CS3700 Introduction to Database Systems

Assignment 4B: Index Effect Study

CS18B050 Aniswar Srivatsa Krishnan

Description:

Obtain the names and roll numbers of the students from the CSE 2002 batch who have scored the first, second and third highest number of S grades, along with the number of S grades they have scored.

Query:

```
select s.rollNo, s.name, number_of_s
        from (
        select rollNo as r1, count(*) as number_of_s
        from enrollment
        where grade = 'S'
        group by rollNo) as T, student s, department d
        where T.r1 = s.rollNo and s.deptNo = d.deptId and d.name = 'Comp. Sci.' and s.year =
'2002' order by number_of_s desc limit 3;
```

Query plan before creating any index:

Table:

```
+----+-------------+------------+------------+-------+-----------------+-------------+---------+-------------------------+-------+----------+-----------------------------
--------------+
| id | select_type | table      | partitions | type  | possible_keys   | key         | key_len | ref                     | rows  | filtered | Extra
              |
+----+-------------+------------+------------+-------+-----------------+-------------+---------+-------------------------+-------+----------+-----------------------------
--------------+
|  1 | PRIMARY     | d          | NULL       | ALL   | PRIMARY         | NULL        | NULL    | NULL                    |    20 |    10.00 | Using where; Using temporary;
Using filesort |
|  1 | PRIMARY     | s          | NULL       | ref   | PRIMARY,deptNo  | deptNo      | 83      | academic_insti.d.deptId |   100 |    10.00 | Using where
              |
|  1 | PRIMARY     | <derived2> | NULL       | ref   | <auto_key0>     | <auto_key0> | 22      | academic_insti.s.rollNo |    10 |   100.00 | NULL
              |
|  2 | DERIVED     | enrollment | NULL       | index | PRIMARY,courseId | PRIMARY     | 84      | NULL                    | 13586 |    10.00 | Using where
              |
+----+-------------+------------+------------+-------+-----------------+-------------+---------+-------------------------+-------+----------+-----------------------------
--------------+
4 rows in set, 1 warning (0.00 sec)
```

Tree:

```
| -> Limit: 3 row(s)
    -> Sort: T.number_of_s DESC, limit input to 3 row(s) per chunk
        -> Stream results  (cost=2853.93 rows=27213)
            -> Nested loop inner join  (cost=2853.93 rows=27213)
                -> Nested loop inner join  (cost=82.28 rows=20)
                    -> Filter: (d.`name` = 'Comp. Sci.')  (cost=2.25 rows=2)
                        -> Table scan on d  (cost=2.25 rows=20)
                    -> Filter: (s.`year` = 2002)  (cost=30.50 rows=10)
                        -> Index lookup on s using deptNo (deptNo=d.deptId)  (cost=30.50 rows=100)
                -> Index lookup on T using <auto_key0> (r1=s.rollNo)
                    -> Materialize  (cost=1654.57..1654.57 rows=1359)
                        -> Group aggregate: count(0)  (cost=1518.71 rows=1359)
                            -> Filter: (enrollment.grade = 'S')  (cost=1382.85 rows=1359)
                                -> Index scan on enrollment using PRIMARY  (cost=1382.85 rows=13586)
 |
```

Number of row accesses for courseId in enrollment table = 13586

We notice that in our query, we just want the people who have S grades. If we index based on any other column then we need to iterate over the whole table to check whether the grade is S or not. Therefore we create an index for the grade column and now we can easily take the people who have got S grades.

Query plan after creating the index for enrollment(grade):

Table:

```
mysql> create index grade_index on enrollment(grade);
Query OK, 0 rows affected (0.07 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> explain select s.rollNo, s.name, number_of_s     from (         select rollNo as r1, count(*) as number_of_s             from enrollment             where grade = 'S'
          group by rollNo) as T, student s, department d     where T.r1 = s.rollNo and s.deptNo = d.deptId and d.name = 'Comp. Sci.' and s.year = '2002' order by number_o
f_s desc limit 3;
+----+-------------+------------+------------+------+-----------------------------+-------------+---------+-------------------------+------+----------+----------------------
------------------------+
| id | select_type | table      | partitions | type | possible_keys               | key         | key_len | ref                     | rows | filtered | Extra
                       |
+----+-------------+------------+------------+------+-----------------------------+-------------+---------+-------------------------+------+----------+----------------------
------------------------+
|  1 | PRIMARY     | d          | NULL       | ALL  | PRIMARY                     | NULL        | NULL    | NULL                    |   20 |    10.00 | Using where; Using t
emporary; Using filesort |
|  1 | PRIMARY     | s          | NULL       | ref  | PRIMARY,deptNo              | deptNo      | 83      | academic_insti.d.deptId |  100 |    10.00 | Using where
                       |
|  1 | PRIMARY     | <derived2> | NULL       | ref  | <auto_key0>                 | <auto_key0> | 22      | academic_insti.s.rollNo |   10 |   100.00 | NULL
                       |
|  2 | DERIVED     | enrollment | NULL       | ref  | PRIMARY,courseId,grade_index | grade_index | 11      | const                   | 1492 |   100.00 | Using index
                       |
+----+-------------+------------+------------+------+-----------------------------+-------------+---------+-------------------------+------+----------+----------------------
------------------------+
4 rows in set, 1 warning (0.00 sec)
```

