

Kevin Hance
DBMS (CPSC 321)
HW10

READING ASSIGNMENT:

1.
 - a. It would not make much sense to create a dense index on salary for this table, as multiple records will likely be mapped to the same salary. Because of this, having a one-to-one mapping relationship between the records and the salary values wouldn't be useful. In addition, getting the salary of an employee likely doesn't need to be lightning-fast, so there is no major issue with using a sparse index in this case.
 - b. It would make sense to create a clustered index here, as having a sorted list of salaries could be useful in certain situations. For example, if you wanted to query the database to return all the employees making \$50,000 or less per year, the program would stop looking for matches after it passed the 50,000 mark in the database. This would speed up the database and result in queries running faster.
 - c. It would make sense to create a sparse index on this table, as multiple employees likely have the same salary, and this would make the database simpler in terms of storing that data.
2. and 3.

Three version of same query in Relational Algebra

1. $\pi_{\text{instructor.name, course.course_id, course.title}}$
 $\sigma_{\text{instructor.dept} = \text{'Music'}}$
 $(\rho_{\text{course}}(\text{instructor_id}) \times \rho_{\text{teaches}}(\text{instructor_id}))$
 $\times \rho_{\text{instructor}}(\text{instructor_id})$
2. $\sigma_{\text{instructor.dept} = \text{'Music'}}$
 $\pi_{\text{instructor.name, course.course_id, course.title}}$
 $(\rho_{\text{instructor}}(\text{instructor_id}) \times \rho_{\text{course}}(\text{instructor_id}))$
 $\times \rho_{\text{course}}(\text{instructor_id})$
3. $\sigma_{\text{instructor.dept} = \text{music} \wedge \text{instructor.instructor_id} < \text{course.instructor_id}}$
 $\wedge \text{teaches.instructor_id} = \text{course.instructor_id}$
 $(\text{instructor} \times \text{teaches} \times \text{course})$

4.

Materialization makes a bit more sense to write but has a higher computing cost because the cost of writing the results of the temporary relation queries must be taken into account. Materialization starts at the lowest level (selection of table) and works up from there (dealing with the individual relations in the table). Pipelining involves the reduction of the number of temporary files by combining several relational operations into a pipeline of operations. This eliminates the cost of reading and writing temp files, thus reducing the cost of the query.

TECHNICAL WORK:

Step 1:

```
1  DROP TABLE IF EXISTS Employee;
2
3  CREATE TABLE Employee(
4      employee_id INTEGER UNSIGNED,
5      salary INTEGER UNSIGNED,
6      title VARCHAR(50),
7      PRIMARY KEY (employee_id)
8  );
9
10 /*CREATE INDEX salary ON Employee(title, salary);*/
11 /*SELECT * FROM Employee WHERE title = 'engineer' and salary < 50000;*/
```

Step 2:

```

hw10.py > ...
1  import random
2  def main():
3      writeQueryToFile(10000)
4      writeQueryToFile(100000)
5      writeQueryToFile(1000000)
6
7  def writeQueryToFile(a):
8      f = open("hw10-data-" + str(a) + ".sql", "w")
9      title = ["engineer", "salesperson", "administrator", "manager"]
10     arrayCounter = 0
11     i = 0
12     f.write("INSERT INTO Employee VALUES")
13     while (i < a):
14         randSalary = random.randint(12000, 150000)
15         f.write("(" + str(i) + "," + str(randSalary) + "," + "'" + title[arrayCounter] + "'" + ")")
16         if(i != (a - 1)):
17             f.write(", " + "\n")
18             arrayCounter += 1
19             arrayCounter = arrayCounter % 4
20             i += 1
21     f.write(";")
22     f.close()
23
24     main()
25
26

```

Step 3 with no index:

Rows	Update without index (seconds)	Query without index (seconds)
10,000	0.32	0.01
100,000	1.71	0.03
100,000	16.39	0.39

10,000 employees

```
mysql> source hw10-data-10000.sql
Query OK, 10000 rows affected (0.32 sec)
Records: 10000 Duplicates: 0 Warnings: 0
```

9491	30101	manager
9536	30586	engineer
9688	29976	engineer
9714	29828	administrator
9732	29131	engineer
9905	30573	salesperson
9930	29884	administrator
9978	29207	administrator

-----+
139 rows in set (0.01 sec)

100,000 employees

```
mysql> source hw10-data-100000.sql
Query OK, 100000 rows affected (1.71 sec)
Records: 100000 Duplicates: 0 Warnings: 0
```

99474	29110	administrator
99542	30667	administrator
99554	31000	administrator
99614	29561	administrator
99739	29592	manager
99768	30212	engineer
99911	29893	manager
99973	30836	salesperson
99975	30943	manager

-----+
1454 rows in set (0.03 sec)

1,000,000 employees

```
mysql> source hw10-data-1000000.sql
Query OK, 1000000 rows affected (16.39 sec)
Records: 1000000 Duplicates: 0 Warnings: 0
```

999527	30539	manager
999558	30756	administrator
999564	30871	engineer
999607	29980	manager
999645	30933	salesperson
999655	30647	manager
999741	30933	salesperson
999776	29295	engineer
999860	30328	engineer

-----+
14513 rows in set (0.39 sec)

Step 4: Do step 3 again with index on salary

Rows	Update with index (seconds)	Query with index (seconds)
10,000	0.25	0.00
100,000	1.00	0.00
100,000	21.51	0.08

Created index, ran schema on 10,000 entry file

```
mysql> source hw10-data-10000.sql
Query OK, 10000 rows affected (0.25 sec)
Records: 10000 Duplicates: 0 Warnings: 0
```

```

      8443 | 30882 | manager
      1920 | 30888 | engineer
      2383 | 30902 | manager
      8217 | 30916 | salesperson
      1738 | 30922 | administrator
      2491 | 30949 | manager
      9266 | 30961 | administrator
      4282 | 30965 | administrator
      8871 | 30970 | manager
      6248 | 30978 | engineer
      1690 | 30983 | administrator
      8073 | 30998 | salesperson
+-----+-----+
139 rows in set (0.00 sec)
```

Created index, ran schema on 100,000 entry file

```
mysql> source hw10-data-100000.sql
Query OK, 100000 rows affected (1.00 sec)
Records: 100000 Duplicates: 0 Warnings: 0
```

```

      94043 | 30994 | salesperson
      23398 | 30995 | administrator
      50993 | 30995 | salesperson
      79884 | 30995 | engineer
      11860 | 30996 | engineer
      75881 | 30996 | salesperson
      95163 | 30996 | manager
      32143 | 30998 | manager
      82293 | 31000 | salesperson
      99554 | 31000 | administrator
+-----+-----+
1454 rows in set (0.00 sec)
```

Created index, ran schema on 1,000,000 entry file

```
mysql> source hw10-data-1000000.sql
Query OK, 1000000 rows affected (21.51 sec)
Records: 1000000 Duplicates: 0 Warnings: 0
```

```

263964 | 31000 | engineer
309854 | 31000 | administrator
420859 | 31000 | manager
558948 | 31000 | engineer
571507 | 31000 | manager
667692 | 31000 | engineer
728159 | 31000 | manager
942799 | 31000 | manager
951730 | 31000 | administrator
966807 | 31000 | manager
+-----+-----+
14513 rows in set (0.08 sec)
```

Step 5: Do step 4 again with index on title and salary

Rows	Update without and with indexes (seconds)	Query without and with indexes (seconds)
10,000	0.12, 0.22	0.04, 0.00
100,000	0.65, 1.24	0.04, 0.00
100,000	7.51, 15.06	0.32, 0.09

10,000

Removed index, ran schema file again

```
mysql> source hw10-data-10000.sql
Query OK, 10000 rows affected (0.12 sec)
Records: 10000 Duplicates: 0 Warnings: 0
```

Running query `SELECT * FROM Employee WHERE title = 'engineer' and salary < 50000;`

```

  9908 | 33319 | engineer |
  9916 | 37131 | engineer |
  9920 | 23498 | engineer |
  9944 | 24943 | engineer |
  9960 | 41040 | engineer |
  9964 | 47655 | engineer |
  9996 | 49562 | engineer |
+-----+-----+-----+
691 rows in set (0.04 sec)
```

Running schema again with index on title and salary

```
mysql> source hw10-data-10000.sql
Query OK, 10000 rows affected (0.22 sec)
Records: 10000 Duplicates: 0 Warnings: 0
```

Running query `SELECT * FROM Employee WHERE title = 'engineer' and salary < 50000;`

```

  8344 | 49700 | engineer |
  4932 | 49746 | engineer |
  7628 | 49748 | engineer |
  5040 | 49758 | engineer |
  5044 | 49785 | engineer |
  1648 | 49817 | engineer |
  3652 | 49919 | engineer |
  4088 | 49930 | engineer |
+-----+-----+-----+
691 rows in set (0.00 sec)
```

100,000

```
mysql> source hw10-data-100000.sql
Query OK, 100000 rows affected (0.65 sec)
Records: 100000 Duplicates: 0 Warnings: 0
```

```

  99900 | 48509 | engineer |
  99908 | 13524 | engineer |
  99916 | 39571 | engineer |
  99924 | 35678 | engineer |
  99940 | 27806 | engineer |
  99948 | 37567 | engineer |
  99980 | 13805 | engineer |
  99992 | 37762 | engineer |
+-----+-----+-----+
6876 rows in set (0.04 sec)
```

Running again with index on title and salary

```
mysql> source hw10-data-100000.sql
Query OK, 100000 rows affected (1.24 sec)
Records: 100000 Duplicates: 0 Warnings: 0
```

41898	49942	engineer
52980	49942	engineer
68180	49946	engineer
34500	49959	engineer
30600	49960	engineer
31104	49963	engineer
29260	49965	engineer
41384	49980	engineer
16372	49988	engineer
47032	49994	engineer
74232	49996	engineer

+-----+-----+-----+

6876 rows in set (0.00 sec)

1,000,000

```
mysql> source hw10-data-1000000.sql
Query OK, 1000000 rows affected (7.51 sec)
Records: 1000000 Duplicates: 0 Warnings: 0
```

999864	44398	engineer
999880	15972	engineer
999896	44414	engineer
999944	37775	engineer
999964	36037	engineer
999976	26082	engineer
999980	26904	engineer
999988	48852	engineer
999996	13015	engineer

+-----+-----+-----+

68822 rows in set (0.32 sec)

Running again with index on title and salary

```
mysql> source hw10-data-1000000.sql
Query OK, 1000000 rows affected (15.06 sec)
Records: 1000000 Duplicates: 0 Warnings: 0
```



```
518944 | 49991 | engineer |
202780 | 49992 | engineer |
892932 | 49992 | engineer |
500904 | 49993 | engineer |
712056 | 49993 | engineer |
930216 | 49993 | engineer |
588340 | 49994 | engineer |
592664 | 49994 | engineer |
706332 | 49994 | engineer |
215424 | 49995 | engineer |
608224 | 49995 | engineer |
682132 | 49996 | engineer |
304168 | 49997 | engineer |
569932 | 49997 | engineer |
552776 | 49998 | engineer |
600920 | 49998 | engineer |
4040 | 49999 | engineer |
+-----+-----+-----+
68822 rows in set (0.09 sec)
```

Discussion: It is now clear to be that indexing your schema can speed up query time while increasing the time it takes to update the table with values. This was a valuable exercise in Database Management, and I think I learned a lot about indexing during this exercise.