

Storage Structures

Unit 4.3

The Physical Store



Storage Capacity	Medium	Transfer Rate	Seek Time
Main Memory	800 MB/s	500 MB	Instant
Hard Drive	10 MB/s	120 GB	10 ms
CD-ROM Drive	5 MB/s	0.6 GB	100 ms
Flopp Drive	2 MB/s	2.88 MB	300 ms
Tape Drive	1 MB/s	20 GB	30 s

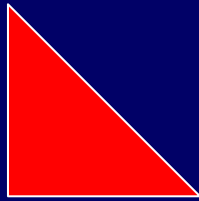
Why not all Main Memory?



The performance of main memory is the greatest of all storage methods, but it is also the most expensive per MB.

- All the other types of storage are 'persistent'. A persistent store keeps the data stored on it even when the power is switched off.
- Only main memory can be directly accessed by the programmer. Data held using other methods must be loaded into main memory before being accessed, and must be transferred back to storage from main memory in order to save the changes.
- We tend to refer to storage methods which are not main memory as 'secondary storage'.

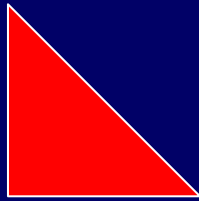
Secondary Storage - Blocks



All storage devices have a block size. Block size is the minimum amount which can be read or written to on a storage device. Main memory can have a block size of 1-8 bytes, depending on the processor being used. Secondary storage blocks are usually much bigger.

- Hard Drive disk blocks are usually 4 KBytes in size.
- For efficiency, multiple contiguous blocks can be requested.
- On average, to access a block you first have to request it, wait the seek time, and then wait the transfer time of the blocks requested.
- Remember, you cannot read or write data smaller than a single block.

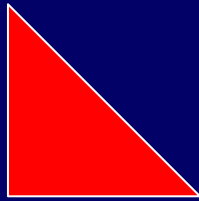
Hard Drives



The most common secondary storage medium for DBMS is the hard drive.

- Data on a hard-drive is often arranged into files by the Operating System.
- the DBMS holds the database within one or more files.
- The data is arranged within a file in blocks, and the position of a block within a file is controlled by the DBMS.
- Files are stored on the disk in blocks, but the placement of a file block on the disk is controlled by the O/S (although the DBMS may be allowed to 'hint' to the O/S concerning disk block placement strategies).
- File blocks and disk blocks are not necessarily equal in size.

DBMS Data Items

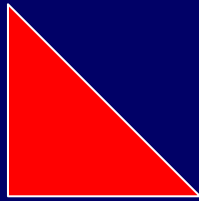


Data from the DBMS is split into records.

- a record is a logical collection of data items
- a file is a collection of records.
- one or more records may map onto a single or multiple file blocks.
- a single record may map onto multiple file blocks.

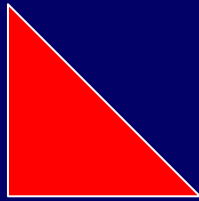
Relational	SQL	Physical Storage
Relational	Table	File
Tuple	Row	Record
Attribute	Column	Data Item/Field
Domain	Type	Data Type

File Organisations



- Serial (or unordered, or heap) - records are written to secondary storage in the order in which they are created.
- Sequential (or sorted, or ordered) - records are written to secondary storage in the sorted order of a key (one or more data items) from each record.
- Hash - A 'hash' function is applied to each record key, which returns a number used to indicate the position of the record in the file. The hash function must be used for both reading and writing.
- Indexed - the location in secondary storage of some (partial index) or all (full index) records is noted in an index.

Storage Scenario

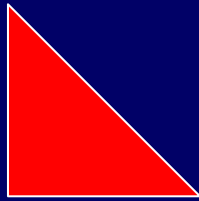


To better explain each of these file organisations we will create 4 records and place them in secondary storage. The records are created by a security guard, and records who passes his desk in the morning and at what time they pass.

The records therefore each have three data items; 'name', 'time', and 'id number'. Only four people arrive for work:

1. name='Russell' at time='0800' with id_number='004'.
2. name='Greg' at time='0810' with id_number='007'.
3. name='Jon' at time='0840' with id_number='002'.
4. name='Cumming' at time='0940' with id_number='003'.

