Design Document

For the lab portion of the homework assignment, the objective was to run some simple trials that will show us how using indexes can alter the cost of database updates and queries in terms of execution time. We were to create a multiple difference schema SQL files that satisfied the requirements for each problem, but there were only small differences and changes that needed to be made between them. We also had to make a Python program that will generate test data to insert into the Employee table. We were supposed to create data for the following sample sizes: 10,000 ; 100,000 ; and 1,000,000. After we have done that, we then had to compare the execution times after adding one and two indexes on the original table. I chose Python as my preferred programming language to generate the data because it is so much more simpler.

What I discovered was that tables that are non-indexed update much faster and quicker. However, when you have to do a large query, then indexed tables are quicker at finding results. So basically what you have is if you need more updates rather than queries, non-indexed queries will be better. If you perform large and more queries than updates, then indexed tables might be better (as we saw – it is useful for optimizing query results).

I did not experience that much difficulty with the lab. The only difficulty part I had was that it took forever for my SQL file with 1,000,000 values to be generated. I would leave it run for so long and then my computer’s fan would kick in and then I would cancel and try again. It was just a little tedious to source the correct scheme, insert the correct data and run the query so many times.

For testing, I simply ran the lab. I generated the SQL files that had 10000,100000, and 1000000 values. I then ran a query which searched the attribute table, and I basically recorded my answers (time) and compared. There is not that major of a difference of using indexed or the other as the size of the index may have not that much to do with the overall time. Below you can see many screenshots which capture my results.

David Hanany

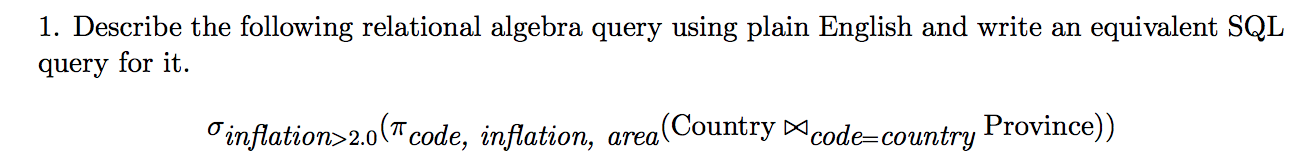
12/07/17

CPSC 321

Dr. Bowers

Assignment #11

Part One



This will return all the province areas that are associate with a country whose inflation is greater than 2.0 and the country’s code and inflation rate.

SELECT province\_name, area, code

FROM Province JOIN Country ON(country\_code=code)

WHERE Country.inflation > 2.0;

2. Rewrite the previous relational algebra query to an equivalent query but where the select happens before the project operator.

П code, area, inflation(σ inflation >2.0(Country l><l code=country Province))

3. Using the schema from HW 4, write a single relational algebra query to find the Cities with the smallest populations.

П city\_name (City) - П City.city\_name(σ City.population > c2.population (City X p c2 (City)))

4. Do the relational algebra queries in Exercise 6.1.

a. П title(σ dept\_name= “Comp. Sci.” ^ credits = 3(Course))

b. П takes.ID (σ name=Einstein(takes l><l course\_id=course\_id, sec\_id=sec\_id, year=year teaches))

c. П salary (instructor) - П instructor.salary(σ instructor.salary < i2.salary(instructor X p i2 (instructor)))

e. course\_id, sec\_id(Gcount \* (σ semester = “Autumn” ^ year = 2009)(takes))

5. Assume the DBMS created a clustered index for table Emp(id, dept, salary, age) on attribute id.

a. Would it make sense to create a dense index on salary for this table? Why or why not?

No, it does not make sense unless the index stores a list of pointers to all of the records that have the same search key value. You can have two instructors that can have identical salaries, but this will violate the constraints of one index entry for each data record.

b. Would it make sense to create a clustered index on salary for this table? Why or why not?

Yes, it would make sense because a clustered index can be ordered on any search key, regardless if it matches the primary key.

c. Would it make sense to create a sparse index on salary for this table? Why or why not?

No, it would not because the search key (salary) isn't the primary key. The index entries may overlap if the search key is not sorted on salary. This makes searching for values based on index entry not possible.

