1. Explain the Algorithms for implementing Join methods? 10M

Methods for Implementing Joins.

■ J1—Nested-loop join (or nested-block join). This is the default (brute force) algorithm, as it does not require any special access paths on either file in the join. 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].9

■ 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— say, attribute B of file S—retrieve each record t in R (loop over file R), and then use the access structure (such as an index or a hash key) 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 possible. 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 (see Section 19.2). In this method, pairs of file blocks are copied into memory buffers in order and the records of each file are scanned only once each for matching with the other file—unless both A and B are non key attributes, in which case the method needs to be modified slightly. A sketch of the sort merge join algorithm is given in Figure 19.3(a). We use R(i) to refer to the ith record in file R. A variation of the sort-merge join can be used when secondary indexes exist on both join attributes. The indexes provide the ability to access (scan) the records in order of the join attributes, but the records themselves are physically scattered all over the file blocks, so this method may be quite inefficient, as every record access may involve accessing a different disk block.

■ J4—Partition-hash join. The records of files R and S are partitioned into smaller files. The partitioning of each file is done using the same hashing function h on the join attribute A of R (for partitioning file R) and B of S (for partitioning file S). First, a single pass through the file with fewer records (say, R) hashes its records to the various partitions of R; this is called the partitioning phase, since the records of R are partitioned into the hash buck ets. In the simplest case, we assume that the smaller file can fit entirely in main memory after it is partitioned, so that the partitioned subfiles of R are all kept in main memory. The collection of records with the same value of h(A) are placed in the same partition, which is a hash bucket in a hash table in main memory. In the second phase, called the probing phase, a single pass through the other file (S) then hashes each of its records using the same hash function h(B) to probe the appropriate bucket, and that record is combined with all matching records from R in that bucket. This simplified description of partition-hash join assumes that the smaller of the two files fits entirely into memory buckets after the first phase. We will discuss the general case of partition-hash join that does not require this assumption below. In practice, techniques J1 to J4 are implemented by accessing whole disk blocks of a file, rather than individual records. Depending on the available number of buffers in memory, the number of blocks read in from the file can be adjusted.

2. a) Explain the methods for implementing Selection Operations? 4M

Implementing the SELECT Operation

There are many algorithms for executing a SELECT operation, which is basically a search operation to locate the records in a disk file that satisfy a certain condition. Some of the search algorithms depend on the file having specific access paths, and they may apply only to certain types of selection conditions. We discuss some of the algorithms for implementing SELECT in this section. We will use the following operations, specified on the relational database in Figure 3.5, to illustrate our dis cussion:

OP1: σSsn = '123456789' (EMPLOYEE)

OP2: σDnumber > 5 (DEPARTMENT)

OP3: σDno = 5 (EMPLOYEE)

OP4: σDno = 5 AND Salary > 30000 AND Sex = 'F'(EMPLOYEE)

OP5: σEssn='123456789' AND Pno =10(WORKS_ON)

Search Methods for Simple Selection.

A number of search algorithms are pos sible for selecting records from a file. These are also known as file scans, because they scan the records of a file to search for and retrieve records that satisfy a selec tion condition.4 If the search algorithm involves the use of an index, the index search is called an index scan. The following search methods (S1 through S6) are examples of some of the search algorithms that can be used to implement a select operation:

■ S1—Linear search (brute force algorithm). Retrieve every record in the file, and test whether its attribute values satisfy the selection condition. Since the records are grouped into disk blocks, each disk block is read into a main memory buffer, and then a search through the records within the disk block is conducted in main memory

■ S2—Binary search. If the selection condition involves an equality compari son on a key attribute on which the file is ordered, binary search—which is more efficient than linear search—can be used. An example is OP1 if Ssn is the ordering attribute for the EMPLOYEE file.5

■ S3a—Using a primary index. If the selection condition involves an equality comparison on a key attribute with a primary index—for example, Ssn = '123456789' in OP1—use the primary index to retrieve the record. Note that this condition retrieves a single record (at most).

