## Task 3

The query was:

Select count(distinct(employees.emp_no)) from salaries, employees where employees.emp_no = salaries.emp_no and salaries > 50000 and salaries < 70000 and to_date - from_date < 1

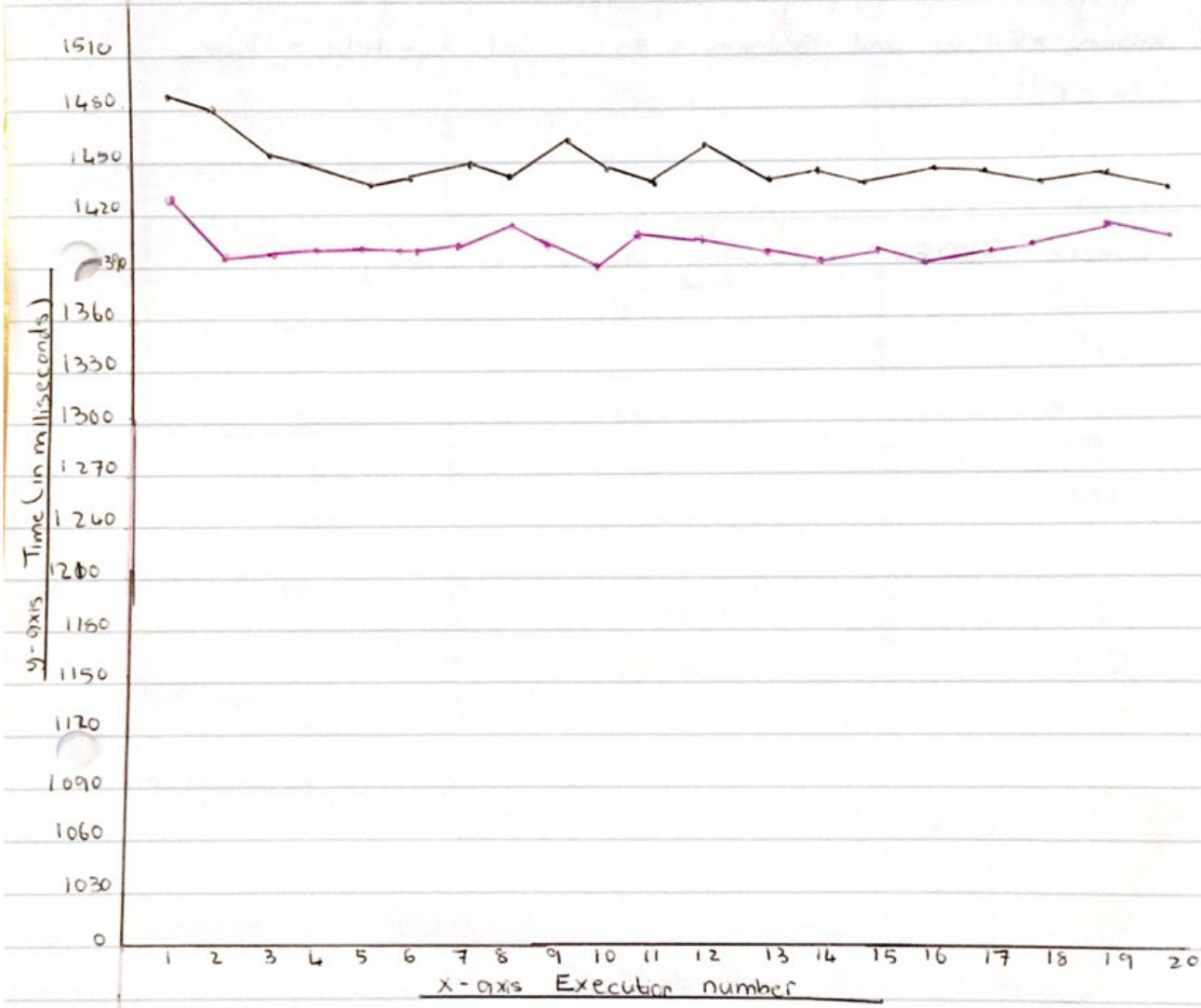
To add an index the query was:

```
Create INDEX ind_salary ON Salaries (salary);
```

Table showing results of running the query with and without the index on Salaries

| Number of Execution | Execution time (without index) in milliseconds | Execution time (with index) in milliseconds |
|---|---|---|
| 1 | 1482 | 1426 |
| 2 | 1480 | 1395 |
| 3 | 1453 | 1397 |
| 4 | 1449 | 1398 |
| 5 | 1441 | 1395 |
| 6 | 1446 | 1397 |
| 7 | 1450 | 1396 |
| 8 | 1442 | 1402 |
| 9 | 1462 | 1393 |
| 10 | 1443 | 1390 |
| 12 | 1458 | 1406 |
| 13 | 1442 | 1399 |
| 14 | 1445 | 1396 |
| 15 | 1443 | 1394 |
| 16 | 1447 | 1396 |
| 17 | 1446 | 1394 |
| 18 | 1434 | 1392 |
| 19 | 1440 | 1395 |
| 20 | 1436 | 1394 |

