LECTURE 3

IMPLEMENTING THE JOIN OPERATION (1/12)

- Time consuming operation
- Common is EQUIJOIN and NATURAL JOIN
- Two-way join :- join on two files
- Multi-way joins :- joins involving more than two files

- Consider the join operation

$$R \bowtie_{A=B} S$$

Where A and B are attributes of R and S resp.

IMPLEMENTING THE JOIN OPERATION (2/12)

- Methods for implementing Joins
  - **(J1) Nested loop join (brute force)**

    For each record t in R (outer loop), retrieve every records s from S (inner loop) and test whether two records satisfy join condition

  - **(J2) Single loop join**

    If index exists for one of the two join attributes, say B of S, retrieve each record t in R one at a time and use access structure to retrieve all matching records directly s from S

IMPLEMENTING THE JOIN OPERATION (3/12)

- **(J3) Sort-merge join**
  - If records of R and S are physically sorted by value of join attributes A and B respectively, we can implement join directly
  - Both files are scanned concurrently in the order of the join attributes matching records that have same value for A and B.
  - If files are not sorted, use external sorting
  - Pairs of file blocks are copied into buffers in order and records of each file are scanned only once each for matching with the other file

IMPLEMENTING THE JOIN OPERATION (4/12)

- **(J4) Hash join**
  - Records of R and S are both hashed to same hash file using same hashing function on join attributes A of R and B of S
  - **First phase** - Single pass through the file with fewer records (say R) hashes its records to the hash file buckets. This is called ***partitioning phase*** since records of R are partitioned into hash buckets

IMPLEMENTING THE JOIN OPERATION (5/12)

- **Second phase**, called ***probing phase***, single pass through the other file (S) then hashes each of its records to probe the appropriate bucket. That record is combined with all matching records from R in that bucket

- It **assumes that smaller of the two files fits entirely in memory buckets** after the first phase.

IMPLEMENTING THE JOIN OPERATION (6/12)

- **Partition hash join**
  - For hash join, only single pass is made through each file, whether or not files are ordered
  - If parts of hash file has to be stored on disk, method becomes complex
  - Each file is first partitioned into M partitions using partitioning hash function on join attributes
  - Then each pair of partition is joined

IMPLEMENTING THE JOIN OPERATION (7/12)

- **Partitioning phase**
  - R is partitioned into M partitions R_1, R_2... R_M, and S into S_1, S_2, ...S_M
  - Each of the files R and S are partitioned separately
- Property : ***Records in R_i only need to be joined with records in S_i***
- For each of the partitions, a single buffer whose size is one disk block is allocated to store records that hash to this partition
- Whenever buffer gets filled, its contents are appended to disk subfile that stores this partition

IMPLEMENTING THE JOIN OPERATION (8/12)

- Partitioning phase has two iterations
- First iteration
  - First file R is partitioned into subfiles R_1, R_2... R_M where all records that hashed to same buffer are in the same partition
- Second iteration
  - S is similarly partitioned
- In probing (joining) phase, M iterations are needed

IMPLEMENTING THE JOIN OPERATION (9/12)

- During iteration i, two partitions R_i and S_i are joined
- If nested loop is used during iteration i, the records from the smaller of the two partitions R_i are copied into buffers
- Then all blocks from other partition S_i are read one at a time and each record is used to probe partition R_i for matching record(s)
- Any matching records are joined and written into result file

IMPLEMENTING THE JOIN OPERATION (10/12)

- **Hybrid hash-join**
  - Variation of partition hash join
  - **The joining phase for one of the partitions is included in partitioning phase**
  - **First pass of partitioning phase**
    - When partitioning the smaller of the two files, also divides the buffer space among M partitions such that all the blocks of **first partition completely reside in main memory**
    - Remainder of the partition is written to disk

IMPLEMENTING THE JOIN OPERATION (11/12)

- **Second pass of partitioning phase**
  - Records of second file being joined are being partitioned
  - If record hashes to first partition, it is joined with matching record in first file and joined records are written to result buffer
  - But if record hashes to other partition other than first, it is partitioned normally
- Hence at the end of the second pass of partitioning phase, all records that hashed to the first partition has been joined

IMPLEMENTING THE JOIN OPERATION (12/12)

- Now only M-1 pairs of partitions on disk is there
- During probing phase, M-1 iterations are needed instead of M
- **Goal is to join as many records during partitioning phase to save cost of storing those records back to disk and rereading them second time during joining phase**

IMPLEMENTATION OF SET OPERATIONS (1/5)

- Expensive to implement
- CARTESIAN PRODUCT is most expensive since its result includes a record for each combination of records from R and S
- UNION, INTERSECTION and SET DIFFERENCE apply to only union-compatible relations
- Use sort-merge strategy to implement them
- Two relations are sorted on same attributes
- After sorting, single scan through each relation is sufficient to produce result

IMPLEMENTATION OF SET OPERATIONS (2/5)

- **UNION operation**
  - Scan and merge both sorted files concurrently
  - Whenever same tuple exists, only one is kept in merged result
- **INTERSECTION operation**
  - Scan and merge both sorted files concurrently
  - In merged result, keep tuples that appear in both relations

IMPLEMENTATION OF SET OPERATIONS (3/5)

- **SET DIFFERENCE operation**
  - Scan and merge both sorted files concurrently
  - In merged result, keep tuples that appear only in first (left hand side) relation

IMPLEMENTATION OF SET OPERATIONS (4/5)

- Hashing can also be used to implement SET operations
- One table is partitioned and other is used to probe appropriate partition
- **UNION operation**
  - First partition records of R
  - Probe records of S but do not insert duplicate records in buckets

IMPLEMENTATION OF SET OPERATIONS (5/5)

- **INTERSECTION operation**
  - Partition records of R to the hash file
  - While hashing each record of S, probe to check if identical record from R is found in the bucket, if so add the record to the result file
- **SET DIFFERENCE operation**
  - Hash the records of R to the hash file buckets
  - While probing each record of S, if identical record is found, remove that record from the bucket

PIPELINING

- Query specified in SQL will be translated into a relational algebra expression that is a sequence of relational operations
- If we execute a single operation at a time, we must generate **temporary files** on disk to hold results of these temporary operations, creating ***excessive overhead*** and is also ***time consuming***
- To reduce number of temporary files, we combine several relational operators into a **pipeline** of operations in which **results of one operation** are **passed along to the next operation** in the pipeline