■ S3b—Using a hash key. If the selection condition involves an equality com parison on a key attribute with a hash key—for example, Ssn = '123456789' in OP1—use the hash key to retrieve the record. Note that this condition retrieves a single record (at most).

■ S4—Using a primary index to retrieve multiple records. If the comparison condition is >, >=, <= on a key field with a primary index—for exam ple, Dnumber > 5 in OP2—use the index to find the record satisfying the cor responding equality condition (Dnumber = 5), then retrieve all subsequent records in the (ordered) file. For the condition Dnumber < 5, retrieve all the preceding records.

■ S5—Using a clustering index to retrieve multiple records. If the selection condition involves an equality comparison on a nonkey attribute with a clustering index—for example, Dno = 5 in OP3—use the index to retrieve all the records satisfying the condition.

■ S6—Using a secondary (B+-tree) index on an equality comparison. This search method can be used to retrieve a single record if the indexing field is a key (has unique values) or to retrieve multiple records if the indexing field is not a key. This can also be used for comparisons involving >, >=, <=.

■ S7—Conjunctive selection using an individual index. If an attribute involved in any single simple condition in the conjunctive select condition has an access path that permits the use of one of the

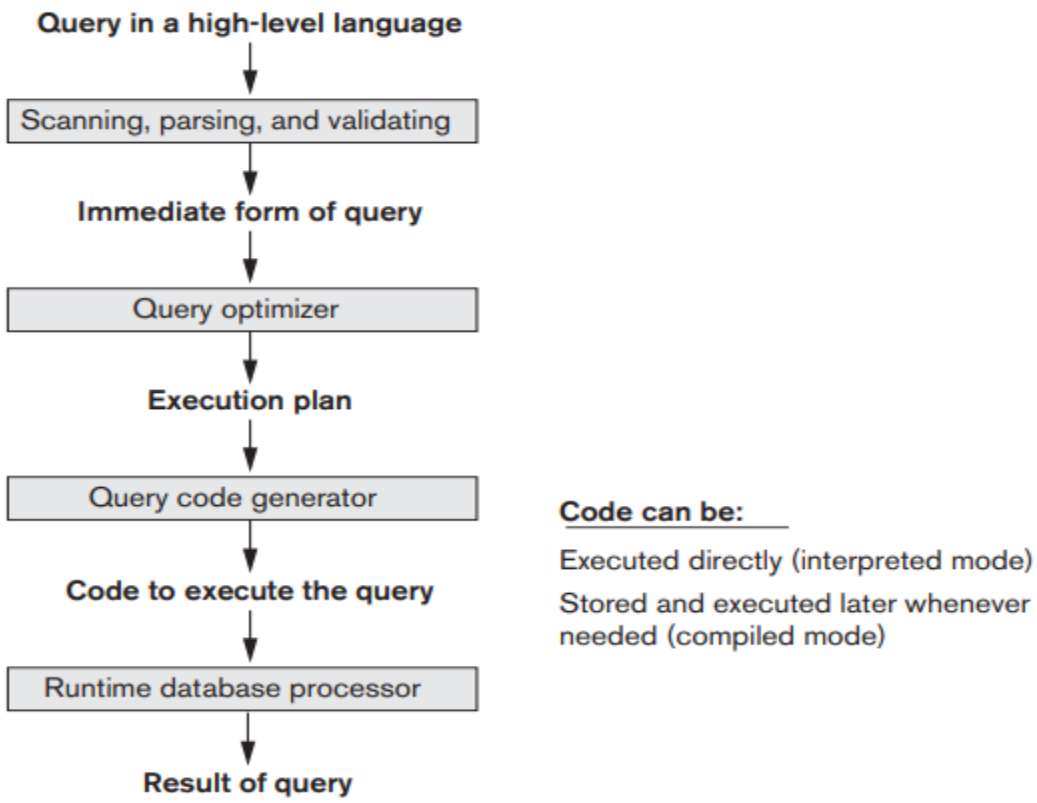
methods S2 to S6, use that condition to retrieve the records and then check whether each retrieved record satisfies the remaining simple conditions in the conjunctive select condition.