A line graph Showing the execution times (in milliseconds) for the given query (when indexed and not indexed) for 20 Successive runs whilst the query remains constant




| Key | |
|---|---|
| — | Without index on Salaries |
| — | With index on Salaries |

Paragraph

When running the query without the index, most (if not all) execution times were above 1440 milliseconds. When running the query with an index, most execution times were around 1395. When running the query with an index on Salary, the execution time was quicker for every single number of execution when compared to running without an

index. The highet time without an index is 1482 milliseconds.
The shortest time without an index is 1436 milliseconds
The highest and shortest times with an index were 1426 and
1390 milliseconds respectively.

1482 - 1390 = 92 milliseconds. This is the time difference from the highest to lowest execution times. From the results, we can conclude that the execution time is quicker when there is an index on salaries compared to no index on salaries. We can also conclude that the execution time with an index stays more constant between executions as opposed to without an index on salaries.

## Task 4

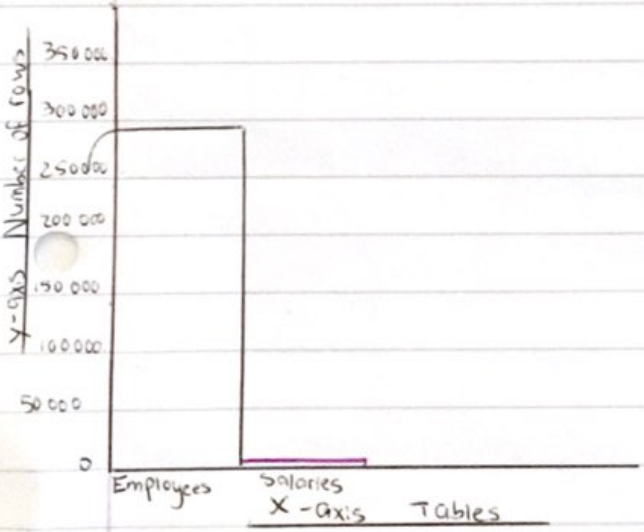
The query was:

Explain Select employees.first-name, Salaries.Salary from employees INNER JOIN Salaries on employees.emp-no = Salaries.emp-no ORDER BY Salary ASC LIMIT 5;

~~The Salaries table was queried~~

| | Employees | Salaries |
|---|---|---|
| Rows | 299778 | 4 |

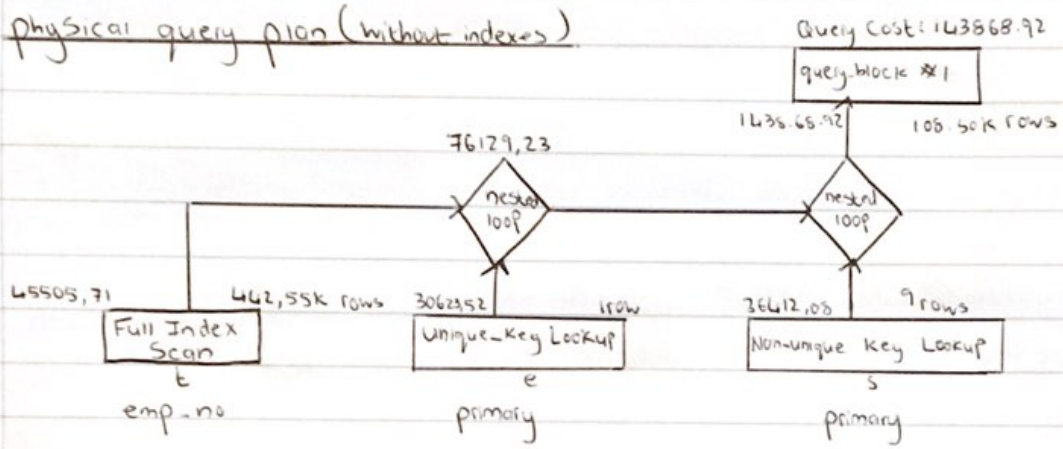
A histogram showing the number of rows queried for each table while the query remained constant



## Task 5

Query: Select e.first_name, e.last_name, t.title, s.salary from employees e, titles t, salaries s, where e.emp_no = t.emp_no and t.emp_no = s.emp_no and t.title = 'engineer' and s.salary > 80 000 and s.salary <100 000.

