Questions

1. Consider the Extendible Hashing index shown in Figure 1.1. Answer the following questions about this index:

   1. What can you say about the last entry that was inserted into the index?
   2. What can you say about the last entry that was inserted into the index if you know that there have been no deletions from this index so far?
   3. Suppose you are told that there have been no deletions from this index so far. What can you say about the last entry whose insertion into the index caused a split?
   4. Show the index after inserting an entry with hash value 68.
   5. Show the index after inserting entries with hash values 17 and 69.
   6. Show the index after deleting the entry with hash value 21.
      (Assume that the full deletion algorithm is used.)
   7. Show the index after deleting the entry with hash value 10.
      Is a merge triggered by this deletion? If not, explain why. (Assume that the full deletion algorithm is used.)
Ans:

1. It could be any one of the data entries in the index. We can always find a sequence of insertions and deletions with a particular key value, among the key values shown in the index as the last insertion. For example, consider the data entry 16 and the following sequence:

\[
1 \ 5 \ 21 \ 10 \ 15 \ 7 \ 51 \ 4 \ 12 \ 36 \ 64 \ 8 \ 24 \ 56 \ 16 \ 56 24 \ 8D
\]

The last insertion is the data entry 16 and it also causes a split. But the sequence of deletions following this insertion cause a merge leading to the index structure shown in Fig 11.1.

2. The last insertion could not have caused a split because the total number of data entries in the buckets \( A \) and \( A2 \) is 6. If the last entry caused a split the total would have been 5.

3. The last insertion which caused a split cannot be in bucket C. Buckets B and C or C and D could have made a possible bucket-split image combination but the total number of data entries in these combinations is 4 and the absence of deletions demands a sum of at least 5 data entries for such combinations. Buckets B and D can form a possible bucket-split image combination because they have a total of 6 data entries between themselves. So do \( A \) and \( A2 \). But for the B and D to be split...
images the starting global depth should have been 1. If the starting global depth is 2, then the last insertion causing a split would be in $A$ or $A_2$.

4. See Fig 11.2.

5. See Fig 11.3.

6. See Fig 11.4.

7. The deletion of the data entry 10 which is the only data entry in bucket C doesn’t trigger a merge because bucket C is a primary page and it is left as a placeholder. Right now, directory element 010 and its split image 110 already point to the same bucket C. We can’t do a further merge. See Fig 11.5.
Hash-Based Indexing & Overview of Query Evaluation

Figure 11.3
Hash-Based Indexing & Overview of Query Evaluation

Figure 11.4

Figure 11.5
2. Consider the Linear Hashing index shown in Figure 1.2. Assume that we split whenever an overflow page is created. Answer the following questions about this index:

1. What can you say about the last entry that was inserted into the index?
2. What can you say about the last entry that was inserted into the index if you know that there have been no deletions from this index so far?
3. Suppose you know that there have been no deletions from this index so far. What can you say about the last entry whose insertion into the index caused a split?
4. Show the index after inserting an entry with hash value 4.
5. Show the index after inserting an entry with hash value 15.
6. Show the index after deleting the entries with hash values 36 and 44. (Assume that the full deletion algorithm is used.)
7. Find a list of entries whose insertion into the original index would lead to a bucket with two overflow pages. Use as few entries as possible to accomplish this. What is the maximum number of entries that can be inserted into this bucket before a split occurs that reduces the length of this overflow chain?

![Figure 1.2](image-url)
Hash-Based Indexing & Overview of Query Evaluation

Ans:

1. Nothing can be said about the last entry into the index: it can be any of the data entries in the index.
2. If the last item that was inserted had a hashcode \( h_0(keyvalue) = 00 \) then it caused a split, otherwise, any value could have been inserted.
3. The last data entry which caused a split satisfies the condition \( h_0(keyvalue) = 00 \) as there are no overflow pages for any of the other buckets. 4. See Fig 11.7
5. See Fig 11.8
6. See Fig 11.9
7. The following constitutes the minimum list of entries to cause two overflow pages in the index: 63, 127, 255, 511, 1023 The first insertion causes a split and causes an update of \( \text{Next} \) to 2. The insertion of 1023 causes a subsequent split and \( \text{Next} \) is updated to 3 which points to this bucket. This overflow chain will not be redistributed until three more insertions (a total of 8 entries) are made. In principle if we choose data entries with key values of the form \( 2^k + 3 \) with sufficiently large \( k \), we can take the maximum number of entries that can be inserted to reduce the length of the overflow chain to be greater than any arbitrary number. This is so because the initial index has 31(binary 11111), 35(binary 10011), 7(binary 111) and 11(binary 1011). So by an appropriate choice of data entries as mentioned above we can make a split of this bucket cause just two values (7 and 31) to be redistributed to the new bucket. By choosing a sufficiently large \( k \) we can delay the reduction of the length of the overflow chain till any number of splits of this bucket.
Hash-Based Indexing & Overview of Query Evaluation

**Figure 11.7** Index from Figure 11.6 after insertion of an entry with hash value 4
Hash-Based Indexing & Overview of Query Evaluation

Figure 11.8 Index from Figure 11.6 after insertion of an entry with hash value 15

Figure 11.9 Index from Figure 11.6 after deletion of entries with hash values 36 and 44
3. Briefly answer the following questions:

1. Describe three techniques commonly used when developing algorithms for relational operators. Explain how these techniques can be used to design algorithms for the selection, projection, and join operators.

2. What is an access path? When does an index match an access path? What is a primary conjunct, and why is it important?

3. What information is stored in the system catalogs?

4. What are the benefits of making the system catalogs be relations?

5. What is the goal of query optimization? Why is optimization important?

6. Describe pipelining and its advantages.

7. Give an example query and plan in which pipelining cannot be used.

8. Describe the iterator interface and explain its advantages.

9. What role do statistics gathered from the database play in query optimization?

10. What were the important design decisions made in the System R optimizer?

11. Why do query optimizers consider only left-deep join trees? Give an example of a query and a plan that would not be considered because of this restriction.

