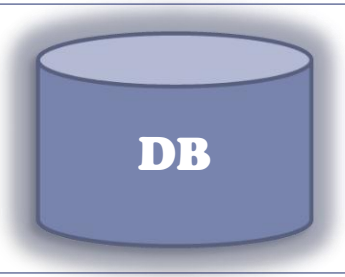


Advanced Databases

4. DBMS File Management

4. DBMS File Management

4.1 Introduction
4.2 Heap Files
4.3 Ordered Files
4.4 Hash Files
4.5 Summary

4.1 Introduction

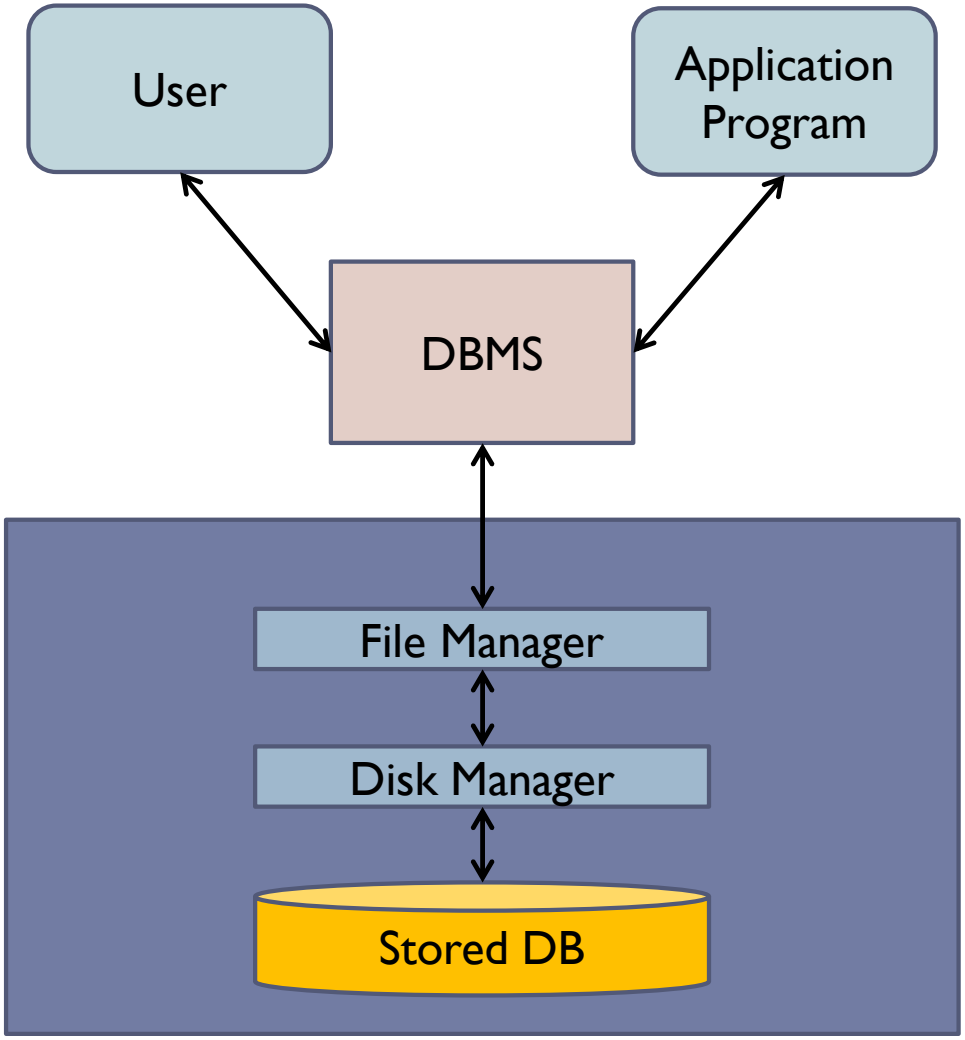
- We will focus on how data is physically stored on secondary storage
  - Different physical organisations of data
  - How the different physical organisations of data are managed

- Different physical organisations of data can significantly affect the performance of retrieving information from the database

4.1 Introduction

- Terminology
  - There are various components at the internal level which are responsible for the storage organisation and physical data management (e.g., file and disk managers). These components together with the storage media are collectively called the **physical database**.
  - Processing at the internal level is referred to as **physical database operations.**

4.1 Introduction



Primary Storage: Fastest storage medium (cache and main memory). Data can be accessed directly by CPU. Limited capacity

Secondary Storage: Media such as magnetic disks. Normally cost less but greater capacity. Slower access to data . Data must be loaded to primary storage before being operated on.

Tertiary Storage: Used mainly for backup and archival. Tapes/DVD-ROM etc.

