Database management system notes

File can be thought of as " logical" or a " physical" entity. File as a logical entity: a sequence of records. Records are either fixed size or variable size. A file as a physical entity: a sequence of fixed size blocks (on the disk), but not necessarily physically contiguous (the blocks could be dispersed).

Blocks in a file are either physically contiguous or not, but the following is generally simple to do (for the file system): B» Find the first block B» Find the last block B» Fold the next block B» Fold the previous block Records are stored in blocks » This gives the mapping between a " logical" file and a " physical" file Assumptions (some to simplify presentation for now) B» Flexed size records B» No record spans more than one block B» There are, In general, several records In a block B» There is some " left space in a block as needed later (for chaining the blocks of the file) We assume that each relation is stored as a file, Each duple is a record in the file. A file Is a sequence of records, where each record Is a collection of data values (or data Items).

A file descriptor(or file header Includes Information that describes the ill, such as the field names and their data types , and the addresses of the file blocks on disk. Records are stored on disk blocks. The blocking factor bar or a file is the (average) number of file records stored in a disk block. A file can have fixed-length records or variable-length records. File records can be unsnapped or spanned Unsnapped : no record can span two blocks Spanned : a record can be stored In more than one block The physical disk blocks that are allocated to hold the records of a file can be contiguous, linked, or indexed. In a file of fixed-length records, all records have the same format. Usually, unsnapped blocking is used with such files.

Files of variable- length records require additional information to be stored in each record, such as separator characters and field types. Usually spanned blocking is used with such files. Typical file operations Include: OPEN : Readies the file for access, and associates a pointer that will refer too current file record at each point in time. MIND : Searches for the first file record that satisfies a certain condition, and makes it the current file record. FINANCED : Searches for the next file record (from the current record) that satisfies a certain onto a program variable. INSERT Inserts a new record into the file & makes it the current file record.

DELETE: Removes the current file record from the file, usually by marking he record to indicate that it is no longer valid. MODIFY: Changes the values of some fields of the current file record. CLOSE: Terminates access to the file. REORGANIZE: Reorganizes the file records. For example, the records marked deleted are physically removed from the file or a new organization of the file records is crated. READ_ORDERED: Read the file blocks in order of a specific field of the file. Unordered Files Also called a heap or a pile file. New records are inserted at the end of the file. A linear search through the file records is necessary to search for a record.

This requires reading and searching half the file blocks on the average, and is hence quite expensive. Record insertion is quite efficient. Reading the records in order of a particular field requires sorting the file records. Ordered Files Also called a sequential file. File records are kept sorted by the values of an ordering field. Insertion is expensive: records must be inserted in the correct order. B» It is common to keep a separate unordered overflow (or transaction) file for new records o improve insertion efficiency; this is periodically merged with the main ordered file. A binary search can be used to search for a record on its ordering field value.

This requires reading and searching logo of the file blocks on the average, an improvement over linear search. Reading the records in order of the ordering field is quite efficient. HASHING TECHNIQUES INDEXING Indexing mechanisms used to speed up access to desired data. E. G. , author catalog in library. Search Key- attribute to set of attributes used to look up records in a file. An index file consists of records (called index entries) of the form. Search-Key Pointer Index files are typically much smaller than the original file . Two basic kinds of indices: Ordered indices: search keys are stored in sorted order Hash indices: search keys are distributed uniformly across " buckets" using a " hash function". Access types supported efficiently. E. G. Records with a specified value in the attribute or records with an attribute value falling in a specified range of values (e. G. 10000 < salary < 40000) Access time , Insertion time , Deletion time, Space overhead Ordered Indices In an ordered index, index entries are stored sorted on the search key value. E. g. , author catalog in library. Primary index: in a sequentially ordered file, the index whose search key specifies the sequential order of the file. Also called clustering index. The search key of a primary index is usually but not necessarily the primary the sequential order of the file. Also called non-clustering index. Index-sequential file: ordered sequential file with a primary index.

Dense Index Files Index record appears for every search-key value in the file. Sparse Index Files contains index records for only some search-key values applicable when records are sequentially ordered on search-key. To locate a record with search-key value K we: Find index record with largest search-key value < K, Search file sequentially starting at the record to which the index record points. Compared to dense indices: Less space and less maintenance overhead for insertions and deletions. , Generally slower than dense index for locating records. Good tradeoff: sparse index with an index entry for every block in file, corresponding to least search-key value in the block.

MULTI-LEVEL INDEXES If primary index does not fit in memory, access becomes expensive. Solution: treat primary index kept on disk as a sequential file and construct a sparse index on it. Utter index - a sparse index of primary index inner index - the primary index file If even outer index is too large to fit in main memory, yet another level of index can be created, and so on. Indices at all levels must be updated on insertion or deletion from the file. Index Update: Record Deletion If deleted record was the only record in the file with its particular search-key value, the search-key is deleted from the index also. Single-level index deletion: Dense indices - deletion of search-key: similar to file record deletion.

Sparse indices - if deleted key value exists in the index, the value is replaced by the next search-key alee in the file (in search-key order). If the next search-key value already has an index entry, the entry is deleted instead of being replaced. Single-level index insertion: Perform a lookup using the key value from inserted record. Dense indices - if the search-key value does not appear in the index, insert it. Sparse indices - if index stores an entry for each block of the file, no change needs to be made to the index unless a new block is created. If a new block is created, the first search-key value appearing in the new block is inserted into the index.

Multilevel insertion (as well as deletion) algorithms are simple extensions of the single-level algorithms Secondary Indices Example Index record points to a bucket that contains pointers to all the actual records with that particular search-key value. Secondary indices have to be dense. Indices offer substantial benefits when searching for records. BUT: Updating indices imposes must be updated, Sequential scan using primary index is efficient, but a sequential scan using a secondary index is expensive.