Tree:

```
| -> Limit: 3 row(s)
    -> Sort: T.number_of_s DESC, limit input to 3 row(s) per chunk
        -> Stream results  (cost=3120.90 rows=29885)
            -> Nested loop inner join  (cost=3120.90 rows=29885)
                -> Nested loop inner join  (cost=82.28 rows=20)
                    -> Filter: (d.`name` = 'Comp. Sci.')  (cost=2.25 rows=2)
                        -> Table scan on d  (cost=2.25 rows=20)
                    -> Filter: (s.`year` = 2002)  (cost=30.50 rows=10)
                        -> Index lookup on s using deptNo (deptNo=d.deptId)  (cost=30.50 rows=100)
                -> Index lookup on T using <auto_key0> (r1=s.rollNo)
                    -> Materialize  (cost=465.09..465.09 rows=1492)
                        -> Group aggregate: count(0)  (cost=315.89 rows=1492)
                            -> Covering index lookup on enrollment using grade_index (grade='S')  (cost=166.69 rows=1492)
 |
```

Number of row accesses for courseId in enrollment table = 1492
Number of row accesses for department table = 20
We have obtained almost a **10-fold** improvement in the number of row accesses for courseId in the enrollment table. We now notice that there are 20 row accesses for the department table. We observe that the query filters rows that have the department name as ‘Comp Sci’. Since department name is not a key we have to iterate over the entire table and check for the department name. Therefore we create an index for the department name and now we can check the department name.

Query plan after creating the index for department(name):

Table:

```
mysql> explain select s.rollNo, s.name, number_of_s     from (         select rollNo as r1, count(*) as number_of_s             from enrollment             where grade = 'S'
          group by rollNo) as T, student s, department d     where T.r1 = s.rollNo and s.deptNo = d.deptId and d.name = 'Comp. Sci.' and s.year = '2002' order by number_o
f_s desc limit 3;
+----+-------------+------------+------------+------+-----------------------------+---------------+---------+-------------------------+------+----------+-----------------
---------------------------+
| id | select_type | table      | partitions | type | possible_keys               | key           | key_len | ref                     | rows | filtered | Extra
                           |
+----+-------------+------------+------------+------+-----------------------------+---------------+---------+-------------------------+------+----------+-----------------
---------------------------+
|  1 | PRIMARY     | d          | NULL       | ref  | PRIMARY,deptname_index      | deptname_index | 83      | const                   |    1 |   100.00 | Using index; Usin
g temporary; Using filesort |
|  1 | PRIMARY     | s          | NULL       | ref  | PRIMARY,deptNo              | deptNo         | 83      | academic_insti.d.deptId |  100 |    10.00 | Using where
                           |
|  1 | PRIMARY     | <derived2> | NULL       | ref  | <auto_key0>                 | <auto_key0>    | 22      | academic_insti.s.rollNo |   10 |   100.00 | NULL
                           |
|  2 | DERIVED     | enrollment | NULL       | ref  | PRIMARY,courseId,grade_index | grade_index    | 11      | const                   | 1492 |   100.00 | Using index
                           |
+----+-------------+------------+------------+------+-----------------------------+---------------+---------+-------------------------+------+----------+-----------------
---------------------------+
4 rows in set, 1 warning (0.01 sec)
```

Tree:

```
| -> Limit: 3 row(s)
    -> Sort: T.number_of_s DESC, limit input to 3 row(s) per chunk
        -> Stream results  (cost=1559.67 rows=14942)
            -> Nested loop inner join  (cost=1559.67 rows=14942)
                -> Nested loop inner join  (cost=40.37 rows=10)
                    -> Covering index lookup on d using deptname_index (name='Comp. Sci.')  (cost=0.35 rows=1)
                    -> Filter: (s.`year` = 2002)  (cost=31.00 rows=10)
                        -> Index lookup on s using deptNo (deptNo=d.deptId)  (cost=31.00 rows=100)
                -> Index lookup on T using <auto_key0> (r1=s.rollNo)
                    -> Materialize  (cost=465.09..465.09 rows=1492)
                        -> Group aggregate: count(0)  (cost=315.89 rows=1492)
                            -> Covering index lookup on enrollment using grade_index (grade='S')  (cost=166.69 rows=1492)
 |
```

Number of row accesses for courseId in enrollment table = 1492
Number of row accesses for department table = 1
We have decreased the number of row accesses for the department table from **20 to 1**

We have thus observed that the performance of queries can be increased by creating various indices appropriately. We need to analyze the queries that are being used frequently and create indices accordingly.