4.1 Introduction

- How can we effectively store large amounts of data on disk?
  - Key question for database designers and database administrators
  - Different options will be available with regards to how the data can be organised on disk
- File Organisation
  - Data stored on disk will be organised as files of records.
  - Each record is a collection of data values interpreted as facts about entities, their attributes and relationships.
  - Storage of records should make it possible to locate them efficiently when needed

4.1 Introduction

- File Organisation Types
  - **Heap (**or **Unordered)**
    - Records are placed on disk in no particular order
  - **Sorted**
    - Records are ordered by the value of a specified field
  - **Hash**
    - Records are placed on disk according to a hash function

An ***access method*** is also associated with a file organisation. The access method is the steps involved in storing and retrieving records from a file.

5.1 Introduction

- Database Access
  - Consider the following example *Staff* table:

Sno	Lname	Position	NIN	Bno
SL21	White	Manager	WK4416	B5
SG37	Beech	Snr Asst	WL7868	B3
SG14	Ford	Deputy	WL87678	B3

  - We would expect each tuple to map to a record in the operating system file that holds the *Staff* relation
  - Each field in a record would store one attribute from the *Staff* relation
  - When a user requests a tuple from the DBMS, for example Staff tuple SG37, the DBMS maps this **logical record** onto a **physical record** and retrieves the physical record into the DBMS **buffers** in primary storage using the operating system's file access routines

4.1 Introduction

- Database Access
  - The physical record is the unit of transfer between the disk and primary storage and vice versa.
  - Generally, a physical record consists of more than one logical record
    - However, it is possible that a single logical record can correspond to one physical record.
    - It may also be the case that a single logical record spans more than one physical record
  - Physical records are also called **blocks** or **pages**

4.2 Heap Files

- Heap File Organisation
  - One of the simplest and most basic type of file organisation
  - Records are stored in the file in the order in which they are inserted
  - New records are inserted in the last page of the file; if there is insufficient space in the last page, a new page is added to the file
    - This makes insertion very efficient as the address of the last file back can be kept in the file header

Sometimes called a **pile** or **sequential file**

4.2 Heap Files

- Heap File Organisation
  - Searching for records is very inefficient though.
    - Since there is no particular ordering with respect to field values, a linear search must be performed to access a record.
    - A linear search involves reading pages from the file until the required record is found.
    - If only one record satisfies the search condition then on average half the file blocks will have to be transferred into main memory before the desired record is found.
    - If no records or several records satisfy the search condition then all blocks will have to be transferred.

4.2 Heap Files

- Heap File Organisation
  - What happens when we want to delete a record?
    - Physical deletion leaves unused space in the block.
    - This results in a large amount of spaces being wasted if there are frequent deletions.
    - Performance will progressively deteriorate as deletions occur
    - Heap files will require regular reorganisation to reclaim the unused space

Heap files are very efficient at bulk loading data into a table as records are inserted at the end of the heap – there is no overhead of calculating what page the record should go on.

5.3 Ordered Files

- Ordered File Organisation
  - The records in a file can be physically ordered based on the values of one or more of the fields.
  - Such a file organisation is called an **ordered file** (or **sorted file**)
  - The field(s) that the file is sorted on is called the **ordering field(s)**.

4.3 Ordered Files

- Ordered File Organisation
  - Consider the following example *Staff* table:

	Sno	Lname	Position	NIN	Bno
Block 1	SG14	White	Manager	WK4416	B5
Block 2	SG21	Beech	Snr Asst	WL7868	B3
	SG24	Ford	Deputy	WL87678	B3
. .	SG36	Brown	Assistant	WF765675	B4
. .	SG37	Black	Assistant	WD7867	B4
. .	SL20	Red	Manager	WG786	B5
	SL21	Murphy	Assistant	WF766675	B4
	SL37	Whyte	Deputy	WD78167	B3
Block 9	SL66	Blue	Manager	WG7816	B5

4.3 Ordered Files

- Ordered File Organisation
  - Consider the following SQL query:

```
SELECT *
FROM Staff
ORDER BY Sno
```

  - If the tuples are already ordered according to the ordering field *Sno* it should be possible to reduce the execution time for the query as no sorting is necessary.

5.3 Ordered Files

- Ordered File Organisation
  - Consider the following SQL query:

```
SELECT *
FROM Staff
WHERE Sno = 'SG37'
```

  - In this case we can use a ***binary search*** to execute the query involving a search condition based on the ordering field *Sno*
  - For simplicity we have assumed that there is one record per page.

4.3 Ordered Files

- Ordered File Organisation
  - Binary Search Algorithm

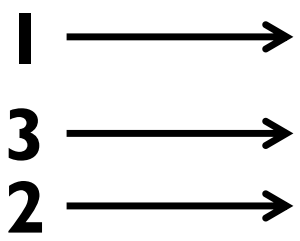
1. Retrieve the mid-page of the file. Check if the required record is between the first and last record of this page. If so, no more pages need to be retrieved – the record lies in this page
2. If the value of the key field in the first record on the page is greater than the required value, the required value if it exists, occurs on an earlier page. Therefore, we repeat step 1) using the lower half of the file as the new search area.
3. If the value of the key field in the last record on the page is less than the required value, the required value occurs on a later page, and so we repeat step 1) using the top half of the file as the new search area.
4. In this way half the search space is eliminated from the search with each page retrieved.

