Question 2dii: Explain your answer to the previous part (Question 2di) based on your knowledge from lectures and details from the query plans. Your explanation should also include why you didn't choose certain options. Please answer in maximum 5 sentences.

The primary distinction between a View and a Materialized View is that the latter stores its output on disk, while the former does not. As a result, options 'A', 'B', and 'C' can be dismissed since a view, not being stored on disk, doesn't alter the cost or runtime of a query. In contrast, a materialized view enhances performance by reducing both cost and runtime, as it provides pre-computed data from the disk, eliminating the need to fetch data from scratch. This understanding negates options 'E' and 'F'. Furthermore, since creating a materialized view involves disk I/O operations, it inherently takes longer than creating a regular view, validating option 'J'.

0.1 Question 3c:

Given your findings from inspecting the query plans of queries from Questions 3a and 3b, fill in the blank and **justify your answer**. Explain your answer based on your knowledge from lectures, and details from the query plans (your explanation should include why you didn't choose other options). Your response should be no longer than 3 sentences.

Note: Your answer should be formatted as follows: `A because ...`

Adding a filter _____ the cost. A. increased B. decreased C. did not change

B. decreased because the cost of executing the query with the filter (209.85) is significantly lower than the cost without the filter (528.30). The filter effectively narrows down the number of rows to be processed, leading to a more efficient query execution.

0.2 Question 3d:

Given your findings from inspecting the query plans of queries from Questions 3a and 3b, fill in the blank and **justify your answer**. Explain your answer based on your knowledge from lectures, and details from the query plans (your explanation should include why you didn't choose other options). Your response should be no longer than 3 sentences.

Note: Your answer should be formatted as follows: `A because ...`

Adding a filter _____ the execution time. A. increased B. decreased C. did not change

B. decreased because the execution time with the filter (0.735 ms) is notably shorter than the execution time without the filter (4.451 ms). By adding the filter, the database processes fewer rows, leading to a faster query execution.

0.3 Question 4d

Given your findings above, why did the query optimizer ultimately choose the specific join approach you found in each of the above three scenarios in Questions 4a, 4b, and 4c? 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?

Note: Your answer should be formatted as follows: `Q4a: A because ... Q4b: A because ...` You should write no more than 5 sentences.

Q4a: The optimizer selected a Hash Join due to the large number of rows and equality-based join condition, leveraging the efficiency of hash tables for such scenarios.

Q4b: A Merge Join was chosen because of the sort requirement on playerid and the pre-sorted output from an index scan.

Q4c: The Nested Loop Join was favored due to the inequality join condition (p1.playerid <> p2.playerid), which is typically handled best by nested loops

0.3.1 Question 5di Justification

Explain your answer to `Question 5d` above based on your knowledge from lectures, and details from inspecting the query plans (your explanation should include why you didn't choose certain options). Your answer should be no longer than 3 sentences.

The g_batting index notably improved the query's efficiency by enabling a Bitmap Index Scan, eliminating the need for a full scan on the appearances table. In contrast, the salary index didn't enhance the query's performance, as the plan still used a sequential scan on the salaries table, with metrics similar to those without the index.

0.3.2 Question 6ei Justification

Explain your answer to `Question 6e` above based on your knowledge from lectures, and details from inspecting the query plans (your explanation should include why you didn't choose certain options). Your answer should be no longer than 3 sentences.

Adding an index on g_batting for an AND predicate reduces both execution time and cost due to efficient data retrieval in the B+ tree structure, eliminating options 'A' and 'B'. Indexing a column in an OR predicate doesn't guarantee efficiency since all records must be verified for the second condition, ruling out "D", "E", and "F". A multicolumn index on g_batting and g_all optimizes execution time, especially when the query involves the leftmost column, g_batting, making 'G' the correct choice.

0.4 Question 7c

Given your findings from `Question 7`, which of the following statements is true? A. An index on the column being aggregated in a query will always provide a performance enhancement. B. A query finding the MIN(salary) will always benefit from an index on salary, but a query finding MAX(salary) will not. C. A query finding the COUNT(salary) will always benefit from an index on salary, but a query finding AVG(salary) will not. D. Queries finding the MIN(salary) or MAX(salary) will always benefit from an index on salary, but queries finding AVG(salary) or COUNT(salary) will not.

Justify your answer. Explain your answer based on your knowledge from lectures, and details of the query plans (your explanation should include why you didn't choose certain options). Your response should be no longer than 3 sentences.

Note: Your answer should be formatted as follows: "A because …"

D because the query finding the MIN(salary) benefited from the index on the salary column, as evidenced by the shift from a sequential scan to an index-only scan, significantly reducing execution time and cost. However, the query finding AVG(salary) still relied on a sequential scan, showing no performance improvement with the index. This suggests that while MIN and MAX operations can leverage indexes for optimization, AVG and COUNT operations might not always benefit in the same way.

0.5 Question 9c:

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? Your answer should be no longer than 3 sentences.

After adding an index, the insertion time noticeably increased. This is likely because every new data insertion requires the database to maintain the order of the indexed values. For instance, if a new value needs to be placed between existing indexed data, the system must first insert the new data and then adjust subsequent indexed values accordingly, leading to additional time overhead.

1 Question 10: Project Takeaways

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.

Your answer should be no longer than ten sentences. Each method identification/discussion is 2 points.

Index Creation and Utilization: Through this project, I grasped the transformative impact of indexes on query performance. Without an index, the database performs a sequential scan, sifting through each row to find the desired data. However, with an indexed column, the data is organized in a B+tree structure, allowing for direct and efficient access to the required rows. This structured arrangement ensures that rows with similar index values are adjacent, drastically reducing query cost and execution time.

Subqueries and Their Impact: I discovered that while subqueries can be powerful, they often come at a performance cost. Each subquery essentially reads datasets from the beginning, increasing the overall execution time. To optimize, it's beneficial to consider alternatives like using the 'WITH' clause or joining tables on essential columns. The key takeaway is the importance of minimizing the size of the datasets being worked on, as larger datasets inherently demand more resources.

Understanding Predicate Effects on Indexing: The project highlighted the nuanced relationship between query predicates ("OR" vs. "AND") and index effectiveness. It's crucial to recognize that an index might not always optimize performance, especially in queries with "OR" predicates. Being aware of these intricacies ensures that we apply indexing judiciously, maximizing its benefits.