cost=1.60 rows=0.1) (actual time=0.065..0.093 rows=3 loops=1)
                    -> Nested loop inner join  (cost=1.25 rows=1) (actual time=0.059..0.084 rows=3 loops=1)
                        -> Nested loop inner join  (cost=0.90 rows=1) (actual time=0.041..0.048 rows=3 loops=1)
                            -> Filter: (Permission.Scope = 'manage')  (cost=0.55 rows=1) (actual time=0.027..0.030 rows=3 loops=1)
                                -> Table scan on Permission  (cost=0.55 rows=3) (actual time=0.023..0.025 rows=3 loops=1)
                            -> Filter: (Cart.CPUName is not null)  (cost=0.35 rows=1) (actual time=0.005..0.006 rows=1 loops=3)
                                -> Single-row index lookup on Cart using PRIMARY (CartId=Permission.CartId)  (cost=0.35 rows=1) (actual time=0.005..0.005 rows=1 loops=3)
                        -> Covering index lookup on CPU using PRIMARY (ProductName=Cart.CPUName)  (cost=0.35 rows=1) (actual time=0.010..0.012 rows=1 loops=3)
                    -> Filter: (Manufacturer.CountryCode = 'US')  (cost=0.26 rows=0.1) (actual time=0.003..0.003 rows=1 loops=3)
                        -> Single-row index lookup on Manufacturer using PRIMARY (BrandName=`CPU`.Brand)  (cost=0.26 rows=1) (actual time=0.002..0.002 rows=1 loops=3)
                -> Single-row covering index lookup on Person using PRIMARY (Username=Permission.Username)  (cost=1.05 rows=1) (actual time=0.006..0.006 rows=1 loops=3)
        -> Limit: 15 row(s)  (cost=1.64 rows=0.1) (actual time=0.025..0.054 rows=3 loops=1)
            -> Nested loop inner join  (cost=1.64 rows=0.1) (actual time=0.025..0.054 rows=3 loops=1)
                -> Nested loop inner join  (cost=1.60 rows=0.1) (actual time=0.021..0.043 rows=3 loops=1)
                    -> Nested loop inner join  (cost=1.25 rows=1) (actual time=0.018..0.038 rows=3 loops=1)
                        -> Nested loop inner join  (cost=0.90 rows=1) (actual time=0.007..0.013 rows=3 loops=1)
                            -> Filter: (Permission.Scope = 'manage')  (cost=0.55 rows=1) (actual time=0.004..0.006 rows=3 loops=1)
                                -> Table scan on Permission  (cost=0.55 rows=3) (actual time=0.004..0.005 rows=3 loops=1)
                            -> Filter: (Cart.GPUName is not null)  (cost=0.35 rows=1) (actual time=0.002..0.002 rows=1 loops=3)
                                -> Single-row index lookup on Cart using PRIMARY (CartId=Permission.CartId)  (cost=0.35 rows=1) (actual time=0.002..0.002 rows=1 loops=3)
                        -> Covering index lookup on GPU using PRIMARY (ProductName=Cart.GPUName)  (cost=0.35 rows=1) (actual time=0.007..0.008 rows=1 loops=3)
                    -> Filter: (Manufacturer.CountryCode = 'US')  (cost=0.26 rows=0.1) (actual time=0.001..0.001 rows=1 loops=3)
                        -> Single-row index lookup on Manufacturer using PRIMARY (BrandName=GPU.Brand)  (cost=0.26 rows=1) (actual time=0.001..0.001 rows=1 loops=3)
                -> Single-row covering index lookup on Person using PRIMARY (Username=Permission.Username)  (cost=1.05 rows=1) (actual time=0.003..0.003 rows=1 loops=3)
 |
```

Cost: 5.81

INDEX ON Manufacturer.CountryCode

```
| -> Table scan on <union temporary>  (cost=6.16..6.16 rows=1) (actual time=0.176..0.176 rows=3 loops=1)
    -> Union materialize with deduplication  (cost=3.65..3.65 rows=1) (actual time=0.175..0.175 rows=3 loops=1)
        -> Limit: 15 row(s)  (cost=1.78 rows=1) (actual time=0.066..0.108 rows=3 loops=1)
            -> Nested loop inner join  (cost=1.78 rows=1) (actual time=0.065..0.106 rows=3 loops=1)
                -> Nested loop inner join  (cost=1.60 rows=1) (actual time=0.056..0.085 rows=3 loops=1)