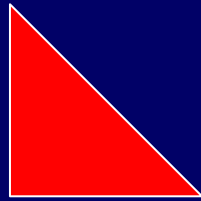
Serial Organisation



Russell 0800 004	Greg 0810 007	Jon 0840 002	Cumming 0940 003
1	2	3	4

- Writing - the data is written at the end of the previous record.
- Reading -
 - reading records in the order they were written is a cheap operation.
 - Trying to find a particular record means you have to read each record in turn until you locate it. This is expensive.
- Deleting - Deleting data in such an structure usually means marking the data as deleted (thus not actually removing it) which is cheap but wasteful or rewriting the whole file to overwrite the deleted record (space-efficient but expensive).

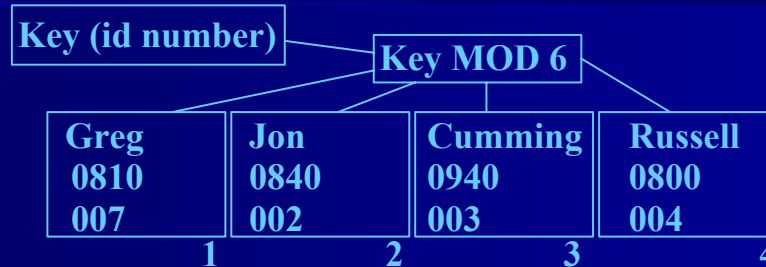
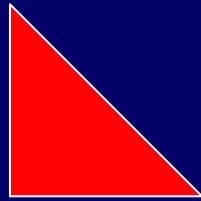
Sequential Organisation



Jon	Cumming	Russell	Greg
0840	0940	0800	0810
002	003	004	007
1	2	3	4

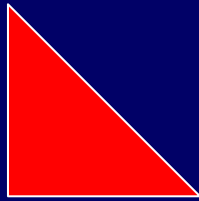
- Writing - records are in 'id number' order, thus new records may need to be inserted into the store needing a complete file copy (expensive).
- Deleting - as with serial, either leave holes or perform make file copies.
- Reading -
 - reading records in 'id number' order is cheap.
 - the ability to chose sort order makes this more useful than serial.
 - 'binary search' could be used. Goto middle of file - if record key greater than that wanted search the low half, else search the high half, until the record is found. (average accesses to find something is \log_2 no_of_records.)

Hash Organisation



- Writing - Initially the file has 6 spaces ($n \text{ MOD } 6$ can be 0-5). To write, calculate the hash and write the record in that location (cheap).
- Deleting - leave holes (wasteful) by marking the record deleted (cheap);
- Reading -
 - reading records an order is expensive.
 - finding a particular record from a key is cheap and easy.
 - If two records can result in the same hash number, then a strategy must be found to solve this problem (which will incur overheads).

