Fall 2007 CS186 Discussion Section:
Weeks 10, 10/29 - 11/02

Your friendly TAs

October 31, 2007

1 Relational Operators

1. In this question, you will consider joining two relations from an ecommerce database. The `CARTS` relation represents shopping cars of items. the `CONTENTS` relations has the contents of all the carts, with a foreign key called `cartID` to `CARTS`:

```
CARTS(cartID, customerID, date, comment)
CONTENTS(cartID, productID, quantity, price)
```

Assume the following:

- fixed size tuples in both relations
- 10 `CARTS` tuples per page
- 100 `CONTENTS` tuples per page
- 1000 pages in the `CARTS` relation
- 5000 pages in the `CONTENTS` relation
- 1002 frames in the buffer pool
- join on `CARTS.cartID=CONTENTS.cardID`

(a) How many I/Os are required for a page-oriented nested loops join, with `CARTS` as the outer relation and `CONTENTS` as the inner relation? Assume that the buffer manager is not used (i.e. every page reference generates an I/O).

$p(CARTS) + p(CARTS) * p(CONTENTS) =$
$1,000 + 1,000 * 5,000 = 5,001,000$

(b) Consider the previous question.
Assuming that the buffer manager is fully used and:

- The buffer manager starts empty
- One output buffer frame is pinned by the join, used to hold output tuples until they are ready to be flushed to disk. This frame is not unpinned by the join until the end of the query.
- One input buffer frame is pinned by the join and holds the current page of the outer relation at all times. All I/Os to the outer relation are placed explicitly into this frame, which is not unpinned until the end of the query

Symbol	Meaning
$p(R)$	number of blocks of relation R
$t(R)$	number of tuples of relation R
$l(R)$	number of leaf pages (data entry pages) of B index of relation R
sel	selectivity factor

Table 1: Symbolisms used in the solution of the cost estimation exercises.

- The buffer manager is running MRU replacement policy
- This is a single query system

Given these assumptions, how many I/Os are required for a page-oriented nested loops join?

This question is a bit tricky. We assume that we retain one buffer frame to hold a page from the **CARTS** *relation, one buffer frame for the output page and the rest (1002-2) for the inner relation. During the first scan of the inner relation, we read as many pages as we can (1000), one page at a time, and we perform the joins with that current page and the pinned inner relation page. When we attempt to read the 1001st page of the inner relation, according to the MRU replacement policy, we replace the very last page we read (1000th), and we keep replacing that page, until we finish scanning the inner relation for the first time.*

Then we replace the first page of the outer relation with its second block, we pin it, and we rescan the inner. But now, the first 999 blocks of **CONTENTS** *(and the 5000th one) are already in memory (we avoid 999 reads). When the 1000th block is to be read, we evict the most recently used, bring the new one in, and we keep evicting the same, until we scan the whole inner relation for the second time. This process is repeated for $p(CARTS) - 2$ more times.*

To summarize, the **CARTS** *relation is read only once, the* **CONTENTS** *is read fully once for the first iteration, and for the remaining $p(CARTS) - 1$ iterations, not all but $p(CONTENTS) - 999$ blocks are read from disk. Thus:*

$p(CARTS) + p(CONTENTS) + (p(CARTS) - 1) * (p(CONTENTS) - 999) =$
$1,000 + 5,000 + (999 * 4,001) = 4,002,999$

(c) How many I/Os are required for a block nested loops join, with **CONTENTS** as the outer relation, **CARTS** as inner relation, and 1000 pages per block?

$p(CONTENTS) + ceil(p(CONTENTS)/1,000) * p(CARTS) =$
$5,000 + 5 * 1,000 = 10,000$

(d) Assume there are only 52 frames in the buffer pool total. Give the I/Os for the following join algorithms.

- Hash Join, using CARTS to build the hash table in the second phase

  Since ceil(p(CARTS)/51) fits in memory and **CARTS** *is the smaller relation, we only need 2 passes of all data.*

  $3 * (p(CONTENTS) + p(CARTS)) = 3 * (1,000 + 5,000) = 18,000$

- Sort-Merge Join

  The number of passes we need to sort **CONTENTS** *is: $ceil(\log_{51} \frac{p(CONTENTS)}{52}) = 2$ and the number of memory blocks occupied during the second pass is $ceil(\frac{p(CONTENTS)}{2*52}) = 49 < 51$.*

  Similarly, for **CARTS** *is: $ceil(\log_{51} \frac{p(CARTS)}{52}) = 2$ and the number of memory blocks occupied during the second pass is $ceil(\frac{p(CARTS)}{2*52}) = 10 < 51$.*

