Matthew Mulenga

mam558

11144528

**Part 1**

Index 1

|  |  |  |
| --- | --- | --- |
| Execution Time | Without Index (ms) | With Index (ms) |
| Query 1 | P: 0.127 | E: 1.189 | P: 0.552 | E: 2.653 |
| Query 2 | P: 0.100 | E: 1.799 | P: 0.314 | E: 3.365 |

1. Adding the index increased both the planning and execution timings for both queries.
2. Adding this index did not change the explain plan. A sequential scan was done for both queries 1 and 2, both with and without the index. The filter remained the same on both queries as well.
3. Yes, this is what was expected because the index created uses both the first and last name columns, whereas we’re only using the first name column in our search. Meaning our index will be ignored because it’s less efficient and the execution plan should remain the same.

Index 2

|  |  |  |
| --- | --- | --- |
| Execution Time | Without Index (ms) | With Index (ms) |
| Query 3 | P: 0.319 | E: 2.037 | P: 0.554 | E: 0.514 |

1. Adding the index increased the planning timing, but reduced the execution timing.
2. The index changed the execution plan. Instead of doing a sequential scan on employees, the execution plan does a bitmap index scan on the index we created, followed by a bitmap heap scan on employees.
3. Yes, this is what was expected, the planning time should increase because there are more instructions to be parsed and the execution time should decrease because the index is on a column that is used in a join and has lots of different values. The execution plan changed because it’s faster to use the provided index than to sequentially scan.

Index 3

|  |  |  |
| --- | --- | --- |
| Execution Time | Without Index (ms) | With Index (ms) |
| Query 4 | P: 1.180 | E: 23.184 | P: 1.133 | E: 6.502 |

1. Adding the index doesn’t appear to have changed the planning timing by a significant amount, however the execution timing was drastically reduced.
2. The index changed the plan. Within the subplan1 section of the execution plan, the plan changes from using a sequential scan on employee\_jobs, with the filter (employee\_id = e.id) to a index scan of our created index with an index condition equal to the filter above. Within the join section of the main plan the plan switches from a sequential scan of employee\_jobs, with the filter (job\_id = j.id) to an index scan of our created index, with an index condition (job\_id = j.id).
3. I did not expect the planning time to remain essentially the same, I thought it would increase slightly. I did expect the execution time to reduce quite a bit, because we’re eliminating the need to go through the entire employee\_jobs table when we’re searching for matches of job IDs and employee IDs.

**Part 2**

**1st Normal Form**

**2nd Normal Form (Red = primary key)**

Step 1

Step 2





**3rd Normal Form (After following steps from 2NF)**





**Part 3**

**Question 1**

a)

|  |  |  |
| --- | --- | --- |
| Account Number | Account Nickname | Account Balance |
| 1 | Chequing | 350 |
| 2 | Chequing | 100 |

b)

*Lost Update* because Transaction B overwrites the data of Transaction A after Transaction A updates with it’s own update.

*Dirty Read* because Transaction B reads the uncommitted data of Transaction A after Transaction A updates.

**Question 2**

A *Non-repeatable Read* data inconsistency is caused because Transaction C reads all account information from the database. Then Transaction D inserts a new record and updates account 1’s balance and finally Transaction C rereads the data all before committing.

x

**Question 3**

A *Dirty Read* data inconsistency is caused because Transaction F reads the accounts before Transaction E commits (in this case rolls back).