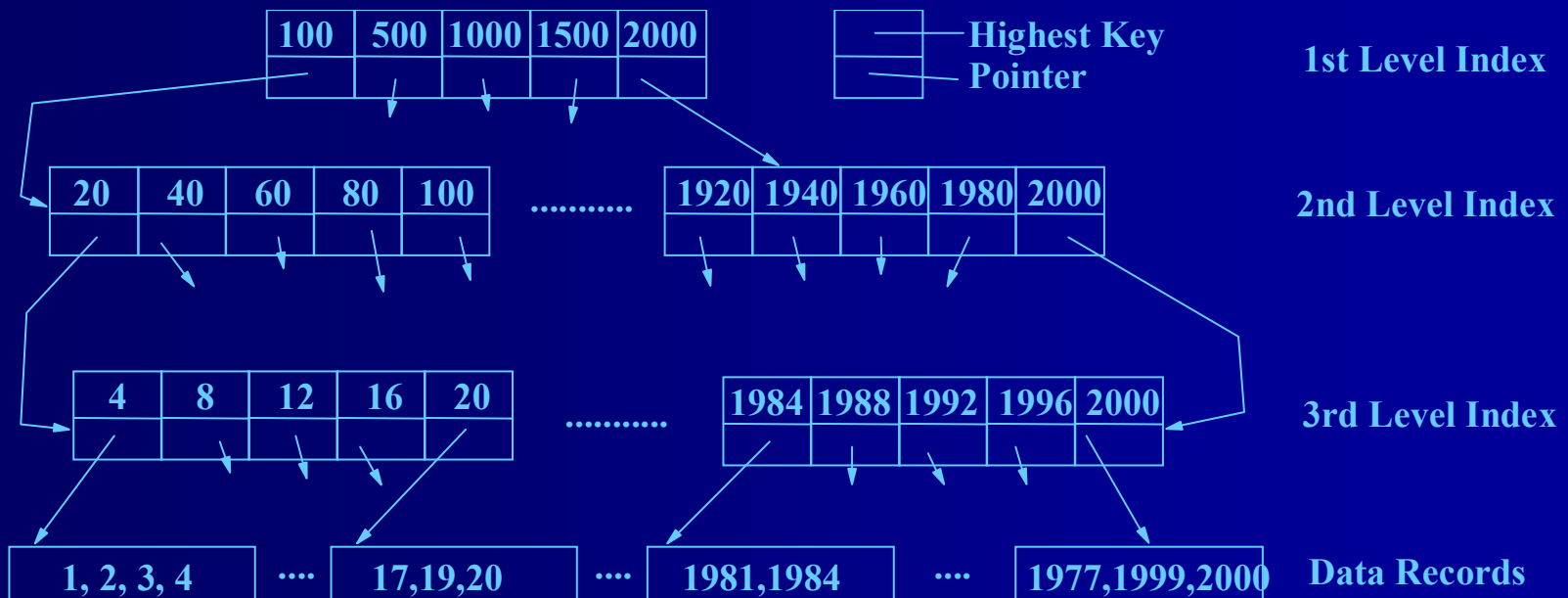
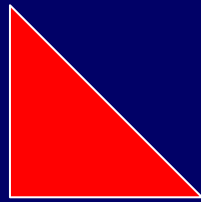
Indexed Sequential Access Method



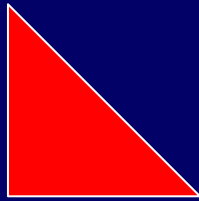
The Indexed Sequential Access Method (ISAM) is frequently used for partial indexes.

- there may be several levels of indexes, commonly 3
- each index-entry is equal to the highest key of the records or indices it points to.
- the records of the file are effectively sorted and broken down into small groups of data records.
- the indices are built when the data is first loaded as sorted records.
- the index is static, and does not change as records are inserted and deleted
- insertion and deletion adds to one of the small groups of data records. As the number in each group changes, the performance may deteriorate.

