LECTURE 2

BASIC ALGORITHM FOR EXECUTING QUERY OPERATIONS (1/4)

- **External Sorting**
  - Sorting algorithms that are ***suitable for large files*** of records stored on disk that do not fit entirely in memory
  - Uses a sort-merge strategy
    1. **Sorts** small subfiles, called ***runs***, of the main file
    2. **Merges** the sorted runs creating larger sorted subfiles

BASIC ALGORITHM FOR EXECUTING QUERY OPERATIONS (2/4)

- Algorithm consists of two phases
  - Sorting phase
  - Merging phase

- *Sorting phase*
  - Runs that can fit in available buffer space are read into memory, sorted and written back to disk as temporary runs

  - Size of runs and **number of initial runs (n_R)** depends on the **number of file blocks** (b) and **available buffer space** (n_B)

BASIC ALGORITHM FOR EXECUTING QUERY OPERATIONS (3/4)

- Eg. If n_B = 5 blocks, b = 1024 blocks,

  then $n_R = b / n_B$

  n_R = 1024/5 = 205 initial runs each of size 5 block (except last run which is 4 blocks)

- After sort phase, **205 sorted runs** are stored as temporary subfiles on disk

BASIC ALGORITHM FOR EXECUTING QUERY OPERATIONS (4/4)

- *Merging phase*
  - Sorted runs are merged during one or more passes
  - **Degree of merging (d_M)** is the number of runs that can be merged together in each pass
  - d_M is the smaller value of (n_B - 1) and n_R
  - No of passes = $\log_{dM} (n_R)$

EXAMPLE – EXTERNAL SORTING

If $d_M = 4$ (i.e. number of runs that can be merged together in each pass)

Initial runs = 205

First pass = 205 / 4 = 52

Second pass = 52 / 4 = 13

Third pass = 13 / 4 = 4

Fourth pass =4 / 4 = 1

Therefore, four passes are needed

IMPLEMENTING THE SELECT OPERATION (1/4)

- Methods for **Simple Selection**
  - **(S1) Linear search (brute force)**

    Retrieve every record in the file
  - **(S2) Binary search**

    Use equality comparison on **key** attribute for ordered file
  - **(S3) Using primary index (hash key)**

    Use equality comparison on **key** attribute with primary index

IMPLEMENTING THE SELECT OPERATION (2/4)

- **(S4) Using primary index to retrieve multiple records**

  For comparison condition <, <=, >, >= on **key** field with primary index

- **(S5) Clustering index to retrieve multiple records**

  Use equality comparison on **non-key** attribute with clustering field

- **(S6) Secondary index on equality comparison**

  Retrieve single record if indexing field is a **key**, or multiple records if indexing field is **not key**

IMPLEMENTING THE SELECT OPERATION (3/4)

- Methods for **Complex selection**
  - **(S7) Conjunctive selection using one index**
    - Retrieve records
    - Check whether each retrieved record satisfies the remaining conditions in the conjunctive condition
  - **(S8) Conjunctive selection using composite index**
    - If 2 or more attributes are involved in the conjunctive condition and a composite index exists on the combined fields, we can use index directly

IMPLEMENTING THE SELECT OPERATION (4/4)

- **(S9) Conjunctive selection by intersection of record pointers :-**
  - If secondary indexes are available on more than one of the fields involved in conjunctive condition, and if indexes include record pointers, then each index can be used to retrieve the set of record pointers that satisfy individual condition
  - Intersection of these sets of record pointers gives the record pointers that satisfy the conjunctive condition, which are then used to retrieve those records directly.

IMPLEMENTATION OF PROJECT OPERATION

- Π <attribute list> R
- If <attribute list> includes a key of relation R,

Result of operation = number of tuples as R

- If <attribute list> does not include a key of relation R, **Duplicate tuples must be eliminated**
- Sort the result of operation and then eliminate consecutive duplicates