  However, $10 + 49 > 51$, so we can't combine the last passes with the merge phase. Thus:

  $p(CONTENTS) + p(CARTS) + Sort(p(CARTS)) + Sort(p(CONTENTS)) =$
  $5,000 + 1,000 + 4 * 1,000 + 4 * 5,000 = 30,000$

- Block Nested Loops, CARTS as outer, block-size=50
  $p(CARTS) + ceil(p(CARTS)/50) * p(CONTENTS)$
  $1,000 + 20 * 5,000 = 101,000$

2. Consider the same database in Question 1, now running a selection range query on `CONTENTS.quantity` (e.g. `CONTENTS.quantity > c`). Let the selectivity of the query be $sel = 0.4$.

(a) How many I/Os do we need if the whole table is scanned and `CONTENTS` is unsorted?

Since `CONTENTS` relation is unsorted, we have to scan it from its first to its last page to find all the qualifying records. Thus:
$$p(CONTENTS) = 5,000$$

(b) What if it is sorted on `CONTENTS.quantity`?

If the relation is sorted on `CONTENTS.quantity`, then with a simple binary search scheme, we can find the first qualifying record, and then scan from that record on the whole file.

$$ceil(binary\text{-}search(p(CONTENTS))) + ceil(p(CONTENTS) * sel) =$$
$$ceil(\log p(CONTENTS)) + ceil(p(CONTENTS) * sel) =$$
$$13 + ceiling(5000 * .4) = 2,013$$

(c) How many I/Os are required given the following:
   - unclustered 3 level B+ tree index on CONTENTS.quantity.
   - data distributed evenly among data record pages
   - 500 data entries/page
   - query only requires rids of the CONTENTS records.

Traverse the index to locate the first qualifying data entry, and then scan all the data entry pages from that data entry on, as the B+ structure guarantees that the data entries will be in sorted order.

$$Indexlookup + ceil(l(CONTENTS) * sel) - 1 =$$
$$3 + ceil(\frac{5000 pages * 100 records/page}{500 data\ entries/page} * 0.4) - 1 =$$
$$2 + 400 = 402$$

(d) What if the previous query requires entire data records?

Steps:
   i. *Look up first value in index*
   ii. *look at all necessary data pages*
   iii. *sort rids by data record page ID*
   iv. *fetch data records*

Assuming
   - *All needed data entry pages fit in memory*
   - *each data entry page read only once*
   - *data evenly distributed on data record pages*

The cost is the same as before, only that now we have to retrieve the actual data records from the relation file, in which they are randomly placed among its pages (unclustered index). We could measure that last part of the cost in three ways:
   - Utterly pessimistic: We don't follow the optimization of sorting the rids by data record page RID. Thus, we can visit a page more than once. The number of pages we could read from the disk could be as high as the number of qualifying tuples: $t(CONTENTS) * sel$.
   - Pessimistic: Because there might be many data records that store each qualifying `CONTENTS.qualtity` value, we could assume that there is a tuple with a qualifying `CONTENTS.qualtity` value in each page of the CONTENTS file. Thus we could end up reading the whole file!
   - Optimistic / Realistic: Depending on the assumptions we make, we can reduce the cost of this final step; the selectivity corresponds to the percentage of **unique** values ought to be retrieved (e.g. $\frac{maxQuantity - c}{maxQuantity - minQuantity}$). In general, there might be many qualifying data records that store each `CONTENTS.qualtity` value, but because the uniform distribution assumption does hold, the above ratio preserves correctness! Therefore, the number of tuples to be returned, and consecutively, the number of blocks to be read (again because of the uniform distribution assumption) will be simply $p(CONTENTS) * sel$.

$Indexlookup + ceil(l(CONTENTS) * sel) - 1 + ceil(p(CONTENTS) * sel) =$
$3 + 400 - 1 + 2000 = 2402$

(e) Let there be an index similar to the previous two questions. Assume that each data records page has a 10% chance of having an overflow page. How many I/Os are required using the index if the query requires entire data records?

$Indexlookup + ceil(l(CONTENTS) * sel) - 1 + ceil(p(CONTENTS) * sel * 1.1) =$
$3 + 400 - 1 + 2000 * 1.1 = 2602$

(f) Let there be an unclustered 3 level B+ tree index on (`CONTENTS.price`, `CONTENTS.quantity`). How many I/Os are required using the index if the query requires the entire data record?

The first key of the index is not `CONTENTS.quantity`, so we cannot use this index. We would have to resort to a table scan, with an I/O cost from part a) or b) depending on if `CONTENTS` is sorted on quantity.