ISAM Example



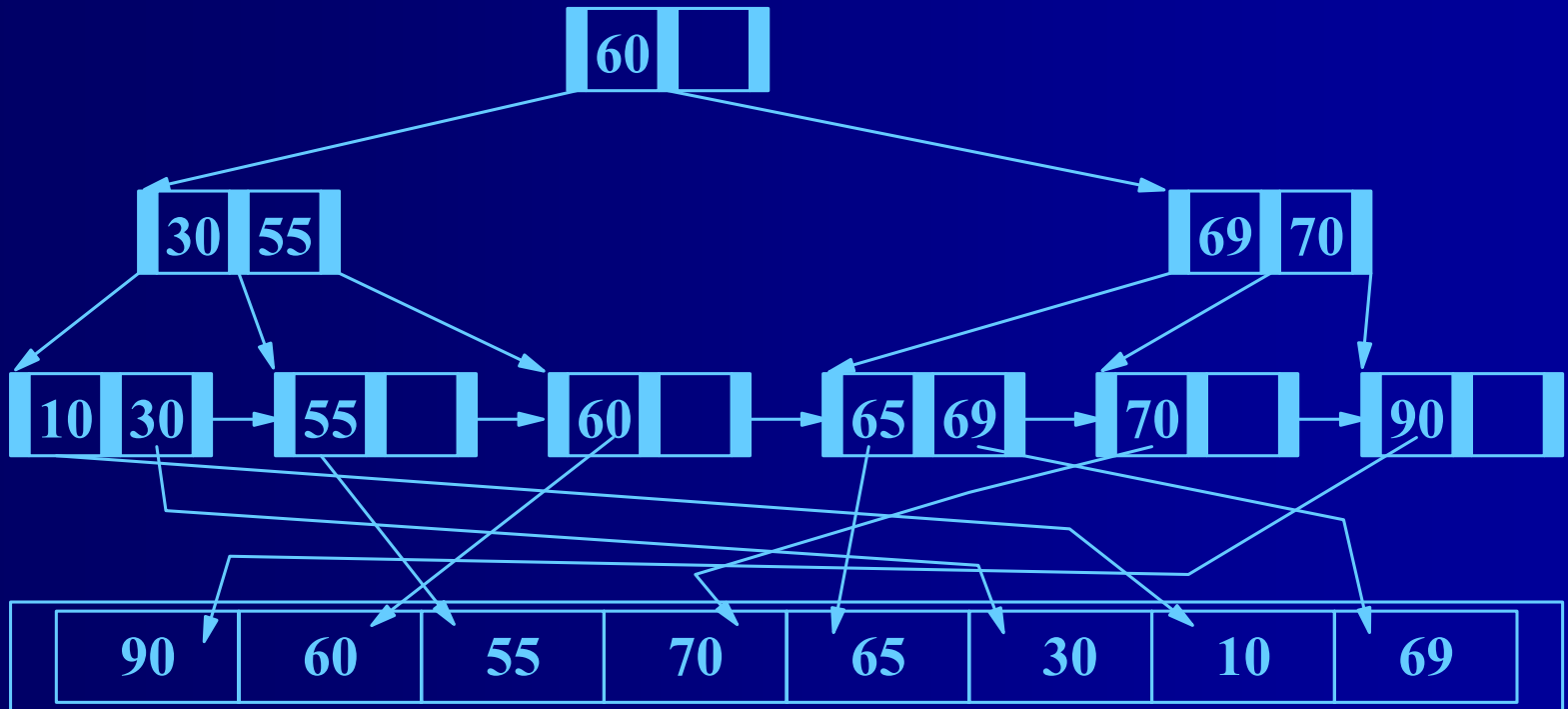
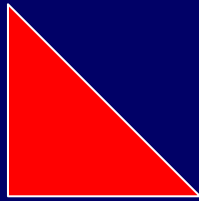
B+ Tree Index



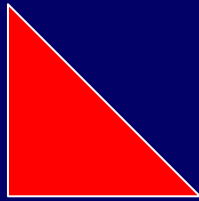
With B+ tree, a full index is maintained, allowing the ordering of the records in the file to be independent of the index. This allows multiple B+ tree indices to be kept for the same set of data records.



- the lowest level in the index has one entry for each data record.
- the index is created dynamically as data is added to the file.
- as data is added the index is expanded such that each record requires the same number of index levels to reach it (thus the tree stays 'balanced').
- the records can be accessed via an index or sequentially.

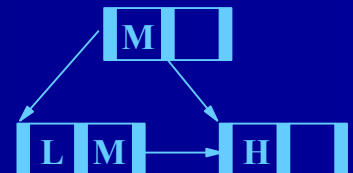
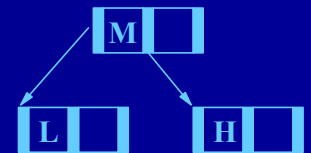
B+ Tree Example



