Review Questions: Query Evaluation, Indexing, and Optimization

DSC 208R - Data Management for Analytics

Questions and Answers

Question 1

What is a logical query plan? How is it different from a physical query plan?

Answer:

A **Logical Query Plan (LQP)** is a Directed Acyclic Graph (DAG) whose vertices represent "Logical Operators" from Extended Relational Algebra. It describes *what* is computed in a query.

A **Physical Query Plan (PQP)** is also a DAG whose vertices are "Physical Operators." It specifies the exact algorithm or code to run for each logical operation, including all parameters, thus detailing *how* the query is computed. A single LQP can have many possible PQPs. The key difference is that LQP focuses on the "what," while PQP focuses on the "how" of query execution.

Question 2

Briefly explain 2 benefits of declarativity in data systems.

Answer:

Declarativity, often referred to as logical-physical separation or data independence, offers significant benefits in data systems:

1. **Increased User Productivity**: Users can focus on what data they want (the logical query) without needing to specify the exact execution steps, simplifying query writing.
2. **Automated Optimization and Scalability**: The RDBMS can automatically optimize the query code, leading to faster and more scalable performance without manual tuning by the user. Another benefit is **Application Portability**, meaning that changes to the internal system (physical implementation) do not require changes to the application code.

Question 3

Which index structure is useful only for equality predicates?

Answer:

A **Hash Index** structure is generally efficient and useful primarily for equality predicates ('='). It is not efficient for range queries (e.g., '¿', '¡', 'BETWEEN') or 'LIKE' pattern matching with wildcards at the beginning.

Question 4

Suppose you have two predicates, P1 with selectivity 20% and P2 with selectivity 2%. Which is said to be more selective?

Answer:

P2 with 2% selectivity is said to be more selective. Selectivity refers to the fraction of rows that satisfy a predicate. A lower percentage indicates that the predicate filters out a larger proportion of data, leaving a smaller, more specific result set. More selective predicates help reduce the amount of data that needs to be processed, which is beneficial for query performance.

Question 5

Are all composite indexes unique indexes? Are all unique indexes primary indexes?

Answer:

- **Are all composite indexes unique indexes? No.** A composite index is an index on multiple columns. It does not inherently guarantee uniqueness across those columns unless explicitly defined as a 'UNIQUE' index.
- **Are all unique indexes primary indexes? No.** A 'UNIQUE' index ensures that all values in the indexed column(s) are distinct. While a 'PRIMARY KEY' constraint inherently creates a unique index, not all 'UNIQUE' indexes are primary keys. A table can have multiple 'UNIQUE' constraints, but only one 'PRIMARY KEY'. The primary key also cannot contain 'NULL' values, whereas a unique index might allow one 'NULL' value (depending on the DBMS).

Question 6

Given a relation with 3 attributes, how many different B+ tree indexes can be built on this relation?

Answer:

Given a relation with 3 attributes (let's call them A, B, C), the number of different B+ tree indexes that can be built is calculated by considering all possible subsets of attributes and their permutations (since the order of attributes matters in composite B+ tree indexes).

- **Single-attribute indexes (1 attribute):** (A), (B), (C) = 3 options
- **Two-attribute composite indexes (2 attributes, order matters):** (A, B), (B, A), (A, C), (C, A), (B, C), (C, B) = 6 options
- **Three-attribute composite indexes (3 attributes, order matters):** (A, B, C), (A, C, B), (B, A, C), (B, C, A), (C, A, B), (C, B, A) = 6 options

Total number of different B+ tree indexes = $3 + 6 + 6 =$ **15** different B+ tree indexes.

Question 7

What are the 2 main parts of a typical RDBMS query optimizer?

Answer:

The two main parts of a typical RDBMS query optimizer are:

1. **Plan Enumerator**: This component explores and generates alternative Physical Query Plans (PQPs) for a given Logical Query Plan (LQP). It does this by applying algebraic rewrite rules to transform LQPs and by selecting different physical operators for logical operations.

2. **Plan Cost Estimator**: This component calculates an analytically determined runtime cost for each generated PQP. This estimation helps the optimizer choose the most efficient plan.

Question 8

Why is a selection pushdown rewrite often faster?

Answer:

A selection pushdown rewrite is often faster because it performs the filtering (selection) operation *before* other more expensive operations, such as joins. By applying the selection predicate as early as possible (pushing it down the query plan tree), it significantly reduces the size of the intermediate data sets that need to be processed by subsequent operations (like joins). Smaller intermediate data sets lead to fewer I/O operations and less CPU processing, resulting in faster query execution.