6. ALGORTIHM FOR SELECT AND JOIN OPERATION

BASIC ALGORITHMS FOR EXECUTING RELATIONAL QUERY OPERATIONS

- An RDBMS must include one or more alternative algorithms that implement each relational algebra operation (SELECT, JOIN.) and, in many cases, that implement each combination of these operations.
- Each algorithm may apply only to particular storage structures and access paths (such index.).
- Only execution strategies that can be implemented by the RDBMS algorithms and that apply to the particular query and particular database design can be considered by the query optimization module.
- These algorithms depend on the file having specific access paths and may apply only to certain types of selection conditions.
- We will use the following examples of SELECT operations:

  (OP1): σSSN=_123456789‘ (EMPLOYEE)

  (OP2): σ DNUMBER > 5 (DEPARTMENT)

  (OP3): σDNO=5 (EMPLOYEE)

  (OP4): σDNO=5 AND SALARY>30000 AND SEX = _F‘ (EMPLOYEE)

  (OP5): σESSN=_123456789‘ AND PNO=10 (WORKS_ON)

6.1 ALGORTIHM FOR SELECT

Many search methods can be used for simple selection: S1 through S6

S1: Linear Search (brute force) –full scan in Oracle‟s terminology

-Retrieves every record in the file, and test whether its attribute values satisfy the selection condition: an expensive approach.

S2: Binary Search

– If the selection condition involves an equality comparison on a key attribute on which the file is ordered.

σSSN=_1234567‘ (EMPLOYEE), SSN is the ordering attribute.

S3: Using a Primary Index (hash key)

An equality comparison on a key attribute with a primary index (or hash key).

This condition retrieves a single record (at most).

S4: Using a primary index to retrieve multiple records

Comparison condition is >, >=, <, or <= on a key field with a primary index

σDNUMBER >5(DEPARTMENT)

Use the index to find the record satisfying the corresponding equality condition (DNUMBER=5), then retrieve all subsequent records in the (ordered) file.

For the condition (DNUMBER <5), retrieve all the preceding records.

Method used for range queries too (i.e. queries to retrieve records in certain range)

S5: Using a clustering index to retrieve multiple records

If the selection condition involves an equality comparison on a non-key attribute with a clustering index.

σDNO=5(EMPLOYEE) – Use the index to retrieve all the records satisfying the condition

S6: Using a secondary (B+-tree) index on an equality comparison

The method can be used to retrieve a single record if the indexing field is a key or to retrieve multiple records if the indexing field is not a key.

This can also be used for comparisons involving >, >=, <, or <=.

Method used for range queries too.

Many search methods can be used for complex selection which involve a Conjunctive Condition: S7 through as S9.

Conjunctive condition: several simple conditions connected with the AND logical connective.

(OP4): s DNO=5 AND SALARY>30000 AND SEX = ‗F‘ (EMPLOYEE).

S7: Conjunctive selection using an individual index.

If an attribute involved in any single simple condition in the conjunctive condition has an access path that permits the use of one of the Methods S2 to S6, use that condition to retrieve the records.

Then check whether each retrieved record satisfies the remaining simple conditions in the conjunctive condition

S8: Conjunctive selection using a composite index:

If two or more attributes are involved in equality conditions in the conjunctive condition and a composite index (or hash structure) exists on the combined fields.

Example: If an index has been created on the composite key (ESSN, PNO) of the WORKS_ON file, we can use the index directly.

– (OP5): σESSN=„123456789" AND PNO=10 (WORKS_ON).

S9: Conjunctive selection by intersection of record pointers

If the secondary indexes are available on more than one of the fields involved in simple conditions in the conjunctive condition, and if the indexes include record pointers (rather than block pointers), then each index can be used to retrieve the set of record pointers that satisfy the individual condition.

- The intersection of these sets of record pointers gives the record pointers that satisfy the conjunctive condition.
- If only some of the conditions have secondary indexes, each retrieval record is further tested to determine whether it satisfies the remaining conditions.

6.2 ALGORITHMS FOR IMPLEMENTING JOIN OPERATION

Join: time-consuming operation. We will consider only natural join operation

- Two-way join: join on two files.
- Multiway join: involving more than two files.

The following examples of two-way JOIN operation (RΘ A=BS) will be used:

- OP6: EMPLOYEE Θ DNO=DNUMBER DEPARTMENT
- OP7: DEPARTMENT Θ MGRSSN=SSN EMPLOYEE

J1: Nested-loop join (brute force)

For each record t in R (outer loop), retrieve every record's from S (inner loop) and test whether the two records satisfy the join condition t[A] = s[B].

J2: Single-loop join (using an access structure to retrieve the matching records)

If an index (or hash key) exists for one of the two join attributes (e.g B of S), retrieve each record t in R, one at a time (single loop), and then use the access structure to retrieve directly all matching records s from S that satisfy s[B] = t[A]

J3: Sort-merge join:

- If the records of R and S are physically sorted (ordered) by value of the join attributes A and B, respectively, we can implement the join in the most efficient way.
- Both files are scanned concurrently in order of the join attributes, matching the records that have the same values for A and B.
- If the files are not sorted, they may be sorted first by using external sorting.
- Pairs of file blocks are copied into memory buffers in order and records of each file are scanned only once each for matching with the other file if A & B are key attributes.
- he method is slightly modified in case where A and B are not key attributes.

J4: Hash-join

The records of files R and S are both hashed to the same hash file using the same hashing function on the join attributes A of R and B of S as hash keys.

Partitioning Phase

First, a single pass through the file with fewer records (say, R) hashes its records to the hash file buckets.

Assumption: The smaller file fits entirely into memory buckets after the first phase.

(If the above assumption is not satisfied, the method is a more complex one and number of variations have been proposed to improve efficiency: partition has join and hybrid hash join.)

Probing Phase

A single pass through the other file (S) then hashes each of its records to probe appropriate bucket, and that record is combined with all matching records from R in that bucket.