Milestone 4 - Ryan Inghilterra - Sales

Precomputed tables

```
CREATE TABLE customer_category_sales AS
   SELECT customer_id, category_id, SUM(s.quantity) as total_quantity,
SUM(s.quantity*s.price) as total_dollar
   FROM product p LEFT OUTER JOIN sale s
   ON p.product_id = s.product_id
   GROUP BY customer_id, category_id


 CREATE TABLE customer_total_sales AS
    SELECT
      s.customer_id,
      SUM(s.quantity * s.price) AS total_dollar
    FROM product p LEFT OUTER JOIN sale s
       ON p.product_id = s.product_id
    GROUP BY customer_id;


 CREATE TABLE category_total_sales AS
      SELECT
        p.category_id,
        SUM(s.quantity * s.price) AS total_dollar
      FROM product p LEFT OUTER JOIN sale s
         ON p.product_id = s.product_id
      GROUP BY category_id;
```

The cost of maintenance would be to update these tables any time there is a sale added, removed, or updated. This cost could be reasonable, as you can have a trigger that updates the values in these tables every time there is a sale. This could be a lot less expensive than if you had to recompute the entire table every time there was a new sale.

New Query 6 with the 3 precompute tables

```
WITH top_20_customers AS (
        SELECT *
  FROM customer_total_sales
  ORDER BY total_dollar DESC LIMIT 20
),
top_20_categories AS (
        SELECT *
  FROM category_total_sales
  ORDER BY total_dollar DESC LIMIT 20
),
t1 AS (
        SELECT c2.category_id, c1.customer_id
    FROM top_20_customers c1 CROSS JOIN top_20_categories c2
    ORDER BY c2.category_id, c1.customer_id
),
t3 AS (
        SELECT t1.category_id, t1.customer_id, COALESCE(t2.total_quantity,0)
    AS total_quantity, COALESCE(t2.total_dollar,0) AS total_dollar
    FROM t1 LEFT OUTER JOIN customer_category_sales t2
    ON t1.category_id = t2.category_id AND t1.customer_id = t2.customer_id
)
SELECT * FROM t3;
```

Updated Query 6 Cost:
CTE Scan on t3 (cost=21650.70..21658.70 rows=400 width=48) (actual time=192.725..246.019 rows=400 loops=1)

Compared to original Query 6 Cost
CTE Scan on t3 (cost=335428.95..335560.61 rows=6583 width=48) (actual time=2734.929..3675.620 rows=400 loops=1)

A huge improvement in performance! A large part of the cost in the original Query 6 came from the groupAggregates. The precomputed tables I added already have the groupAggregates done, so the new Query 6 does not have to do the groupAggregates, making the query cost a lot lower.

Now lets add a helpful index on our precomputed table customer_category_sales

CREATE INDEX s6 ON customer_category_sales (category_id, customer_id);

Updated Cost of Query 6 With Index on precomputed table customer_category_sales
CTE Scan on t3 (cost=3540.50..3548.50 rows=400 width=48) (actual time=2.921..6.249 rows=400 loops=1)
So even a further performance improvement when adding this index