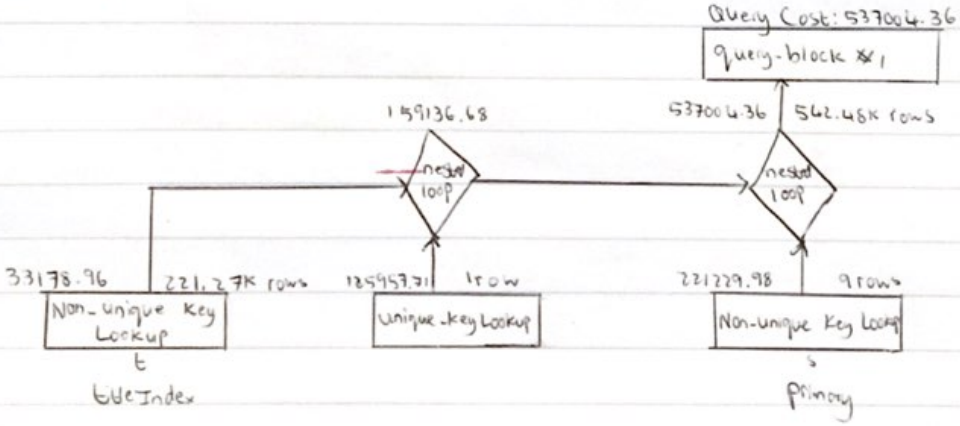
### physical query plan (without indexes)




Two indexes you can add could be on the titles and on employees

- Create Index employeeIndx on employees(emp_no);
- Create Index titleIndx on titles(title);

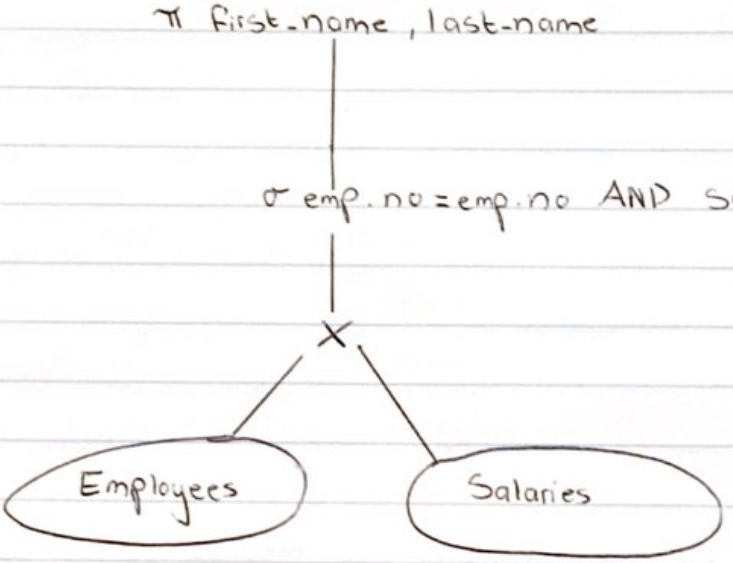
### physical query plan (with indexes)



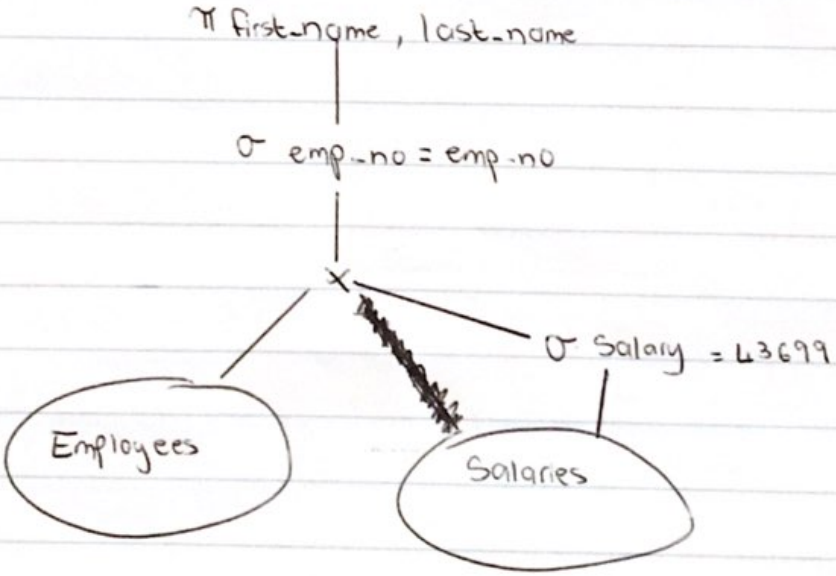
Task 6

Select distinct(first_name), last_name from employees, Salaries where employees.emp_no = Salaries.emp_no and salary > 43699;

Query tree




Optimised query tree



Relational algebra Statement

$\pi$ first_name, last_name ( employees $\bowtie$ salaries.emp_no = employees.emp_no

( $\sigma$ salary = 43699 (salaries) )

Paragraph

Yes, I expected the optimised query tree execution times to be slightly better than the initial query tree execution times. I expected the time for optimisation queries to be much faster though. But nevertheless, faster times were achieved in the end.