                    -> Nested loop inner join  (cost=1.25 rows=1) (actual time=0.047..0.074 rows=3 loops=1)
                        -> Nested loop inner join  (cost=0.90 rows=1) (actual time=0.029..0.037 rows=3 loops=1)
                            -> Filter: (Permission.Scope = 'manage')  (cost=0.55 rows=1) (actual time=0.019..0.021 rows=3 loops=1)
                                -> Table scan on Permission  (cost=0.55 rows=3) (actual time=0.017..0.019 rows=3 loops=1)
                            -> Filter: (Cart.CPUName is not null)  (cost=0.35 rows=1) (actual time=0.004..0.005 rows=1 loops=3)
                                -> Single-row index lookup on Cart using PRIMARY (CartId=Permission.CartId)  (cost=0.35 rows=1) (actual time=0.004..0.004 rows=1 loops=3)
                        -> Covering index lookup on CPU using PRIMARY (ProductName=Cart.CPUName)  (cost=0.35 rows=1) (actual time=0.010..0.012 rows=1 loops=3)
                    -> Filter: (Manufacturer.CountryCode = 'US')  (cost=0.30 rows=0.5) (actual time=0.003..0.003 rows=1 loops=3)
                        -> Single-row index lookup on Manufacturer using PRIMARY (BrandName=`CPU`.Brand)  (cost=0.30 rows=1) (actual time=0.003..0.003 rows=1 loops=3)
                -> Single-row covering index lookup on Person using PRIMARY (Username=Permission.Username)  (cost=0.45 rows=1) (actual time=0.007..0.007 rows=1 loops=3)
        -> Limit: 15 row(s)  (cost=1.78 rows=1) (actual time=0.028..0.057 rows=3 loops=1)
            -> Nested loop inner join  (cost=1.78 rows=1) (actual time=0.028..0.057 rows=3 loops=1)
                -> Nested loop inner join  (cost=1.60 rows=1) (actual time=0.024..0.045 rows=3 loops=1)
                    -> Nested loop inner join  (cost=1.25 rows=1) (actual time=0.020..0.039 rows=3 loops=1)
                        -> Nested loop inner join  (cost=0.90 rows=1) (actual time=0.007..0.012 rows=3 loops=1)
                            -> Filter: (Permission.Scope = 'manage')  (cost=0.55 rows=1) (actual time=0.003..0.005 rows=3 loops=1)
                                -> Table scan on Permission  (cost=0.55 rows=3) (actual time=0.003..0.004 rows=3 loops=1)
                            -> Filter: (Cart.GPUName is not null)  (cost=0.35 rows=1) (actual time=0.002..0.002 rows=1 loops=3)
                                -> Single-row index lookup on Cart using PRIMARY (CartId=Permission.CartId)  (cost=0.35 rows=1) (actual time=0.002..0.002 rows=1 loops=3)
                        -> Covering index lookup on GPU using PRIMARY (ProductName=Cart.GPUName)  (cost=0.35 rows=1) (actual time=0.008..0.009 rows=1 loops=3)
                    -> Filter: (Manufacturer.CountryCode = 'US')  (cost=0.30 rows=0.5) (actual time=0.001..0.001 rows=1 loops=3)
                        -> Single-row index lookup on Manufacturer using PRIMARY (BrandName=GPU.Brand)  (cost=0.30 rows=1) (actual time=0.001..0.001 rows=1 loops=3)
                -> Single-row covering index lookup on Person using PRIMARY (Username=Permission.Username)  (cost=0.45 rows=1) (actual time=0.004..0.004 rows=1 loops=3)
 |
```

Cost: 6.16

INDEX ON Permission.Scope

```
| -> Table scan on <union temporary>  (cost=7.82..7.82 rows=0.5) (actual time=0.279..0.280 rows=3 loops=1)
    -> Union materialize with deduplication  (cost=5.32..5.32 rows=0.5) (actual time=0.278..0.278 rows=3 loops=1)
        -> Limit: 15 row(s)  (cost=2.64 rows=0.2) (actual time=0.073..0.162 rows=3 loops=1)
            -> Nested loop inner join  (cost=2.64 rows=0.2) (actual time=0.072..0.134 rows=3 loops=1)
                -> Nested loop inner join  (cost=2.55 rows=0.2) (actual time=0.056..0.102 rows=3 loops=1)
                    -> Nested loop inner join  (cost=1.85 rows=2) (actual time=0.047..0.089 rows=3 loops=1)
