School of Computer Science, McGill University

COMP-421B Database Systems, Winter 2014

Written Assignment 3: Query Evaluation

April-1 11:00

A drugstore retail chain sells various products. `buyingPrice` is the price the chain pays for buying one unit of the product from the producer. The chain has stores in several places. Not all stores sell all products. Products may have different selling prices in the different stores. `sellingPrice` is the price the product costs in the given store. `inStock` indicates how many items of the given product are in stock at the given store.

`Products(pid:INT, pname:VARCHAR(40), type:VARCHAR(40), producer:VARCHAR(40), buyingPrice:FLOAT)`
e.g.: (7225, 'Aspirin 120 extra', 'Drug', 'Bayer', 4.95)

`Stores(storeId:CHAR(20), address:VARCHAR(250), manager: VARCHAR(60))`
e.g.: ('MontrealBranch12456', '4075 St. Denis, Montreal, X2A 2Y7, Montreal','Marieu')

`StorePrices(storeId:CHAR(20), pid:INT, sellingPrice:FLOAT, inStock:INT)`
e.g., : ('MontrealBranch12456', 7225, 6.95, 100)

INT and FLOAT have 4 Bytes, a char has 1 Byte. The relation `Products` has around 20,000 tuples on 600 data pages. All the varchar attribute values have on average 30 characters. The relation `Stores` has 1000 tuples that are stored on 80 pages. Each store sells around 20% of the possible products. Thus, the number of tuples in `StorePrices` is 0.2*20,000*1000 = 4 Mio. And the number of data pages for `StorePrices` is around 40,000.

Exercise 1: Index (15 Points)

There is an indirect B+-tree for `Products` on the combination of attributes `type` and then `producer`. You can assume that in the inner pages, a prefix of the `type` and `producer` strings are used for search where each type resp. producer prefix has 20 Bytes. For the leave pages, the full producer and type strings are used (which are, on average, 30 Bytes). There exist around 200 different types and 50 different producers . Products are uniformly distributed among the types and producers and the two attributes are independent of each other.

Each *rid* has 10 Bytes, each pointer (of internal index pages) has 6 Bytes. Leaf pages are filled around 70%. An index page has 4KBytes (use 4000 Bytes).

Calculate

1. *total number of data entries, the number of rids per data entry, and the size of a data entry,*
2. *the height of the tree, and the number of pages on each level of the tree.*

Exercise 2: Query execution (45 Points)

1. (15 Points) One typical query is

```
SELECT type, count(*), count(DISTINCT producer)
FROM Products
GROUP BY type
HAVING count(*) > 300
```

   Give two different strategies of how to execute the query, and give a rough estimation of the number of I/O for each of these strategies.

2. (30 Points) Another typical query checks for a product X (input variable of application program) which stores need to reorder the item (depending on Y, also given as input parameter).

```
SELECT storeID
FROM  StorePrices
WHERE pid = X AND inStock < Y
```

   The values for `inStock` are uniformly distributed between 1 and 500.

   a.) Indicate the access costs (in number of pages retrieved leading to I/O) for this query if

   i. you do not use any index

   ii. you use an unclustered index on `pid`

   iii. you use an unclustered index on `inStock`

   iv. you use both indexes

   Give the access costs both in general (for X and Y), and for the concrete values: X=200 and Y=10.

   b.) Would a clustered index on either `storeId` or `InStock` increase performance? Give a short explanation.

Exercise 3: Joins (25 Points)

Now assume there is an index on `pid` on `Products` and on `pid` on `storePrices`

1. Estimate the number of output tuples
2. Calculate the estimated I/O (a bit more than 100 buffer pages available)

   a.) index nested loop join between `Products` and `StorePrices` and `Products` is outer relation

   b.) index nested loop join between `Products` and `StorePrices` and `StorePrices` is outer relation

   c.) block nested loop join between `Products` and `StorePrices` and `Products` is outer relation

   d.) block nested loop join between `Products` and `StorePrices` and `StorePrices` is outer relation

   e.) sort merge join between `Products` and `StorePrices` (100 buffer pages available)

Exercise 4: Optimization (15 Points)

Now look at the query

```
SELECT P.pid, P.pname, S.storeID
FROM   Products P, Stores S, StorePrices SP
WHERE  P.pid = SP.pid
  AND  S.storeID = SP.storeID
  AND  (SP.sellingPrice * SP.inStock < 100)
  AND S.address Like '%Montreal%'
```

You can assume an average sellingPrice of 10$ and 100 stores in Montreal. inStock is again between 1 and 500. A non-optimized relational expression for this query is

$\pi_{pid,pname,storeID}(\sigma_{address\_contains\_Montreal \wedge sellingPrice * inStock < 100}(Products \times Stores) \bowtie StorePrices))$

Perform an algebraic optimization of your expression according to the rules discussed in class and show the operator tree. Indicate the types of joins you would perform. Assume that the only indices that exists are those on the primary key. Give an estimate of the number of tuples that flow from one operator to the next. These numbers, of course, depend on the selectivity of certain attributes. Make reasonable assumptions, and indicate your assumptions.