Part 2

Step 1

/\*

\* Name: David Hanany

\* Date: 12/07/17

\* CPSC 321

\* HW #11

\* Description: Part 1 of the homework assignment

\*/

DROP TABLE IF EXISTS Employee;

CREATE TABLE Employee

(

employee\_id INT NOT NULL,

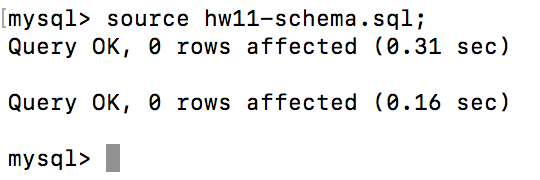
salary INT,

title VARCHAR (255),

PRIMARY KEY (employee\_id)

)

ENGINE = InnoDB;



Step 2

#Name: David Hanany

#Date: 12/07/17

#CPSC 321

#HW 11

#Description: Part 2 of the homework assignment

#Generates test data to insert into the Employee table.

import random

#creates insert statement

#n is the number of instances

def create\_insert\_stmt(n):

joblist=["engineer", "manager", "salesperson", "administrator"]

values=""

for x in range(0, n):

s=random.randint(12000, 150000)

j= joblist[x%4]

c=","

if x==n-1:

c= ";"

values=values+'('+str(x+1)+','+str(s)+','+ '"'+j+'"'+')'+c+'\n'

insert\_stmt= "INSERT INTO Employee VALUES "+'\n'+values

return insert\_stmt

# File I/O

def generate\_file(insert\_stmt):

file= open("data", "w")

file.write(insert\_stmt)

file.close()

print insert\_stmt

def main():

n=10000

insert\_stmt=create\_insert\_stmt(n)

generate\_file(insert\_stmt)

if \_\_name\_\_ == '\_\_main\_\_':

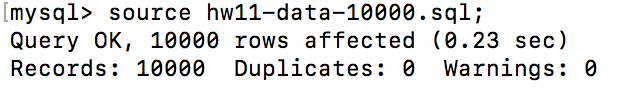
main()

Step 3

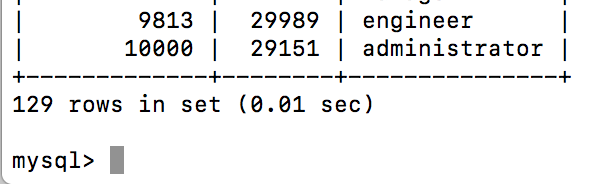
|  |  |  |
| --- | --- | --- |
| Rows | Update (seconds) | Query (seconds) |
| 10000 | 0.23 | 0.01 |
| 100000 | 2.00 | 0.13 |
| 1000000 | 21.36 | 1.07 |

10,000

Sourcing File:

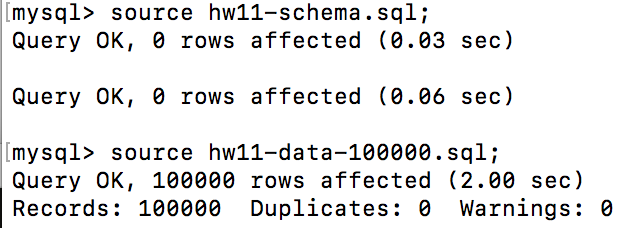


Running Query:

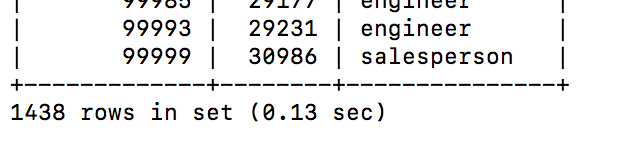


100,000

Sourcing File:

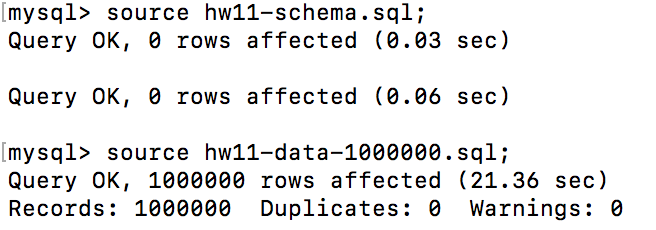


Running Query:

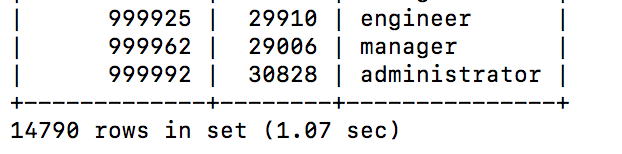


1,000,000

Sourcing File:



Running Query:



Step 4

/\*

\* Name: David Hanany

\* Date: 12/07/17

\* CPSC 321

\* HW #11

\* Description: Part 4 of the homework assignment

\*/

-- to enforce various constraints

SET sql\_mode = STRICT\_ALL\_TABLES;

DROP TABLE IF EXISTS Employee;

CREATE TABLE Employee (

employee\_id INT NOT NULL,

salary INT,

title VARCHAR(30) NOT NULL,

PRIMARY KEY (employee\_id)

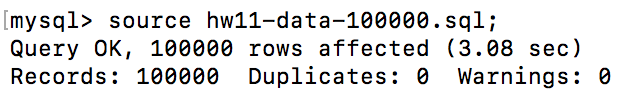
) ENGINE = InnoDB;

CREATE INDEX emp\_salary\_ind ON Employee (salary);

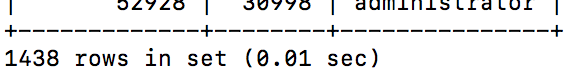
|  |  |  |
| --- | --- | --- |
| Rows | Update (seconds) | Query (seconds) |
| 10000 | 3.08 | 0.01 |
| 100000 | 3.38 | 0.02 |
| 1000000 | 44.15 | 0.17 |

10,000:

Sourcing File:

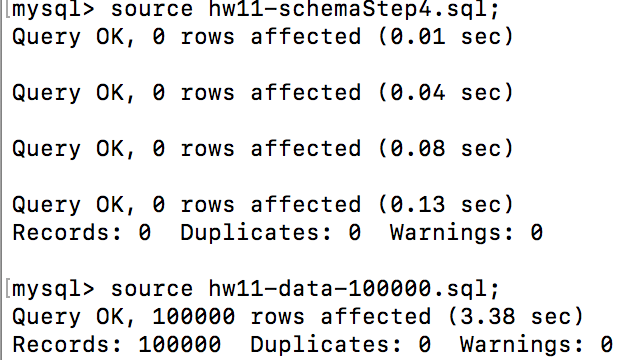


Running Query:

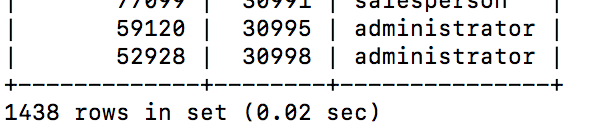


100,000

Sourcing File:

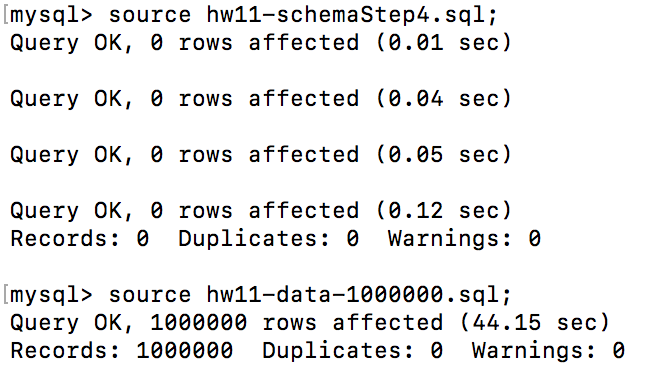


Running Query:

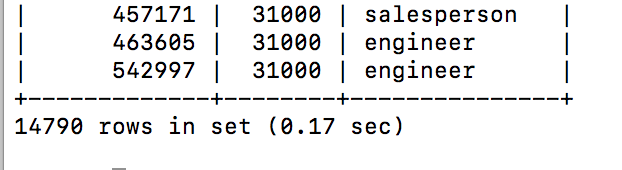


1,000,000

Sourcing File:



Running Query:



Step 5

Indexed

|  |  |  |
| --- | --- | --- |
| Rows | Update (seconds) | Query (seconds) |
| 10000 | 0.38 | 0.01 |
| 100000 | 3.53 | 0.01 |
| 1000000 | 105.93 | 0.12 |

/\*

\* Name: David Hanany

\* Date: 12/07/17

\* CPSC 321

\* HW #11

\* Description: Part 5 of the homework assignment

\*/

-- to enforce various constraints

SET sql\_mode = STRICT\_ALL\_TABLES;

DROP TABLE IF EXISTS Employee;

CREATE TABLE Employee (

employee\_id INT NOT NULL,

salary INT,

title VARCHAR(30) NOT NULL,

PRIMARY KEY (employee\_id)

