CS 245
Midterm solution set – Winter 1999

This solution set was assembled from actual answers that students provided. For many of the problems in the exam, there may have been one or more approaches to solving the problem– this solution set simply provides an example of answers that received full credit for each problem.

1. **(10 pts)** Please indicate if the following statements are TRUE or FALSE. If false, please write one sentence explaining why. (If true, then just write the word TRUE.)

   (a) In query optimization, it is always better to perform projections as early as possible.

   `FALSE`. Refer to textbook example 7.13

   (b) Sequential IOs are much more expensive than random IOs.

   `FALSE`. Sequential IOs are cheaper because they save on disk seek time and rotational latency

   (c) Generally, when the sizes of join relations are big, nested loop joins perform better than hash joins.

   `FALSE`. Nested loop join is quadratic (propotional to the product of the sizes of two relations,) while hash join is linear (propotional to the sum of the sizes of two relations.)

   (d) To modify a block on disk, the block must we read into memory first.

   `TRUE`.

   (e) A second level index for an indexed sequential file is usually dense.

   `FALSE`. Second level index are always sparse, since it does not make any sense to have a dense second-level index.

   (f) A (plain) B tree may be superior to a B+ tree if variable length bufferes and disk blocks are available.

   `TRUE`.

   (g) If $V(C, R) = 1000$ and $n_R = 10,000$, we can expect that a selection on R on the condition $C = 5$ will yield a relation with 10 tuples.

   `TRUE`.

   (h) Natural join is associative and commutative.

   `TRUE`.

   (i) For the queries of the form $\sigma_{A=a}(R)$, extensible hashing is better than B-tree.

   `TRUE`.

   (j) For the queries of the form $\sigma_{A>a}(R)$, extensible hashing is better than B-tree.

   `FALSE`. B-tree is more efficient in range queries.

2. **(10 pts)**

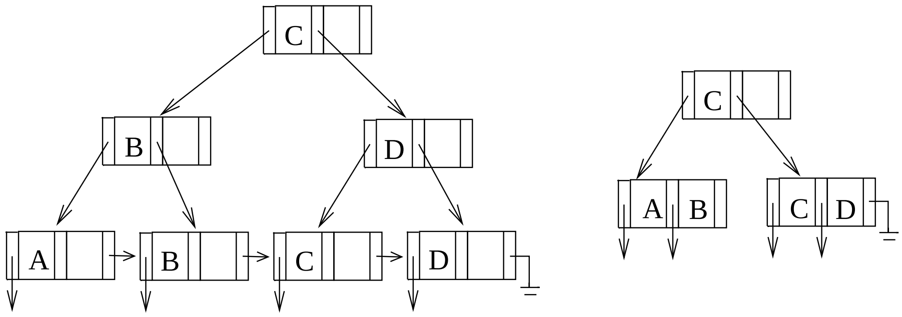
Let $R(A, B, C)$ and $S(C, D, E, F)$ be two relations. Transform the following relational algebra expression into an equivalent relational algebra expression in which selections and projections are performed as early as possible. (Assume that the relations are sets, not bags.)

$$\Pi_{A,D}[\sigma_{A=3 \text{ and } R.C=S.C}\{\Pi_{A,C}(R) \times \Pi_{C,D,E}(S)\}]$$

$$Answer : \Pi_{A,D}[\Pi_{A,C}(\sigma_{A=3}(R)) \bowtie \Pi_{C,D}(S)]$$

3. **(10 pts)**

Assume that $n = 3$ (i.e. the number of pointers in a node is 3) and give an example of a B+ tree of height $j = 3$ that could be reorganized into a 2 level tree with exactly the same keys. Your example should consists of two trees: one, the 3-level tree, and two, the equivalent 2-level tree. Your examples should show all the keys and pointers in the leaf and non-leaf nodes of the tree. Use letters A, B, C, ... to represent the keys. (Assume A < B < C ...).



4. **(20 pts)**

After graduating from Stanford, you decided to work for a new start-up company called Amazon.Com as their lead database engineer, and you have been given the challenging responsibility of designing Amazon's customer database.

So far, you have come up with the following physical schema encoded in DDL as below:

```
CREATE TABLE CUSTOMER (
        CUSTOMER_ID             INTEGER,
        LAST_NAME               CHARACTER(80),
        MIDDLE_INITIAL          CHARACTER(1),
        FIRST_NAME              CHARACTER(80),
        CREDIT_CARD_NO          CHARACTER(10),
        SNAIL_MAIL_ADDRESS      CHARACTER(256),
        EMAIL_ADDRESS           CHARACTER(80),
        PREFERRED_CUSTOMER      BOOLEAN);

CREATE TABLE ORDER (
        ORDER_ID                LONG,
        DATE_ORDERED            DATE,
        CUSTOMER_ID             INTEGER,
        COMMENT                 CHARACTER(64),
        TOTAL_PRICE             REAL);

CREATE TABLE BOOK_ORDER (
        ORDER_ID                LONG,
        BOOK_ID                 INTEGER);
```

`BOOLEAN`s and `CHARACTER`s are 1-byte. `INTEGER`s are 4-bytes. `LONG`s and `DATE`s are 8-bytes.