4.3 Ordered Files

- Ordered File Organisation
  - Binary Search Algorithm Example

```
SELECT *
FROM Staff
WHERE Sno = 'SG37'
```

1. Initial mid-page is page 4. 'SG36' is not the record we are searching for. The value we are searching for is greater than 'SG36' so we discard the first half of the file.
2. Retrieve the mid-page of the top half of the file, that is page 6. The value of the key field 'SL20' is greater than 'SG37'. Discard the top half of the search space.
3. Retrieve the mid-page of the remaining search space, that is page 5 which contains the record we are searching for.



Sno	Page
SG14	1
SG21	2
SG24	3
SG36	4
SG37	5
SL20	6
SL21	7
SL37	8
SL66	9

4.3 Ordered Files

- Ordered File Organisation
  - In general, the binary search is more efficient than a linear search.
  - Inserting and deleting records in a sorted file is problematic because the order of the records has to be maintained
    - To insert a record we must find the correct position in the file and then find space to insert the record.
    - If there is sufficient space in the required page then the page can be reordered and written back to disk.
    - If there is not sufficient space then it would be necessary to move one or more records onto the next page. This may cause a cascading effect.
    - One solution is to use an **overflow** or **transaction file**. Insertions are added to the overflow and periodically merged with the main file
      - Efficient for insertions
      - Inefficient for retrievals
    - When deleting a record we must reorganise the records to remove the free slot.

4.4 Hash Files

- Hash File Organisation
  - Records do not have to be written sequentially to the file
  - A ***hash function*** is used to calculate the address of a page in which the record is to be stored based on one or more fields in the record
  - The base field is called the ***hash field***
  - If the hash field is also a key field of the file then it is called the ***hash key***
  - Records in a hash file will appear to be randomly distributed across the available file space. For this reason, hash files are sometimes called ***random*** or ***direct*** files

4.4 Hash Files

- Hash File Organisation
  - The hash function is chosen so that records are as evenly distributed throughout the file as possible
  - Hash Function Examples:
    - **Folding**
      - This technique applies an arithmetic function to different parts of the hash field.
      - Character strings can be converted to numeric values to be used in the hash function
      - The result of the arithmetic operations determines the address of the disk page where the record is stored

5.4 Hash Files

- Hash File Organisation
  - Hash Function Examples:
    - **Division-Remainder**
      - More popular technique
      - Uses the *modulo* function
        - Divides by some pre-determined integer value and takes the remainder as the disk address

4.4 Hash Files

- Hash File Organisation
  - Can you think of any problems that may occur when using hash functions?

Main issue is that most hashing functions do not guarantee a unique address because of the number of possible values a hash field can take is typically much larger than the number of available addresses for records.

5.4 Hash Files

- Hash File Organisation
  - Hash File Operation
    - Each address generated by a hash function corresponds to a page (or a **bucket**) with **slots** for multiple records.
    - Within a bucket, records are placed in order of arrival.
    - When the same address is generated for two or more records a **collision** is said to have occurred and the records are called **synonyms** in this case.
      - We must insert the new record in another position when a collision occurs.
      - Collision management complicates hash file management and degrades overall performance

4.4 Hash Files

- Hash File Organisation
  - Collision Management with Hash Files
    - Open Addressing
      - If the required position is found occupied the program will check the subsequent positions in turn until an available space is located. Collisions increase with open addressing causing performance degradation.
    - Unchained Overflow
      - An overflow area is maintained for collisions that cannot be placed at the hash address. The record that is colliding with another will be placed in the overflow area.

5.4 Hash Files

- Hash File Organisation
  - Collision Management with Hash Files
    - Chained Overflow
      - An overflow area is maintained for collisions that cannot be placed at the hash address. However, each bucket has an additional field (sometimes called a **synonym pointer**) that indicates if a collision has occurred and if so points to the overflow page used. A linked list of overflow records for each hash address space is maintained.
    - Multiple Hashing
      - If a hash function causes a collision, a second hash function is applied

4.4 Hash Files

- Hash File Organisation
  - Dynamic Hashing
    - The hashing techniques we have considered so far are ***static*** in that the hash address space is fixed when the file is created. When the space becomes full it is said to be ***saturated.***
      - In this case it is necessary to reorganise the hash structure
      - May involve creating a new file with more space, choosing a new hash function and mapping the old file to the new file.
    - An alternative is **dynamic hashing**
      - This allows the file size to change dynamically to accommodate growth and shrinkage of the database.

4.4 Hash Files

Hash File Organisation

Limitations of Hashing

- The use of hashing for retrievals depends upon the complete hash field. In general, hashing is inappropriate for retrievals based on pattern matching or ranges of values.
- Hashing is inappropriate for retrievals based on a field other than the hash field. In this case, it would be necessary to perform a linear search to find the record

4.5 Summary

- Physical Database
- File Organisation
  - Heap Files
  - Ordered Files
  - Hash Files