■ S8—Conjunctive selection using a composite index. If two or more attrib utes are involved in equality conditions in the conjunctive select condition and a composite index (or hash structure) exists on the combined fields— for example, if an index has been created on the composite key (Essn, Pno) of the WORKS_ON file for OP5—we can use the index directly.

■ S9—Conjunctive selection by intersection of record pointers.6 If second ary indexes (or other access paths) are available on more than one of the fields involved in simple conditions in the conjunctive select 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 indi vidual condition. The intersection of these sets of record pointers gives the record pointers that satisfy the conjunctive select condition, which are then used to retrieve those records directly. If only some of the conditions have secondary indexes, each retrieved record is further tested to determine whether it satisfies the remaining conditions.7 In general, method S9 assumes that each of the indexes is on a nonkey field of the file, because if one of the conditions is an equality condition on a key field, only one record will satisfy the whole condition

b) Explain the steps involved in the query processing with a neat diagram? 6M

Algorithms for Query Processing and Optimization



vel

. A query expressed in a high-level query language such as SQL must first be scanned, parsed, and validated.

1 The scanner identifies the query tokens—such as SQL keywords, attribute names, and relation names—that appear in the text of the query, whereas the parser checks the query syntax to determine whether it is formulated according to the syntax rules (rules of grammar) of the query language. The query must also be validated by checking that all attribute and relation names are valid and semantically meaningful names in the schema of the particular database being queried. An internal representation of the query is then created, usually as a tree data structure called a query tree. It is also possible to rep resent the query using a graph data structure called a query graph. The DBMS must then devise an execution strategy or query plan for retrieving the results of the query from the database files. A query typically has many possible execution strate gies, and the process of choosing a suitable one for processing a query is known as query optimization The query optimizer module has the task of producing a good execution plan, and the code generator generates the code to execute that plan. The runtime database processor has the task of running (executing) the query code, whether in compiled or inter preted mode, to produce the query result. If a runtime error results, an error mes sage is generated by the runtime database processo

3. Explain the algorithm for implementing 5M+5M
a) External sorting

External sorting refers to sorting algorithms that are suitable for large files of records stored on disk that do not fit entirely in main memory, such as most data base files.3 The typical external sorting algorithm uses a sort-merge strategy, which starts by sorting small subfiles—called runs—of the main file and then merges the sorted runs, creating larger sorted subfiles that are merged in turn.

In the sorting phase, runs (portions or pieces) of the file that can fit in the available buffer space are read into main memory, sorted using an internal sorting algorithm, and written back to disk as temporary sorted subfiles (or runs). The size of each run and the number of initial runs (nR) are dictated by the number of file blocks (b) and the available buffer space (nB).

In the merging phase, the sorted runs are merged during one or more merge passes. Each merge pass can have one or more merge steps. The degree of merging (dM) is the number of sorted subfiles that can be merged in each merge step. During each merge step, one buffer block is needed to hold one disk block from each of the sorted subfiles being merged, and one additional buffer is needed for containing one disk block of the merge result, which will produce a larger sorted file that is the result of merging several smaller sorted subfiles. Hence, dM is the smaller of (nB − 1) and nR, and the number of merge passes is $\lceil (\log_{dM}(nR)) \rceil$.

