Database Systems WS 2019/20

Priom Biswas — Md Nabid Imteaj

11 November 2019

1 Question 1: Index tuning

Q1-Solution: This query requires join between Store, Article and Inventory table. Primary indexes that are created on primary keys are adequate for performing this query efficiently.

Q2-Solution: We can create a composite B+ tree on EID and Salary column of Employee table. Index on EID column will be used for point query and index on Salary column will be used for range query for performance gain.
Another solution can be to create indexes seperately where EID will be used as hash index for point query and Salary will be B+ tree index for range query.

Q3-Solution: A B+ tree index on AID and Price would be sufficient to improve performance of the query.

Q4-Solution: We can create a hash index for point query on Producer and B+ tree index on AID and Price column for performance improvements.

Q5-Solution: The columns involved (IID, SID), which are both primary keys, are already part of index. An additional non-clustered hash index on SID can help optimize the query. Hash index

Q6-Solution: A composite index on (Price, ProductionDate) which will perform range query will help performance gain.

2 Question 2: Index Costs

a) Here AID is primary index column, so a clustered index is created.

No. of pages required to store 400000 articles is 400000/5 = 80000 pages.

Initially to search 10000, we need to go beneath the tree, and as the height of tree is 'h', it requires h page accesses.

From 10000, we perform a sequential scan (since B+ tree) till 30000, so total access will be $1 + (30000 - 10000) = 20001$ no of tuples

Each page can hold 5 tuples, so no of pages required to access these tuples is $20001/5 = 4001$ pages
For leaf access, we require $4001/15 = 267$ pages

In total page access needed is $= 267 + 4001 + h$

b) Additionally, created Producer column is a non-clustered dense index. As it is a non-clustered index, in worst case, we might have to access entire data file, that is, starting from 10000 to 30000, we have to access 20001 data pages. This also includes page access till the end of tree, which is the height of the tree 'h' plus leaf access of $20001/15 = 1334$ pages. So in worst case,

Total page access needed is = 1334 + 20001 + h

3 Question 3: Composite Index

f(select EID=15) = 1/30,000

$$f(\text{select salary}>=75000) = \frac{(75000 - 30000)}{(40000 - 30000)}$$

= 0.68%

Solution:
Note, EID is primary index. Both criteria are not equally selective. All records with EID = 15 are stored together.

Given, Number of employee records 30,000.
Each page can contain 2 tuples, total 15,000 pages.
Each leaf can store 10 references of pages, total 1,500 leaves.

So, for (EID, salary): EID=15 will be in [15 / 2] = 8th page, thus expected page access is 8. h + 1.68

Now, for (salary, EID): Average salary is 95,000 [uniformly distributed].
In worst case, every employee gets salary of 95,000.
In that case, these 30,000 records will be in 15,000 pages, and thus total page access will be 15,000.

#leaves=
30000 * 1/10 * 0.68

4 Question 4: Index Implementation

¿java IndexExercise
Index exercise
==================================
Generating data...
Querystring: J21y4oIG41
==================================

```
Testing default
Creating Index: 0ms.
Executing Equality: 17ms.
Executing Greater: 23ms.
===================================
Testing dense
Creating Index: 0ms.
Index creation time - Greater: 43412ms.
Query time - Greater: 80ms.
Executing Equality: 18ms.
Executing Greater: 43493ms.
Equality result correct.
Greater result correct.
===================================
```

```
¿java IndexExercise
Index exercise
===================================
Generating data...
Querystring: ZAlmyKAHQp
===================================
Testing default
Creating Index: 1ms.
Executing Equality: 20ms.
Executing Greater: 22ms.
===================================
Testing dense
Creating Index: 0ms.
Index creation time - Greater: 45748ms.
Query time - Greater: 65ms.
Executing Equality: 19ms.
Executing Greater: 45814ms.
Equality result correct.
Greater result correct.
===================================
```

For the second (greater) query, index build time takes over 4000 ms, but query is executed within less than 100ms.