CSE 444 Practice Problems

DBMS Architecture

1. **Data Independence**

   (a) What is physical data independence?

   Solution:
   Physical data independence is a property of a DBMS that ensures decoupling between the physical layout of data and applications which access it. In other words, with physical independence, applications are insulated from changes in physical storage details: changes in how the data is stored do not cause application changes.

   (b) What properties of the relational model facilitate physical data independence?

   Solution:
   Declarative query language or set-at-a-time query language.

(c) What is logical data independence?

Solution:
Logical data independence is a property of a DBMS that ensures decoupling between the logical structure of data and applications that operate on it. With this property, changes in the logical data layout like tables, rows, and columns do not require application to be changed.

(d) How can one provide a high level of logical data independence with the relational model?

Solution:
By defining views.

2. **High-Level DBMS Architecture**

   You should know the key components of a relational DBMS. Please see the lecture notes for an overview.

   You should also be able to discuss what you implemented in the labs. For example, we could ask you for a high-level description of your buffer manager.

3. **Data Storage and Indexing**

   You should be able to show what happens when one adds/removes data to/from a B+ Tree. Please see the web quizzes for some good examples.

4. **Data Storage and Indexing**

Suppose we have a relation `R(a,b,c,d,e)` and there are at least 1000 distinct values for each of the attributes.

Consider each of the following query workloads, independently of each other. If it is possible to speed it up significantly by adding up to **two** additional indexes to relation R, specify for each index (1) which attribute or set of attributes form the search key of the index, (2) if the index should be clustered or unclustered, (3) if the index should be a hash-based index or a B+-tree. You may add at most two new indexes. If adding a new index would not make a significant difference, you should say so. Give a brief justification for your answers.

(a) 100,000 queries have the form: select * from R where $b <?$
10,000 queries have the form: select * from R where $c =?$

Solution:
Since we need efficient range-queries on R(b), we definitely want a clustered, B+-tree index on R(b).

For the second query, an index on R(c) will help. It can be either a B+-tree or a hash-based index since queries look-up specific key values. The index must be unclustered since the index on R(b) is clustered. That is fine, however, since queries will look-up specific key values and we know that there are many distinct values in the relation.

(b) 100,000 queries have the form: select * from R where $b <?$ and $c =?$
10,000 queries have the form: select * from R where $d =?$
1,000 queries have the form: select * from R where $a =?$

Solution:
For the first query, a clustered B+-tree index on R(c,b) would be most helpful since we could use it to look-up all data items that match both the given value on c and the range on b.

Since we can only add a second index, we will favor the most frequent query and add an index on R(d). As in the question above, this index must be unclustered and can be either a B+-tree or a hash-based index.

(c) 100,000 queries have the form: select a, c from R where $b <?$
10,000 queries have the form: select * from R where $d <?$

Solution:
Since both queries are range-selection queries, we need clustered indexes for both of them, but we cannot have more than one such index. However, we can have a *covering* index.
We thus recommend:

- A clustered, B+-tree index on R(d).
- An unclustered, B+-tree index on R(b,a,c). This is also a covering index in that we only need to use the index to answer the query. We don't need to touch the data.

5. **Relational Algebra and Query Processing**

Consider three tables R(a,b,c), S(d,e,f), and T(g,h,i).

(a) Consider the following SQL query:

```
SELECT   R.b
FROM     R, S, T
WHERE    R.a = S.d
         AND S.e = T.g
         AND T.h > 21
         AND S.f < 50
GROUP BY R.b
HAVING count(*) > 2
```

For each of the following relational algebra expressions, indicate if it is a correct translation of the above query or not.

i. $\pi_{R.b}(\sigma_{\text{TOTAL}>2}(\gamma_{R.b,count(*)\rightarrow\text{TOTAL}}(\sigma_{T.h>21 \text{ AND } S.f<50}(R \bowtie_{R.a=S.d} (S \bowtie_{S.e=T.g} T)))))$

