Question 1ev: Explain your answer to the previous part based on your knowledge from lectures, and details from the query plans (your explanation should include why you didn't choose certain options).

Views will reduce the execution time of a query because they'll store temporary results that can speed up resulting queries. However, since the views store results, they are still increasing the cost. Materialized views reduce execution time and the cost of a query because every time the database runs its query, it will only run it once and store the results in a newly created database. In a materialized view, it does not recalculate the results every time the query is run because the values are stored, which makes the cost less and the runtime faster. Since storing values, a materialized view takes longer to create than a standard view.

Question 2d: Explain your answer based on your knowledge from lectures, and details from the query plans (your explanation should include why you didn't choose certain options).

Adding a filter lowered the cost because it only selected a small amount of data that it had to go through which made the query run faster. In question 2, when we wrote a query to get the all the rows with no filter, the cost was over 500 and the runtime was 15. When we added the view, it had a smaller amount of data to run through which means that the cost was lower and the runtime was much lower.

0.0.1 Question 3d

Please discuss your findings in the previous parts. In particular, we are interested in hearing why you think the query optimizer chooses the ultimate join approach in each of the above three scenarios. Feel free to discuss the pros and cons of each join approach as well.

If you feel stuck, here are some things to consider: Does a non-equijoin constrain us to certain join approaches? What's an added benefit in regards to the output of merge join?

In question 3a, the query optimizer used a hash join because a hash joined is used on tables that are already not sorted. In this question, we only selected columns to do an inner join but neither table had sorted columns. The hash join was used to sort the data from the two tables and then joins the two tables. In question 3b, a merge join was used because the two tables are already being sorted by the "order by" method. The merge join is better than the hash join in this sceanario because the two tables are already being sorted and if the hash method was used then the excecution time and the cost would be much higher. The merge method was the fastest method to use in this case since the two tables are sorted. In question 3c, the nested loop join was used because the table is smaller and you are cross joining in the same table. The nested loop join is the fastest join and least costly in this situation because there is an inner join in the same table. The hash join and the merge join is unnesscary in this case because there is no need to sort values or merge two large quantity of tables. Since the data in one table is small and there is a cross join the nested for loop is the best one to use.

Question 4dii: Explain your answer based on your knowledge from lectures, and details from inspecting the query plans (your explanation should include why you didn't choose certain options).

Adding an index can make the execution time either become really fast or slow. The index can make it become a pointers of where the data is stored in the tables which makes it faster to join. Indexes can also make the execution time slower since it has more indexes, the query would have to do more processing that would increase the cost. Adding indexes can make the costs go higher since it is creating more data and it makes the query go longer

Question 4evii Explain your answer to the previous part based on your knowledge from lectures, and details of the query plans (your explanation should include why you didn't choose certain options).

Adding an index can make the execution time either become really fast or slow. The index can make it become a pointers of where the data is stored in the tables which makes it faster to join. Indexes can also make the execution time slower since it has more indexes, the query would have to do more processing that would increase the cost. Adding indexes can make the costs go higher since it is creating more data and it makes the query go longer. Adding predicates can make the execution time faster since it is specify a certain type of data. Adding a predicate can cost the query more since it is processing the predicate but since it is picking a certain data and not looking through the whole table, the query time would be smaller.

0.0.2 Question 5d:

Explain your answer to the previous part based on your knowledge from lectures, and your inspection of the query plans.

Using count will benefit because it is just going through the whole table and counting rows rather than AVG where it going through the whole table and using a aggregating through the whole table. The AVG function is taking longer to process and will cost more.

0.0.3 Question 6e:

Explain your answer to the previous part based on your knowledge from lectures, and your inspection of the query plans (your explanation should include why you didn't choose certain options).

Clustering data makes the cost of the query go down because it reduces the amount of data processed by queries. The cost goes down because there isn't much data being processed. Since there is less data being processed, the execution time is shorter since there is a smaller data set to process.

0.0.4 Question 7c:

What difference did you notice when you added an index into the salaries table and re-timed the update? Why do you think it happened?

I noticed that the time doubled by creating an index. I think the time doubled because since an index was added it took longer for script to excute since there was more data.

0.1 Question 8: Takeaway

In this project, we explored how the database system optimizes query execution and how users can futher tune the performance of their queries.

Familiarizing yourself with these optimization and tuning methods will make you a better data engineer. In this question, we'll ask you to recall and summarize these concepts. Who knows? Maybe one day it will help you during an interview or on a project.

In the following answer cell, 1. Name 3 methods you learned in this project. The method can be either the optimization done by the database system, or the fine tuning done by the user. 2. For each method, summarize how and why it can optimize query performance. Feel free to discuss any drawbacks, if applicable.

I learned about when tables will use hash joins, nested loop joins, and merge sorts. Tables will use hash joins if you are naturally joining two tables, merge sorts if the two tables are already being sorted, and a nested loop join if you are working on a small amount of data. I also learned that adding indexes can increase the cost and executation time greatly because you making the query process through more data. I have also learned that if you want to reduce costs and the query time, using clustering is the best way to decrease it.