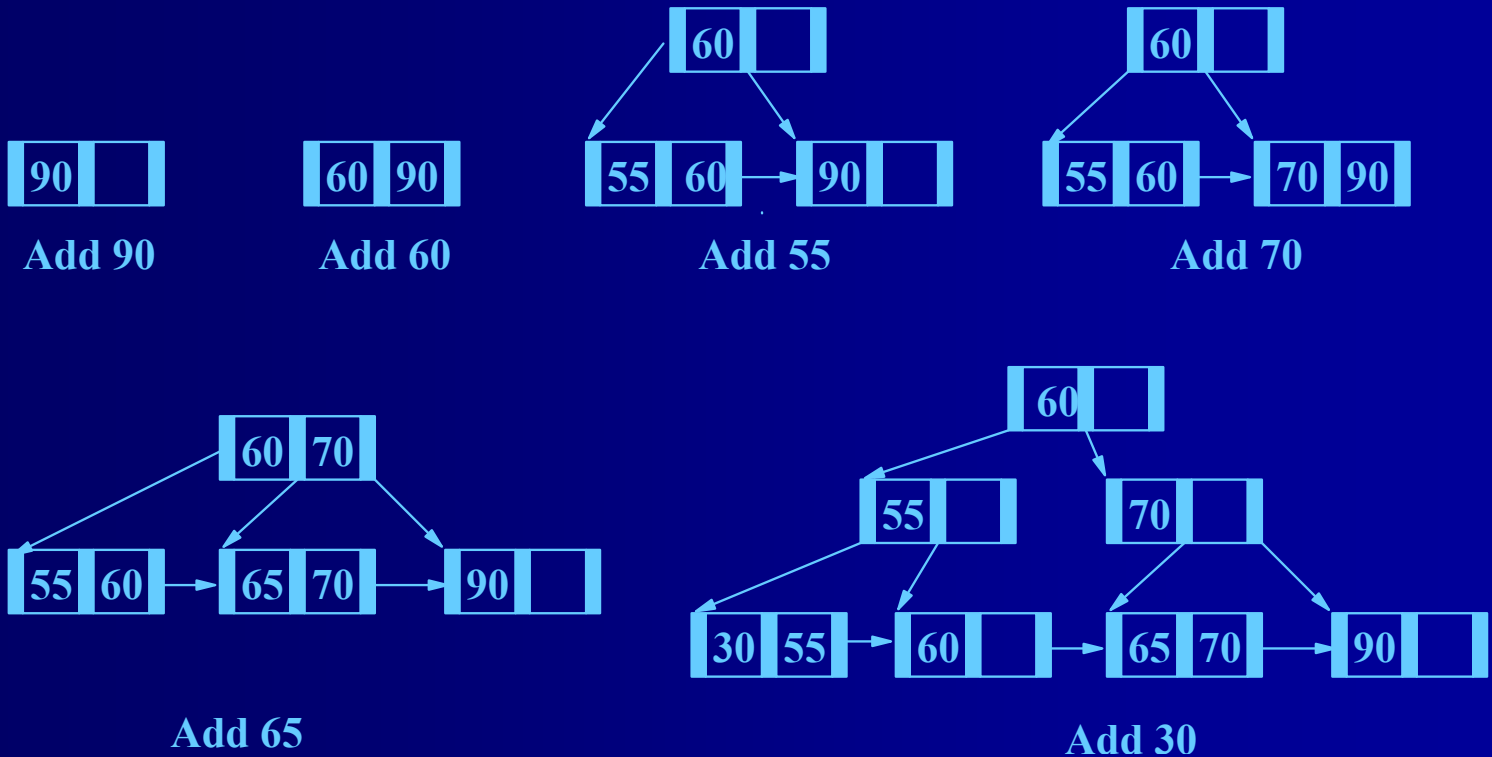
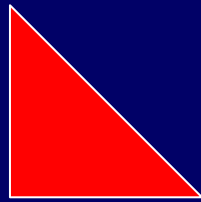
Building a B+ Tree



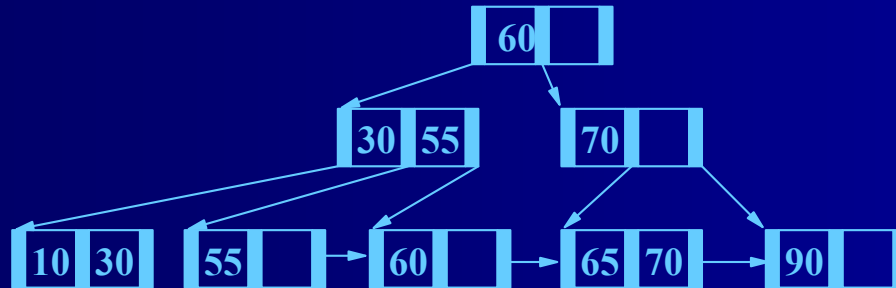
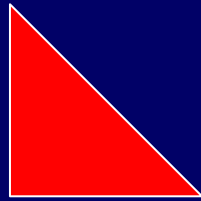
- Only nodes at the bottom of the tree point to records, and all other nodes point to other nodes. Nodes which point to records are called leaf nodes.
- If a node is empty the data is added on the left. 
- If a node has one entry, then the left takes the smallest valued key and the right takes the biggest. 
- If a node is full and is a leaf node, classify the keys L (lowest), M (middle value) and H (highest), and split the node.
- If a node is full and is not a leaf node, classify the keys L (lowest), M (middle value) and H (highest), and split the node.