The `CUSTOMER` relation has one record for each Amazon.Com customer that registers with the web site. Each time the user creates an order, one record is inserted into the `ORDER` relation. A customer may buy any number of books in a single order. For each book that is bought as part of an order, one record is inserted into the `BOOK_ORDER` relation.

Jeff Bezos, the company's CEO, is highly (perhaps even unrealistically) optimistic about the success of Amazon.Com, and expects that in the first five years Amazon will have 2^{25} (just over 32 million!) customers. He also expects that each customer will have placed, on the average, four orders for three books (i.e., will have purchased 12 books.) In answering the following questions, keep these requirements in mind.

Part (a)

You want to determine how many 8GB disks you need to buy to support Jeff's requirements. Also, to be on the safe side, you only want your disks to be 50% full after the first five years of operation.

What is the minimal number of disks that you need to buy? (Assume that your database system is able to span tables across multiple disks. Do not consider the space for indices or meta data)

Size of `CUSTOMER` record = 512 bytes = 2^9 bytes
Size of `ORDER` record = 92 bytes
Size of `BOOK_ORDER` record = 12 bytes
Number of `CUSTOMER` records = 2^{25}
Number of `ORDER` records = $2^{25} \times 4 = 2^{27}$
Number of `BOOK_ORDER` records = $2^{25} \times 12 = 3 \times 2^{27}$

Total space to store all these records = $2^9 \times 2^{25} + 92 \times 2^{27} + 12 \times 3 \times 2^{27} = 32$ GB

Since we want the disks to be only half full, we need 64 GB.
Therefore, we need **8** disks.

Part (b)

Assume that the disks you are considering buying have the following characteristics, and that you are interested in assessing the performance of these disks:

* 8GB usable capacity
* 16 surfaces
* 5400 RPM
* 1024 cylinders
* 10% overhead on each track
* 8KB blocks (sectors are the same size.)

What is the sustained bandwidth per track?

Time for one rotation = $\frac{1}{90}$ sec
8GB (2^{33} bytes) are distributed over 2^4 surfaces with 2^{10} tracks per surface.
Therefore, each track has $\frac{2^{33}}{2^4 \times 2^{10}} = 2^{19}$ bytes.

Hence, sustained bandwidth = $\frac{2^{19}}{\frac{1}{90}} = 45$ MB/s

Part (c)

To build indicies over our tables split across many disks, we would need to design block pointers that can point across disks. That is, given a block pointer, we need to be able to determine which disk, as well as which block on the disk, the pointer points to. In this part, we will determine how big these block pointers need to be.

(i) How many blocks are on one disk?

Number of blocks/disk = $2^{33}/2^{13} = 2^{20}$ blocks.

(ii) How many bits do we need to be able to address all the blocks on one disk?

We need at least 20 bits to address all blocks on one disk.

(iii) Assuming that we don't expect that our database system will ever use more than 16 disks total, how many bits do we need to address a disk?

We need $\lceil \log_2 16 \rceil = 4$ bits.

(iv) What is the total number of bits that we need to address a specific block on a specific disk?

We need $20 + 4 = 24$ bits.

(v) How big (in bytes) should our block pointers be?

Block pointers should be $\lceil 24/8 \rceil = 3$ bytes.

Part (d)

For this problem, assume that block pointers are 12 bytes long, irrespective your answer for problem 4(c). The value 12 bytes may or may not be the correct answer for 4(c), but use this value here.

(i) What is the minimum number of disk blocks that a dense primary index on `CUSTOMER_ID` for the `CUSTOMER` table would occupy?

When the records are contiguous, we only need to store a pointer to the first block. Then the location of all records can be calculated by the offset from the beginning of the index. Therefore, the index will have one block pointer followed by a sequence of key values giving the key of each record. A block has 8192 bytes and each key value is 4 bytes. Therefore, the number of key values in one block = $\lfloor \frac{8192}{4} \rfloor = 2048$ keys/block. But the first index block can have only 2045 keys since it also stores the block pointer pointing to the fist block of the file. Hence

$$\text{Number of blocks in the index} = 1 + \lceil \frac{2^{25} - 2045}{2048} \rceil = 16385$$

(ii) What is the minimum number of disk blocks that a sparse primary index would occupy?

Since each tuple of the CUSTOMER relation is of size 2^9 bytes, one block contains $\lfloor \frac{2^{13}}{2^9} \rfloor = 2^4$, and we need $\lceil \frac{2^{25}}{2^4} \rceil = 2^{21}$ blocks. Therefore the sparse primary index will have 2^{21} entries. To store these entries, we need at least $1 + \lceil \frac{2^{21}-2045}{2048} \rceil = 1025$ blocks.

(iii) If we expect that most queries for a CUSTOMER_ID will be for ids that are actually present in the database, which index should we use and why? Assume that records in the CUSTOMER table are all stored contiguously. Assume that we need 1 byte to identify a record within a block. Assume that all 8KB of a block is available for storage (no block header.)

If most queries are for ids present in the database, then a *sparse* index is preferred. It is because given that most queries will have a non-zero result set, we will not be able to make use of one of the important features of a dense index — which is that we can determine availability of a tuple without accessing the relation. Since this feature is not going to be useful, it does not make sense to waste space on a dense index.