CORRECT INCORRECT

Solution:
CORRECT.

ii. $\pi_{R.b}(\sigma_{\text{TOTAL}>2}(\gamma_{R.b,count(*)\rightarrow\text{TOTAL}}(R \bowtie_{R.a=S.d} ((\sigma_{S.f<50}(S)) \bowtie_{S.e=T.g} (\sigma_{T.h>21}(T))))))$

CORRECT INCORRECT

Solution:
CORRECT.

iii. $\pi_{R.b}(\sigma_{\text{TOTAL}>2}(\sigma_{T.h>21 \text{ AND } S.f<50}(\gamma_{R.b,count(*)\rightarrow\text{TOTAL}}(R \bowtie_{R.a=S.d} (S \bowtie_{S.e=T.g} T)))))$

CORRECT INCORRECT

Solution:
INCORRECT.

(b) For the following SQL query, show an equivalent relational algebra expression. You can give the expression in the same format as we used above or you can draw it in the form of an expression tree or logical query plan.

```
SELECT R.b
FROM R, S
WHERE R.a = S.d
AND R.b NOT IN (SELECT R2.b FROM R as R2, T WHERE R2.b = T.g)
```

Solution:
$\pi_{R.b}(R \bowtie_{R.a=S.d} S) - \pi_{R.b}(R \bowtie_{R.b=T.g} T)$

(c) A user just connected to a database server and submitted a SQL query in the form of a string. Give **four** important steps involved in evaluating that SQL query **and the order** in which they are performed. You only need to name the steps. No need to explain them.

Solution:

(d) Query parsing: the DBMS parses the SQL string into an internal tree representation of the query. Typically, a variety of checks are also performed at that time including syntax checks, authorization checks, simple integrity constraint checks, etc.

(e) Query rewrite: the parse-tree is converted into an initial logical query plan. During that step, views are also rewritten and queries are flattened if possible.

(f) Query optimization: the query optimizer searches for an efficient physical query plan to execute the query.

(g) Query execution: the DBMS actually executes the query and returns the results to the user.

(h) What is the difference between a logical and a physical query plan?

Solution:
A logical query plan is an extended relational algebra tree. A physical query plan is a logical query plan with extra annotations that specify the (1) access path to use for each relation (whether to use an index or a file scan and which index to use), the (2) implementation to use for each relational operator, and (3) how the operators should be executed (pipelined execution, intermediate result materialization, etc.).

6. **Relational Algebra and Query Processing**

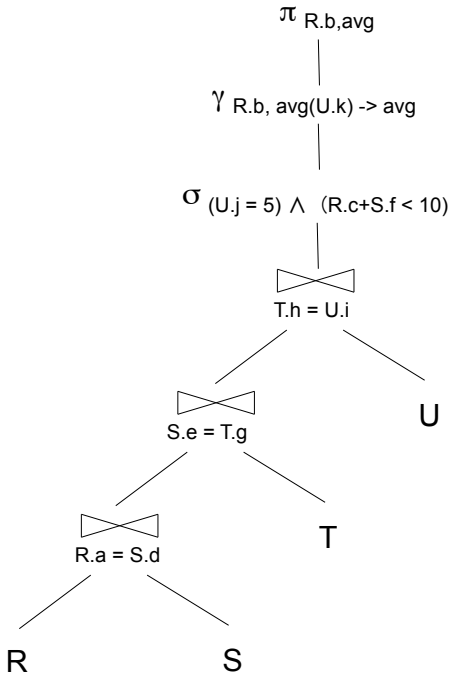
Consider four tables R(a,b,c), S(d,e,f), T(g,h), U(i,j,k).