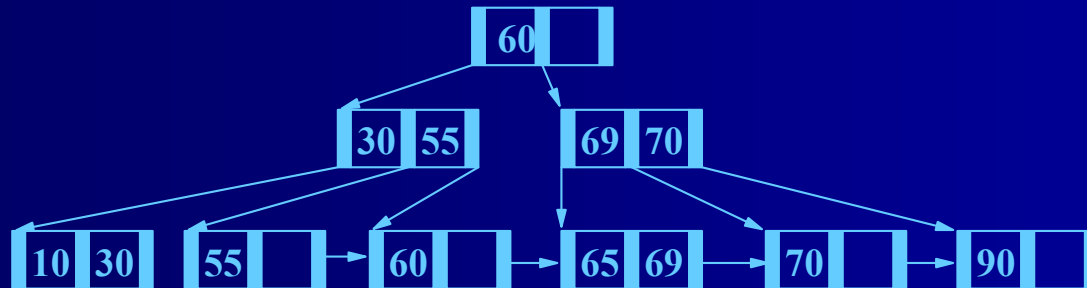
B+ Tree Build Example



B+ Tree Build Example Cont...



Add 10



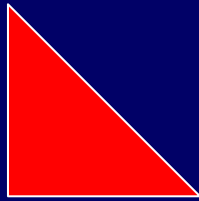
Add 69

Index Structure and Access



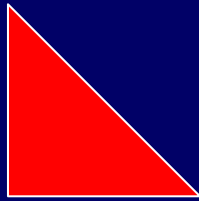
- The top level of an index is usually held in memory. It is read once from disk at the start of queries.
- Each index entry points to either another level of the index, a data record, or a block of data records.
- The top level of the index is searched to find the range within which the desired record lies.
- The appropriate part of the next level is read into memory from disc and searched.
- This continues until the required data is found.
- The use of indices reduce the amount of file which has to be searched.

Costing Index and File Access



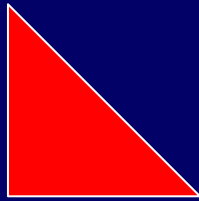
- The major cost of accessing an index is associated with reading in each of the intermediate levels of the index from a disk (milliseconds).
- Searching the index once it is in memory is comparatively inexpensive (microseconds).
- The major cost of accessing data records involves waiting for the media to recover the required blocks (milliseconds).
- Some indexes mix the index blocks with the data blocks, which means that disk accesses can be saved because the final level of the index is read into memory with the associated data records.

Use of Indexes



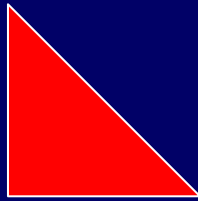
- A DBMS may use different file organisations for its own purposes.
- A DBMS user is generally given little choice of file type.
- A B+ Tree is likely to be used wherever an index is needed.
- Indexes are generated:
 - (Probably) for fields specified with 'PRIMARY KEY' or 'UNIQUE' constraints in a CREATE TABLE statement.
 - For fields specified in SQL statements such as CREATE [UNIQUE] INDEX indexname ON tablename (col [,col]...);
- Primary Indexes have unique keys.
- Secondary Indexes may have duplicates.

Use of Indexes cont...



- An index on a column which is used in an SQL 'WHERE' predicate is likely to speed up an enquiry.
 - this is particularly so when '=' is involved (equijoin)
 - no improvement will occur with 'IS [NOT] NULL' statements
 - an index is best used on a column which widely varying data.
 - indexing and column of Y/N values might slow down enquiries.
 - an index on telephone numbers might be very good but an index on area code might be a poor performer.
- Multicolumn index can be used, and the column which has the biggest range of values or is the most frequently accessed should be listed first.
- Avoid indexing small relations, frequently updated columns, or those with long strings.

Use of indexes cont...



- There may be several indexes on each table. Note that partial indexing normally supports only one index per table.
- Reading or updating a particular record should be fast.
- Inserting records should be reasonably fast. However, each index has to be updated too, so increasing the indexes makes this slower.
- Deletion may be slow.
 - particularly when indexes have to be updated.
 - deletion may be fast if records are simply flagged as 'deleted'.