                        -> Nested loop inner join  (cost=1.15 rows=2) (actual time=0.028..0.044 rows=3 loops=1)
                            -> Filter: (Cart.CPUName is not null)  (cost=0.45 rows=2) (actual time=0.014..0.018 rows=3 loops=1)
                                -> Covering index scan on Cart using CPUName  (cost=0.45 rows=2) (actual time=0.012..0.016 rows=3 loops=1)
                            -> Filter: (Permission.Scope = 'manage')  (cost=0.30 rows=1) (actual time=0.007..0.008 rows=1 loops=3)
                                -> Index lookup on Permission using PRIMARY (CartId=Cart.CartId)  (cost=0.30 rows=1) (actual time=0.005..0.006 rows=1 loops=3)
                        -> Covering index lookup on CPU using PRIMARY (ProductName=Cart.CPUName)  (cost=0.30 rows=1) (actual time=0.013..0.014 rows=1 loops=3)
                    -> Filter: (Manufacturer.CountryCode = 'US')  (cost=0.26 rows=0.1) (actual time=0.004..0.004 rows=1 loops=3)
                        -> Single-row index lookup on Manufacturer using PRIMARY (BrandName=`CPU`.Brand)  (cost=0.26 rows=1) (actual time=0.003..0.003 rows=1 loops=3)
                -> Single-row covering index lookup on Person using PRIMARY (Username=Permission.Username)  (cost=0.65 rows=1) (actual time=0.010..0.010 rows=1 loops=3)
        -> Limit: 15 row(s)  (cost=2.64 rows=0.2) (actual time=0.055..0.099 rows=3 loops=1)
            -> Nested loop inner join  (cost=2.64 rows=0.2) (actual time=0.054..0.098 rows=3 loops=1)
                -> Nested loop inner join  (cost=2.55 rows=0.2) (actual time=0.047..0.080 rows=3 loops=1)
                    -> Nested loop inner join  (cost=1.85 rows=2) (actual time=0.039..0.070 rows=3 loops=1)
                        -> Nested loop inner join  (cost=1.15 rows=2) (actual time=0.016..0.026 rows=3 loops=1)
                            -> Filter: (Cart.GPUName is not null)  (cost=0.45 rows=2) (actual time=0.009..0.011 rows=3 loops=1)
                                -> Covering index scan on Cart using GPUName  (cost=0.45 rows=2) (actual time=0.008..0.010 rows=3 loops=1)
                            -> Filter: (Permission.Scope = 'manage')  (cost=0.30 rows=1) (actual time=0.004..0.005 rows=1 loops=3)
                                -> Index lookup on Permission using PRIMARY (CartId=Cart.CartId)  (cost=0.30 rows=1) (actual time=0.003..0.004 rows=1 loops=3)
                        -> Covering index lookup on GPU using PRIMARY (ProductName=Cart.GPUName)  (cost=0.30 rows=1) (actual time=0.013..0.014 rows=1 loops=3)
                    -> Filter: (Manufacturer.CountryCode = 'US')  (cost=0.26 rows=0.1) (actual time=0.003..0.003 rows=1 loops=3)
                        -> Single-row index lookup on Manufacturer using PRIMARY (BrandName=GPU.Brand)  (cost=0.26 rows=1) (actual time=0.002..0.002 rows=1 loops=3)
                -> Single-row covering index lookup on Person using PRIMARY (Username=Permission.Username)  (cost=0.65 rows=1) (actual time=0.005..0.005 rows=1 loops=3)
 |
```

Cost: 7.82

INDEX ON Manufacturer.CountryCode AND Permission.Scope

```
| -> Table scan on <union temporary>  (cost=7.82..7.82 rows=0.5) (actual time=0.279..0.280 rows=3 loops=1)
    -> Union materialize with deduplication  (cost=5.32..5.32 rows=0.5) (actual time=0.278..0.278 rows=3 loops=1)
        -> Limit: 15 row(s)  (cost=2.64 rows=0.2) (actual time=0.073..0.162 rows=3 loops=1)
            -> Nested loop inner join  (cost=2.64 rows=0.2) (actual time=0.072..0.134 rows=3 loops=1)
                -> Nested loop inner join  (cost=2.55 rows=0.2) (actual time=0.056..0.102 rows=3 loops=1)
                    -> Nested loop inner join  (cost=1.85 rows=2) (actual time=0.047..0.089 rows=3 loops=1)
                        -> Nested loop inner join  (cost=1.15 rows=2) (actual time=0.028..0.044 rows=3 loops=1)