Set
i ← 1;
j ← b; {size of the file in blocks}
k ← nB; {size of buffer in blocks}
m ← $\lceil (j/k) \rceil$;
{Sorting Phase}
while (i ≤ m)
do {
read next k blocks of the file into the buffer or if there are less than k blocks remaining, then read in the remaining blocks;
sort the records in the buffer and write as a temporary subfile;
i ← i + 1;
}
{Merging Phase: merge subfiles until only 1 remains}

```
set     i ← 1;
        p ← ⌈logk–1m⌉     {p is the number of passes for the merging phase}
      j ← m;
while (i ≤ p)
 do {
n ← 1;
q ← ( j/(k–1)⌉ ;   {number of subfiles to write in this pass}
while (n ≤ q)
do {
read next k–1 subfiles or remaining subfiles (from previous pass)
one block at a time;
merge and write as new subfile one block at a time;
n ← n + 1;
 }
 j ← q;
i ← i + 1;
}
```

b) Outer join

In Section 6.4, the outer join operation was discussed, with its three variations: left outer join, right outer join, and full outer join. We also discussed in Chapter 5 how these operations can be specified in SQL. The following is an example of a left outer join operation in SQL:

SELECT Lname, Fname, Dname
FROM (EMPLOYEE LEFT OUTER JOIN DEPARTMENT ON Dno=Dnumber);

The result of this query is a table of employee names and their associated depart ments. It is similar to a regular (inner) join result, with the exception that if an EMPLOYEE tuple (a tuple in the left relation) does not have an associated department, the employee's name will still appear in the resulting table, but the department name would be NULL for such tuples in the query result.

Theoretically, outer join can also be computed by executing a combination of rela tional algebra operators. For example, the left outer join operation shown above is equivalent to the following sequence of relational operations:

1. Compute the (inner) JOIN of the EMPLOYEE and DEPARTMENT tables. TEMP1 ← πLname, Fname, Dname (EMPLOYEE Dno=Dnumber DEPARTMENT)
2. Find the EMPLOYEE tuples that do not appear in the (inner) JOIN result. TEMP2 ← πLname, Fname (EMPLOYEE) – πLname, Fname (TEMP1)
3. Pad each tuple in TEMP2 with a NULL Dname field.
TEMP2 ← TEMP2 × NULL
4. Apply the UNION operation to TEMP1, TEMP2 to produce the LEFT OUTER JOIN result. RESULT ← TEMP1 ∪ TEMP2

4. Explain the notation for Query trees and Query Graphs and also explain the heuristic optimization of query trees? 10M

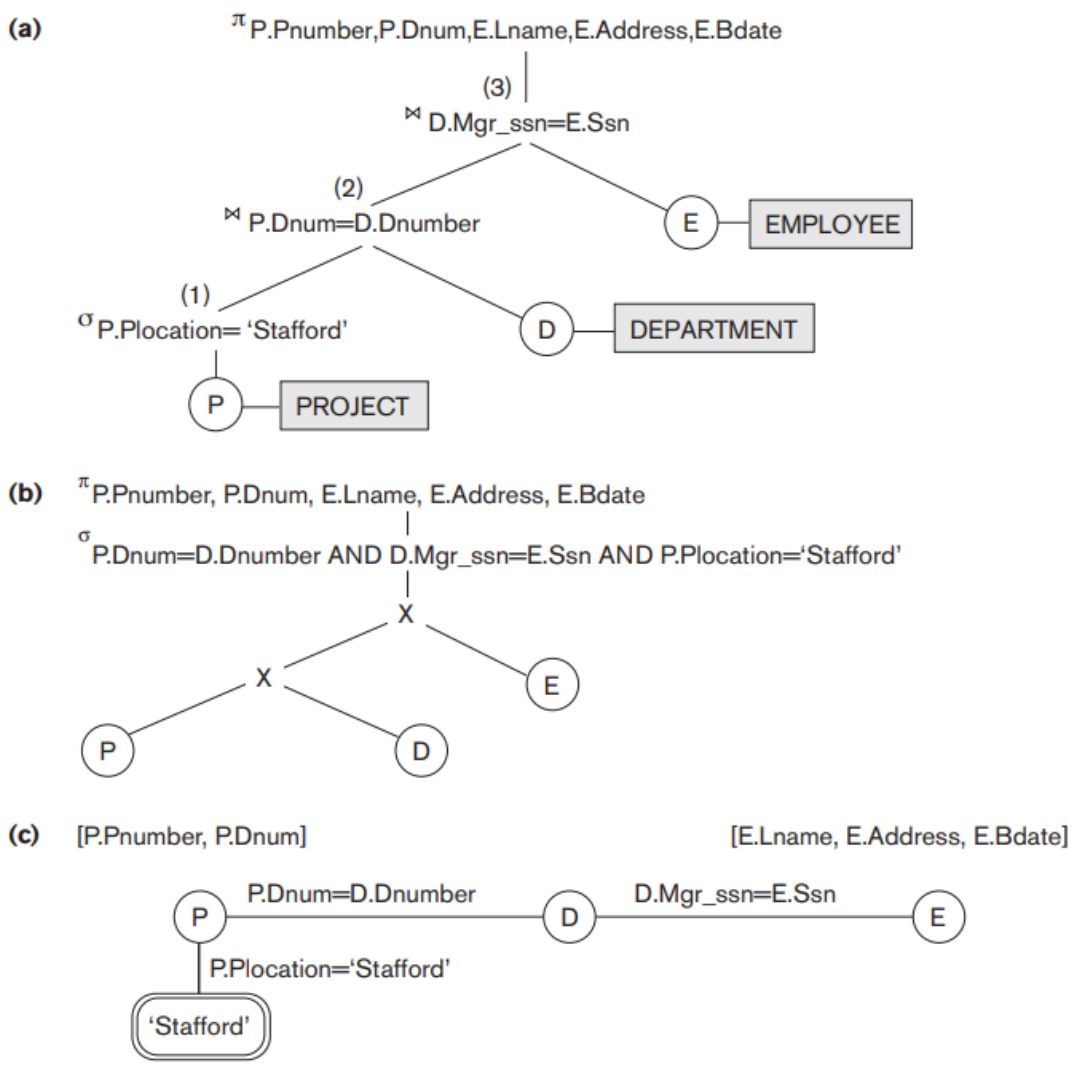
Notation for Query Trees and Query Graphs

