Abishek S
EE18B001

CS3700:
Introduction to Database Systems
Assignment 4b

October 29, 2021

Index Effect Study

Query Description

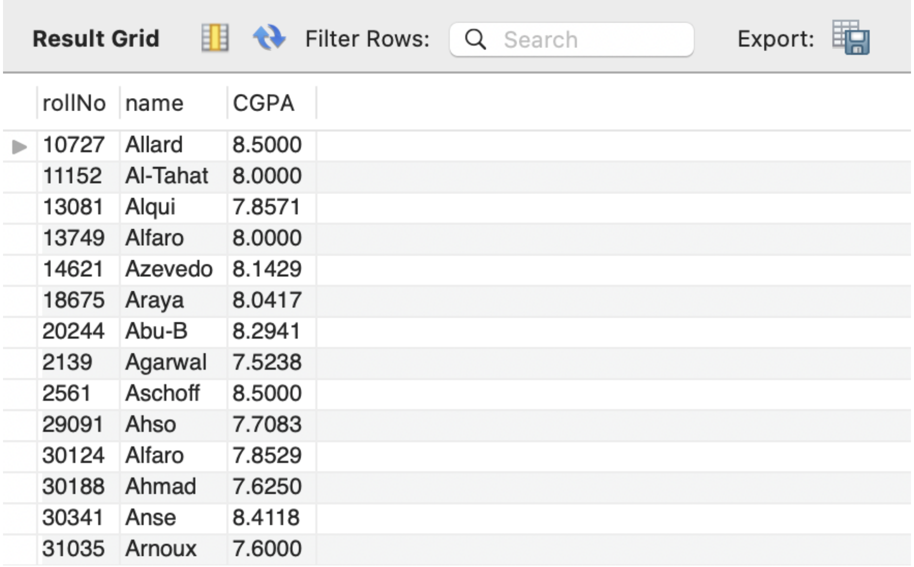
Print the CGPA with name and rollNo of all male students whose names start with ‘A’, have passed at least one course and secured atleast 7.5 CGPA. (S grade: 10, A grade: 9, B grade: 8, C grade: 7, D grade: 6, E grade: 5, U grade: Fail)

SQL Query

Query

```
SELECT
  s.rollNo,
  s.name,
  SUM(
    (CASE
          WHEN grade = 'S' THEN 10
          WHEN grade = 'A' THEN 9
          WHEN grade = 'B' THEN 8
          WHEN grade = 'C' THEN 7
          WHEN grade = 'D' THEN 6
          WHEN grade = 'E' THEN 5
    END)
    * (credits)
  ) / SUM(credits) AS CGPA
FROM enrollment AS e, course AS c, student AS s
WHERE e.courseId = c.courseId AND s.rollNo =  e.rollNo
  AND grade != 'U' AND grade IS NOT NULL AND s.name LIKE 'A%' AND s.sex = 'male'
GROUP BY rollNo
HAVING  CGPA >= 7.5
ORDER BY rollNo;
```

Output of Query

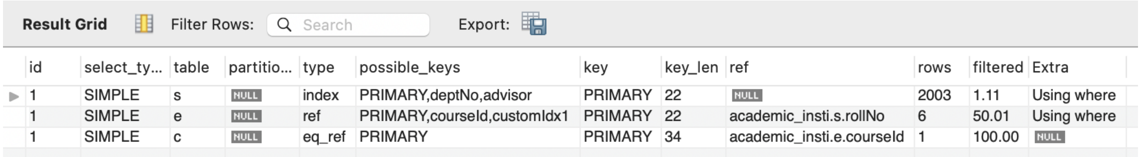


rollNo	name	CGPA
10727	Allard	8.5000
11152	Al-Tahat	8.0000
13081	Alqui	7.8571
13749	Alfaro	8.0000
14621	Azevedo	8.1429
18675	Araya	8.0417
20244	Abu-B	8.2941
2139	Agarwal	7.5238
2561	Aschoff	8.5000
29091	Ahso	7.7083
30124	Alfaro	7.8529
30188	Ahmad	7.6250
30341	Anse	8.4118
31035	Arnoux	7.6000

Figure 1: Query results

`EXPLAIN` before Custom Index

`EXPLAIN` command was used to obtain the query plan generated by MySQL engine for the query above (without the addition of any new index), and the results are as follows:



id	select_ty...	table	partitio...	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	s	NULL	index	PRIMARY,deptNo,advisor	PRIMARY	22	NULL	2003	1.11	Using where
1	SIMPLE	e	NULL	ref	PRIMARY,courseId,customIdx1	PRIMARY	22	academic_insti.s.rollNo	6	50.01	Using where
1	SIMPLE	c	NULL	eq_ref	PRIMARY	PRIMARY	34	academic_insti.e.courseId	1	100.00	NULL

Figure 2: `EXPLAIN` results before creating custom Index

Note that MySQL engine accesses the relations in the order:

1. student
2. enrollment
3. course

It goes through about 2003 × 6 × 1 = **12018** rows to get the result for the query. This is the case because the query has to go through all the rows in the *student* relation and use where to filter it (as seen from Extra column). But in reality the filtered percentage of rows is minimal (1.11%). This hints at the possibility of using index to speed up the query. The number of rows accessed in the subsequent relations are minimal and filtering percentage is also high, so I focus on creating index for the *student* relation.

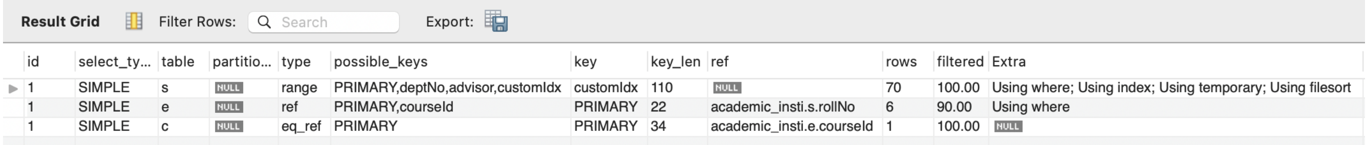
Output of `EXPLAIN` after Custom Index

The query involves mainly two filtering on the *student* table: `s.name LIKE ‘A%’ AND s.sex = ‘male’`. So I create the following index using the command:

Query

```
CREATE INDEX customIdx ON student(sex ASC, name ASC);
```

The intuition behind this is to identify and filter male students with names starting with ‘A’ from the index itself and then the necessary rows alone be traversed.
On using the `EXPLAIN` command again the following query plan was obtained:



id	select_ty...	table	partitio...	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	s	NULL	range	PRIMARY,deptNo,advisor,customIdx	customIdx	110	NULL	70	100.00	Using where; Using index; Using temporary; Using filesort
1	SIMPLE	e	NULL	ref	PRIMARY,courseId	PRIMARY	22	academic_insti.s.rollNo	6	90.00	Using where
1	SIMPLE	c	NULL	eq_ref	PRIMARY	PRIMARY	34	academic_insti.e.courseId	1	100.00	NULL

Figure 3: `EXPLAIN` results after creating custom index

The query plan matches the intuition and only traverses the relevant rows (notice the 100% filtered). This leads to a significant reduction in the number of rows required to provide the output of the query. Now it only requires $70 \times 6 \times 1 =$ **420** rows while it was **12018** earlier. This demonstrates the optimization due to indexing.