                            -> Filter: (Cart.CPUName is not null)  (cost=0.45 rows=2) (actual time=0.014..0.018 rows=3 loops=1)
                                -> Covering index scan on Cart using CPUName  (cost=0.45 rows=2) (actual time=0.012..0.016 rows=3 loops=1)
                            -> Filter: (Permission.Scope = 'manage')  (cost=0.30 rows=1) (actual time=0.007..0.008 rows=1 loops=3)
                                -> Index lookup on Permission using PRIMARY (CartId=Cart.CartId)  (cost=0.30 rows=1) (actual time=0.005..0.006 rows=1 loops=3)
                        -> Covering index lookup on CPU using PRIMARY (ProductName=Cart.CPUName)  (cost=0.30 rows=1) (actual time=0.013..0.014 rows=1 loops=3)
                    -> Filter: (Manufacturer.CountryCode = 'US')  (cost=0.26 rows=0.1) (actual time=0.004..0.004 rows=1 loops=3)
                        -> Single-row index lookup on Manufacturer using PRIMARY (BrandName=`CPU`.Brand)  (cost=0.26 rows=1) (actual time=0.003..0.003 rows=1 loops=3)
                -> Single-row covering index lookup on Person using PRIMARY (Username=Permission.Username)  (cost=0.65 rows=1) (actual time=0.010..0.010 rows=1 loops=3)
        -> Limit: 15 row(s)  (cost=2.64 rows=0.2) (actual time=0.055..0.099 rows=3 loops=1)
            -> Nested loop inner join  (cost=2.64 rows=0.2) (actual time=0.054..0.098 rows=3 loops=1)
                -> Nested loop inner join  (cost=2.55 rows=0.2) (actual time=0.047..0.080 rows=3 loops=1)
                    -> Nested loop inner join  (cost=1.85 rows=2) (actual time=0.039..0.070 rows=3 loops=1)
                        -> Nested loop inner join  (cost=1.15 rows=2) (actual time=0.016..0.026 rows=3 loops=1)
                            -> Filter: (Cart.GPUName is not null)  (cost=0.45 rows=2) (actual time=0.009..0.011 rows=3 loops=1)
                                -> Covering index scan on Cart using GPUName  (cost=0.45 rows=2) (actual time=0.008..0.010 rows=3 loops=1)
                            -> Filter: (Permission.Scope = 'manage')  (cost=0.30 rows=1) (actual time=0.004..0.005 rows=1 loops=3)
                                -> Index lookup on Permission using PRIMARY (CartId=Cart.CartId)  (cost=0.30 rows=1) (actual time=0.003..0.004 rows=1 loops=3)
                        -> Covering index lookup on GPU using PRIMARY (ProductName=Cart.GPUName)  (cost=0.30 rows=1) (actual time=0.013..0.014 rows=1 loops=3)
                    -> Filter: (Manufacturer.CountryCode = 'US')  (cost=0.26 rows=0.1) (actual time=0.003..0.003 rows=1 loops=3)
                        -> Single-row index lookup on Manufacturer using PRIMARY (BrandName=GPU.Brand)  (cost=0.26 rows=1) (actual time=0.002..0.002 rows=1 loops=3)
                -> Single-row covering index lookup on Person using PRIMARY (Username=Permission.Username)  (cost=0.65 rows=1) (actual time=0.005..0.005 rows=1 loops=3)
 |
```

Cost: 7.26

Justification

Since the query appeared to run the fastest when no indices were provided, we decided to forgo using indices for the fourth query. It is entirely possible that adding indices slowed down the data retrieval because the relations are small compared to the larger relations. Because Permission has so few rows, reducing the number of reads would have virtually no effect while also resulting in unnecessary overhead from having the indices. Manufacturer has only eight rows since there can only be so many manufacturers, and Permission has only three rows since adding a significant number of rows may be time-consuming. We may look into automating the process of auto-generating cart configurations, so that the cost of having indices on the relations can be cheaper than the cost of retrieving non-indexed data.