A query tree is a tree data structure that corresponds to a relational algebra expres sion. It represents the input relations of the query as leaf nodes of the tree, and rep resents the relational algebra operations as internal nodes. An execution of the query tree consists of executing an internal node operation whenever its operands are available and then replacing that internal node by the relation that results from executing the operation. The order of execution of operations starts at the leaf

nodes, which represents the input database relations for the query, and ends at the root node, which represents the final operation of the query. The execution terminates when the root node operation is executed and produces the result relation for the query

```
Q2: SELECT  P.Pnumber, P.Dnum, E.Lname, E.Address, E.Bdate
    FROM    PROJECT AS P, DEPARTMENT AS D, EMPLOYEE AS E
    WHERE   P.Dnum=D.Dnumber AND D.Mgr_ssn=E.Ssn AND
            P.Plocation= 'Stafford';
```

Chapter 19 Algorithms for Query Processing and Optimization



Heuristic Optimization of Query Tree

The query parser will typically generate a standard initial query tree to correspond to an SQL query, without doing any optimization. For example, for a SELECT PROJECT-JOIN query, such as Q2, the initial tree is shown in Figure 19.4(b). The CARTESIAN PRODUCT of the relations specified in the FROM clause is first applied; then the selection and join conditions of the WHERE clause are applied, followed by the projection on the SELECT clause attributes. Such a canonical query tree repre sents a relational algebra expression that is very inefficient if executed directly, because of the CARTESIAN PRODUCT (×) operations. For example, if the PROJECT, DEPARTMENT, and EMPLOYEE relations had record sizes of 100, 50, and 150 bytes and contained 100, 20, and 5,000 tuples, respectively, the result of the CARTESIAN PRODUCT would contain 10 million tuples of record size 300 bytes each. However, the initial query tree in Figure 19.4(b) is in a simple standard form that can be eas ily created from the SQL query. It will never be executed. The heuristic query opti mizer will transform this initial query tree into an equivalent final query tree that is efficient to execute

5. Write the Algorithms for implementing the Union, Intersection and Set difference? 10M