Ans:

1. The three techniques commonly used are indexing, iteration, and partitioning:

   **Indexing:** If a selection or join condition is specified, use an index to examine just the tuples that satisfy the condition.

   **Iteration:** Examine all tuples in an input table, one after the other. If we need only a few fields from each tuple and there is an index whose key contains all these fields, instead of examining data tuples, we can scan all index data entries.

   **Partitioning:** By partitioning tuples on a sort key, we can often decompose an operation into a less expensive collection of operations on partitions. Sorting and hashing are two commonly used partitioning techniques. They can be used to design algorithms for selection, projection, and join operators as follows:

   **Selection:** For a selection with more than one tuple matching the query (in general, at least 5%), indexing like B+ Trees are very useful. This comes into play often with range queries. It allows us to not only find the first qualifying tuple quickly, but also the other qualifying tuples soon after (especially if the index is clustered). For a selection condition with an equality query where there are only a few (usually 1) matching tuple,
partitioning using hash indexing is often appropriate. It allows us to find the exact tuple we are searching for with a cost of only a few (typically one or two) I/Os.

**Projection:** The projection operation requires us to drop certain fields of the input, which can result in duplicates appearing in the result set. If we do not need to remove these duplicates, then the iteration technique can efficiently handle this problem. On the other hand, if we do need to eliminate duplicates, partitioning the data and applying a sort key is typically performed. In the case that there are appropriate indexes available, this can lead to less expensive plans for sorting the tuples during duplicate elimination since the data may all ready be sorted on the index (in that case we simply compare adjacent entries to check for duplicates)

**Join:** The most expensive database operation, joins, can combinations of all three techniques. A join operation typically has multiple selection and projection elements built into it, so the importance of having appropriate indexes or of partitioning the data is just as above, if not more so. When possible, the individual selections and projections are applied to two relations before they are joined, so as to decrease the size of the intermediate table. As an example consider joining two relations with 100,000 tuples each and only 5 % of qualifying tuples in each table. Joining before applying the selection conditions, would result in a huge intermediate table size that would then have to be searched for matching selections. Alternatively, consider applying parts of the selection first. We can then perform a join of the 5,000 qualifying tuples found after applying the selection to each table, that can then be searched and handled significantly faster.

2. An access path is a way of retrieving tuples from a table and consists of either a file scan or an index plus a matching selection condition. An index matches a selection condition if the index can be used to retrieve just the tuples that satisfy the condition. An index can match some subset of conjunctions in a selection condition even though it does not match the entire condition and we refer to the conjunct that the index matches as the primary conjuncts in the selection. Primary conjuncts are important because they allow us to quickly discard information we do not need and only focus in on searching/sorting the data that more closely matches the selection conditions.

3. Information about relations, indexes, and views is stored in the system catalogs. This includes file names, file sizes, and file structure, the attribute names and data types, lists of keys, and constraints. Some commonly stored
statistical information includes:
(a) Cardinality - the number of tuples for each relation
(b) Size - the number of pages in each relation
(c) Index Cardinality - the number of distinct key values for each index
(d) Index Size - the number of pages for each index (or number of leaf pages)
(e) Index Height - the number of nonleaf levels for each tree index
(f) Index Range - the minimum present key value and the maximum present key value for each index.

4. There are several advantages to storing the system catalogs as relations. Relational system catalogs take advantage of all of the implementation and management benefits of relational tables: effective information storage and rich querying capabilities. The choice of what system catalogs to maintain is left to the DBMS implementor.

5. The goal of query optimization is to avoid the worst plans and find a good plan. The goal is usually not to find the optimal plan. The difference in cost between a good plan and a bad plan can be several orders of magnitude: a good query plan can evaluate the query in seconds, whereas a bad query plan might take days!

6. Pipelining allows us to avoid creating and reading temporary relations; the I/O savings can be substantial.

7. Bushy query plans often cannot take advantage of pipelining because of limited buffer or CPU resources. Consider a bushy plan in which we are doing a selection on two relations, followed by a join. We cannot always use pipelining in this strategy because the result of the selection on the first selection may not fit in memory, and we must wait for the second relation’s selection to complete before we can begin the join.

8. The iterator interface for an operator includes the functions open, get next, and close; it hides the details of how the operator is implemented, and allows us to view all operator nodes in a query plan uniformly.

9. The query optimizer uses statistics to improve the chances of selecting an optimum query plan. The statistics are used to calculate reduction factors which determine the results the optimizer may expect given different indexes and inputs.

10. Some important design decisions in the System R optimizer are:
(a) Using statistics about a database instance to estimate the cost of a query evaluation plan.
(b) A decision to consider only plans with binary joins in which the inner plan is a base relation. This heuristic reduces the often significant number of
alternative plans that must be considered.
(c) A decision to focus optimization on the class of SQL queries without nesting
and to treat nested queries in a relatively ad hoc way.
(d) A decision not to perform duplicate elimination for projections (except as a
final step in the query evaluation when required by a DISTINCT clause).
(e) A model of cost that accounted for CPU costs as well as I/O costs.
11. There are two main reasons for the decision to concentrate on left-deep plains only:
(a) As the number of joins increases, the number of alternative plans increases
rapidly and it becomes necessary to prune the space of the alternative plans.
(b) Left-deep trees allow us to generate all fully pipelined plans; that is, plans in
which all joins are evaluated using pipelining.
Consider the join A __ B __ C __ D. The query plan (A __ B) __ (C __ D)
would never be considered because it is a bushy tree.