) ENGINE = InnoDB;

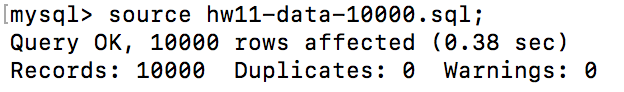
CREATE INDEX emp\_title\_salary\_ind ON Employee (title, salary);

Query:

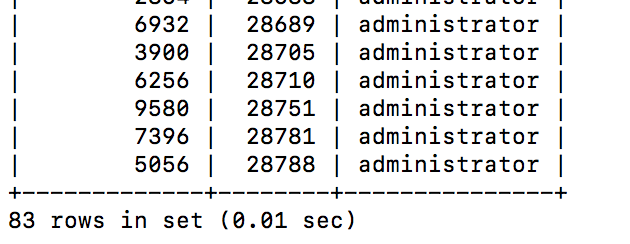
SELECT \* FROM Employee WHERE title = "administrator" AND salary >= 24000 AND salary <= 29000;

10,000

Sourcing File:

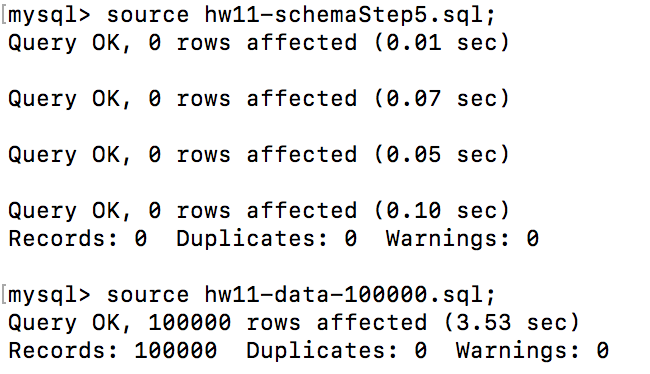


Running Query:

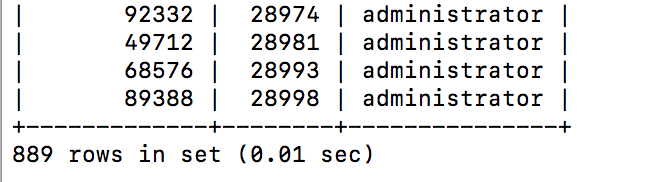


100,000

Sourcing File:

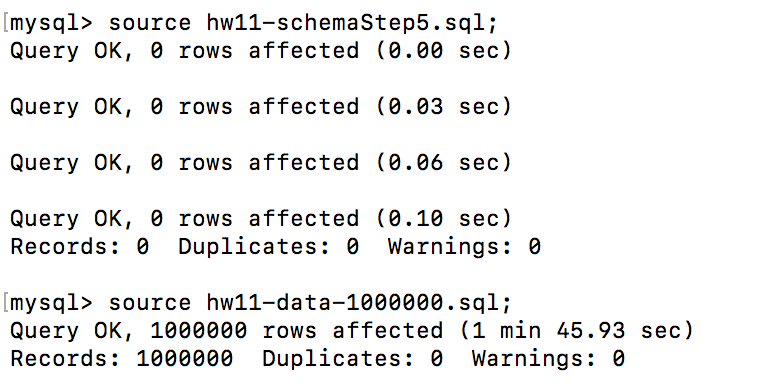


Running Query:

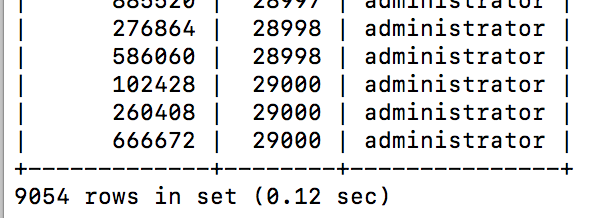


1,000,000

Sourcing File:



Running Query:



Without Indexed

|  |  |  |
| --- | --- | --- |
| Rows | Update (seconds) | Query (seconds) |
| 10000 | 0.43 | 0.01 |
| 100000 | 3.55 | 0.01 |
| 1000000 | 57.69 | 0.09 |

/\*

\* Name: David Hanany

\* Date: 12/07/17

\* CPSC 321

\* HW #11

\* Description: Part 5 of the homework assignment

\*/

-- to enforce various constraints

SET sql\_mode = STRICT\_ALL\_TABLES;