(a) Consider the following SQL query:

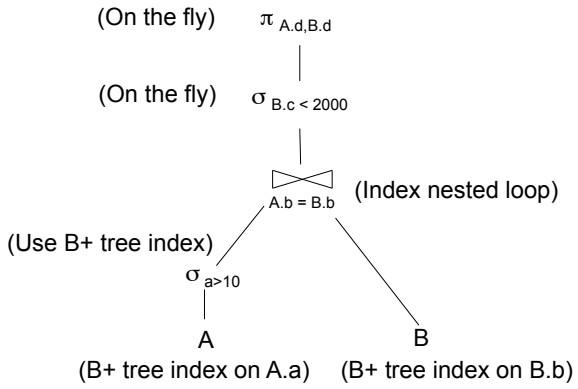
```
SELECT   R.b, avg(U.k) as avg
FROM     R, S, T, U
WHERE    R.a = S.d
         AND S.e = T.g
         AND T.h = U.i
         AND U.j = 5
         AND (R.c + S.f) < 10
GROUP BY R.b
```

Draw a *logical* plan for the query. You may chose any plan as long as it is correct (i.e. no need to worry about efficiency).

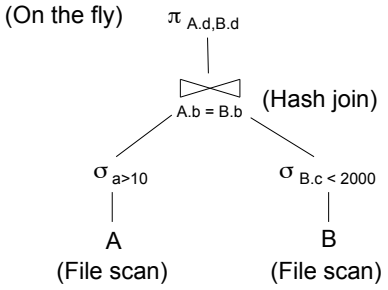
Solution:
Many solutions were possible including:



(b) Consider the following two physical query plans. Give **two** reasons why plan B may be faster than plan A. **Explain** each reason.



Plan A



Plan B

Solution:
The following are three possible reasons why plan B could be faster than plan A:

- The low selectivity of the selection predicate ($a > 10$) and an unclustered index on A.a may make a file scan of A faster than using the index.
- If there are lots of matches, hash joins may be faster than index nested loops. The hash join reads its input only once (unless the input is too large). The index-nested loop may end-up reading the same pages of B multiple times.
- Pushing the selection ($B.c < 2000$) down can get rid of lots of B tuples before the join, reducing the cost of that operation.

7. **Operator Algorithms**

Relation R has 90 pages. Relation S has 80 pages. Explain how a DBMS could efficiently join these two relations given that only 11 pages can fit in main memory at a time. Your explanation should be **detailed**: specify how many pages are allocated in memory and what they are used for; specify what exactly is written to disk and when.

(a) Present a solution that uses a hash-based algorithm.

Solution:
The algorithm proceeds as follows:

- First, we split R into partitions:
  - Allocate one page for the input buffer.
  - Allocate 10 pages for the output buffers: one page per partition.
  - Read in R one page at the time. Hash into 10 buckets. As the pages of the different buckets fill-up, write them to disk. Once we process all of R, write remaining incomplete pages to disk. At the end of this step, we have 10 partitions of R on disk. Assuming uniform data distribution, each partition comprises 9 pages.
- Then, we split S into partitions the same way we split R (must use same hash function).
- For each pair of partitions that match:
  - Allocate one page for input buffer
  - Allocate one page for the output buffer.
  - Read one 9-page partition of R into memory. Create a hash table for it using a different hash function than above.
  - Read corresponding S partition into memory one page at the time. Probe the hash table and output matches.

(b) Present a solution that uses a sort-based algorithm.

Solution:
The algorithm proceeds as follows

- First, we sort R:
  - Read 10 pages worth of R tuples into memory, sort, and write to disk.
  - Repeat for next 10 pages until all R tuples have been processed.
  - Now we have 9 runs of 10 pages sorted on disk.
  - Allocate one page per run and one page for the output.
  - Merge runs in sorted order and output into file.
- Sort S the same way that we sorted R above.
- Read S and R one page at the time. Merge them and output matches.

Note that we can be more efficient if we use a priority queue and output runs of length twice the size of memory in the first step of the algorithm. Then, we can join R and S together while merging their individual runs.