DROP TABLE IF EXISTS Employee;

CREATE TABLE Employee (

employee\_id INT NOT NULL,

salary INT,

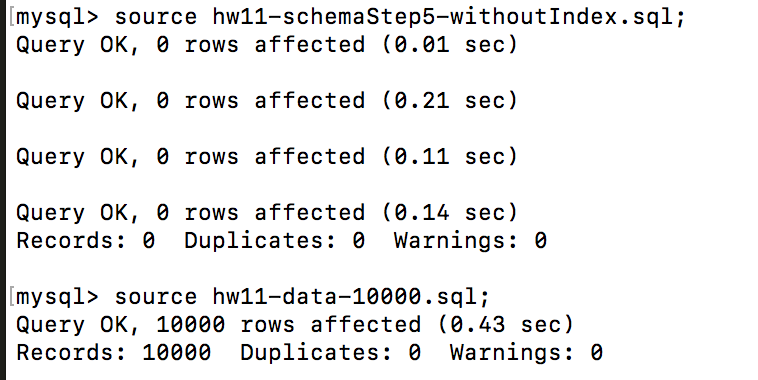
title VARCHAR(30) NOT NULL,

PRIMARY KEY (employee\_id)

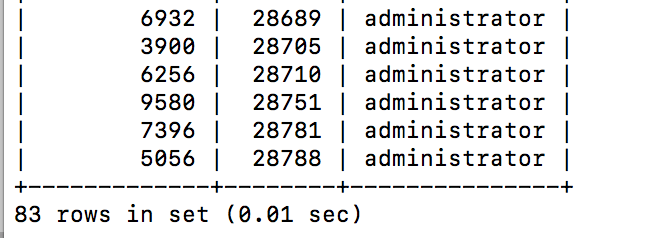
) ENGINE = InnoDB;

10,000

Sourcing Files:

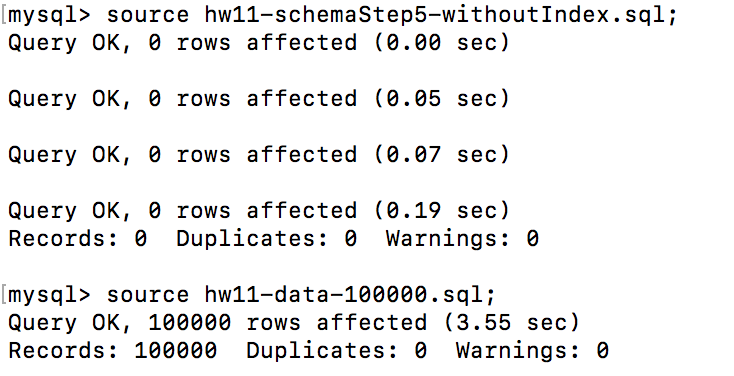


Running Query:

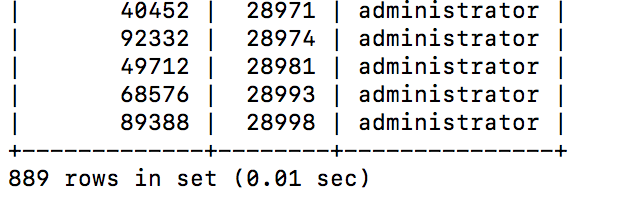


100,000

Sourcing Files:

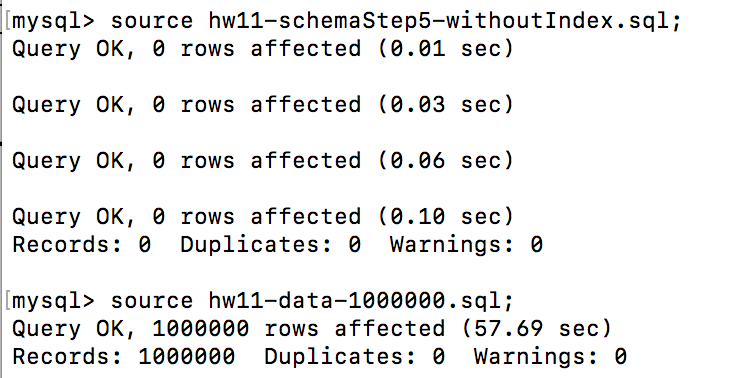


Running Query:



1,000,000

Sourcing Files:



Running Query:

