الجامعة السعودية الإلكترونية
SAUDI ELECTRONIC UNIVERSITY
2011-1432

College of Computing and Informatics

Database Management Systems

IT354

Assignment #1

Deadline: Day 6/10/2024 @ 23:59

[Total Mark for this Assignment is 8]

Student Details:

Name: Reema mohammad **ID:** 220041466

CRN: 11908

Instructions:

- You must submit two separate copies **(one Word file and one PDF file)** using the Assignment Template on Blackboard via the allocated folder. These files **must not be in compressed format**.
- It is your responsibility to check and make sure that you have uploaded both the correct files.
- Zero mark will be given if you try to bypass the SafeAssign (e.g. misspell words, remove spaces between words, hide characters, use different character sets, convert text into image or languages other than English or any kind of manipulation).
- Email submission will not be accepted.
- You are advised to make your work clear and well-presented. This includes filling your information on the cover page.
- You must use this template, failing which will result in zero mark.
- You MUST show all your work, and text must not be converted into an image, unless specified otherwise by the question.
- Late submission will result in ZERO mark.
- The work should be your own, copying from students or other resources will result in ZERO mark.
- Use **Times New Roman** font for all your answers.

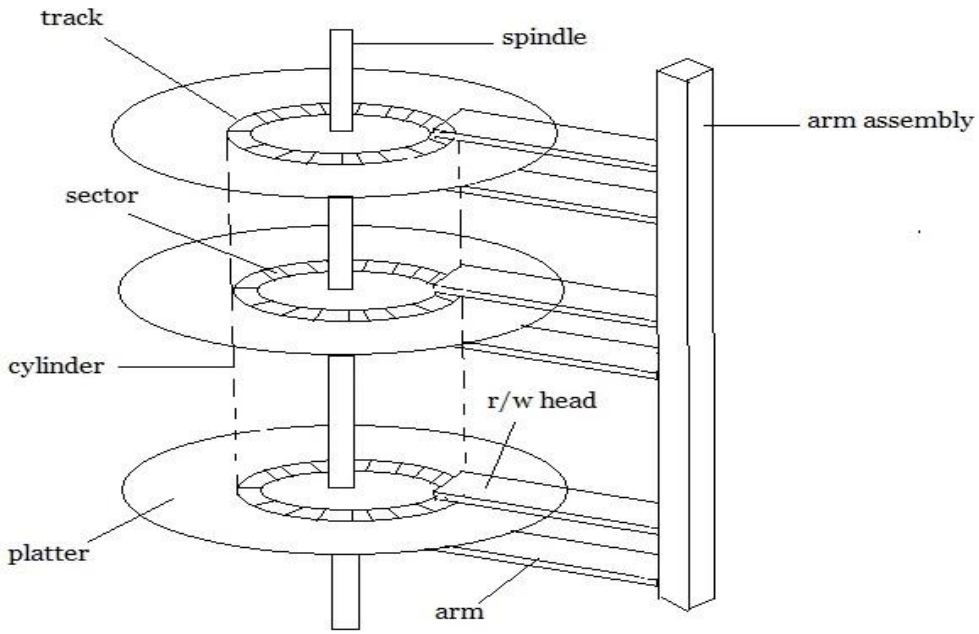
Learning Outcome(s):

CLO1: Recognize database file organization and indexing.

Question One

1.25 Marks

Discuss the structure and components of a hard disk drive. Provide an example of how data is stored on a hard disk. *(Note: You must create your own figure to illustrate this, do not copy from books, the internet, or other sources.)*



Disk Pack Structure

1. Platters

- These are the circular disks where data is stored. In the diagram, there are multiple platters stacked on top of each other, which allows for more storage capacity.
- Each platter is coated with a magnetic material that stores data as magnetic patterns.

2. Spindle

- The spindle is the axis around which the platters rotate. It holds the platters in place and spins them at high speeds (typically ranging from 5,400 to 7,200 RPM, or more in high-performance HDDs).
- The spinning enables the **read/write head** to access different parts of the platters quickly.

3. Tracks

- A track is a circular path on the surface of the platter where data is stored. Each platter surface contains numerous tracks that circle around the platter.
- In the image, tracks are depicted as concentric rings on each platter.

4. Sectors

- Tracks are further divided into smaller segments called **sectors**. A sector is the smallest physical storage unit on the platter and typically stores 512 bytes of data.
- The sectors are shown as segments of the tracks on the platter.

5. Cylinders

- A **cylinder** refers to the set of tracks that align vertically across all platters. This is an important concept because the read/write heads move together as a unit, so they access the same track on all platters simultaneously. Each "cylinder" represents all the tracks on different platters that are accessed at the same time.

6. Read/Write Head (r/w head)

- This component hovers just above the surface of the platter without touching it. It reads data from the platter by detecting the magnetic orientation of the material and writes data by changing the magnetic polarit

- There is a separate read/write head for each platter surface.

7. Arm Assembly and Arm

- The arm assembly moves the read/write heads across the platters. The arms are connected to an actuator that positions the heads precisely over the correct track to read or write data.
- The arm moves in and out over the surface of the platters as they spin, allowing the read/write heads to access different parts of the disk.

How Data is Accessed

- Data is written in **sectors** along the **tracks** of the **platters**. The **read/write heads** move together across all platters, accessing the data in **cylinders**.
- When data is requested, the **actuator arm** moves the **read/write head** to the correct cylinder and track, and the **spindle** spins the platters until the correct sector passes beneath the head.

Learning Outcome(s):

CLO1: Recognize database file organization and indexing.

CLO3: Analyze algorithms for query processing.

Question Two

(2.5 Marks)

A. Explain the difference between a primary index, a clustering index, and a secondary index in a database. In what scenarios would you use each type of index, and how does having multiple secondary indexes affect data retrieval? *Support your answer with examples.*

B. List and discuss three algorithms you would use to search for data in a database. Provide an example of each algorithm and include a figure to illustrate their performance (e.g., time complexity).

A.

Question Two

Index Type	Description	When to Use	Example	Impact of Multiple Secondary Indexes
Primary Index	- Built on the **primary key** of the table. - Ensures unique values and is usually a clustered index. - Data is physically stored in the order of this key.	- When you have a column with unique values that can identify each record. - Typically used for primary key fields.	Example: In a student database, create a primary index on the "student_id" field to quickly search for a student by their ID.	**Not applicable**, since this index is unique and based on the primary key.
Clustering Index	- Index created on **non-unique** columns. - Data is physically ordered based on the values in this index, grouping similar values together.	- When you need to group similar records together. - Used for range-based queries on non-unique fields.	Example: Create a clustering index on "department" in an employee database to group employees by department for quicker retrieval.	**Not applicable**, as clustering index affects the physical organization of data based on one column, usually only one is used.
Secondary Index	- Created on **non-primary key** columns. - Does not affect the physical ordering of the data. - Speeds up queries on non-key columns.	- When you frequently query data using columns that are not the primary key, like names or dates.	Example: Create a secondary index on "last_name" to quickly search for employees by their last name.	Having multiple secondary indexes improves search performance for different fields but **slows down insert/update** operations.

B.

1. Linear Search

- Description: In linear search, each element in the dataset is checked one by one until the desired value is found or the entire dataset is traversed.
- Time Complexity: O(n), where *n* is the number of records.
- When to Use: This is ideal for small datasets or unsorted data, as it doesn't require any ordering or indexing. It is also useful when the cost of setting up more efficient search structures (e.g., indexing) outweighs the benefits.

Example:

- In a small employee database that isn't indexed, if you're searching for an employee by their last name, you'd have to scan through each record sequentially until you find the matching name.

Performance: The worst-case scenario occurs when the searched element is at the end of the dataset or doesn't exist, leading to the need to check every record.

2. Binary Search

- Description: Binary search is a divide-and-conquer algorithm that works only on sorted data. It repeatedly divides the dataset in half and discards one half until the desired element is found or the search space is empty.
- Time Complexity: O(log n), where *n* is the number of records.

- When to Use: Binary search is used when the data is sorted, either naturally or through an index. It’s ideal for large datasets where quick retrieval is needed.

Example:

- In a student database, if the records are sorted by "student_id", binary search can be used to quickly find a specific student by their ID by halving the search space with each step.

Performance: Since binary search eliminates half of the dataset with each comparison, it is much faster than linear search for large datasets, but it requires the data to be sorted beforehand.

3. **B-tree Search**

- **Description**: A B-tree (Balanced Tree) is a self-balancing tree data structure used to efficiently manage large amounts of sorted data and enable fast searches, insertions, and deletions.

- **Time Complexity**: O(log n), where *n* is the number of keys in the tree.

- **When to Use**: B-trees are commonly used in database indexing systems because they allow fast retrieval even with large datasets, while maintaining balanced tree properties and minimizing disk I/O operations.

Example:

- In a database indexing system, B-trees might be used to index employees by "last_name". When searching for an employee's last name, the B-tree structure allows quick navigation from the root to a leaf node where the desired record resides.

Question Two

Learning Outcome(s):

CLO3: Analyze algorithms for query processing.

Question Three

2 Marks

Given the following parameters:

- index size = 5 bytes
- record pointer = 4 bytes
- block size = 512 bytes
- records = 35000

Calculate and discuss the following:

1. The size of the index entry
2. The index blocking factor
3. The number of index blocks
4. Binary search block accesses
5. The average linear search accesses
6. Discuess the average cost between linear search, binary search with indexing, and binary search on ordered records (without an index)

1. **Size of the Index Entry**:
    - Size=5 bytes (index)+4 bytes (pointer)=9 bytes
2. **Index Blocking Factor**:
    - Blocking Factor=[512 bytes/9 bytes]=56
3. **Number of Index Blocks**:
    - Number of Blocks=⌈35000/56⌉=625
4. **Binary Search Block Accesses**:
    - Accesses=log2(625)≈10 blocks
5. **Average Linear Search Accesses**:
    - Average=35000/2=17500 accesses

Cost Discussion

- **Linear Search**: Average of **17,500 accesses** (costly for large datasets).
- **Binary Search with Indexing**: Average of **38 accesses** (10 for blocks + 28 within the block).
- **Binary Search on Ordered Records**: Average of **16 accesses** (without indexing).

Learning Outcome(s):

CLO4: Develop a standard database using DBMS.

Question Four

2.25 Marks

Assume that you are working as a database administrator at Saudi Electronic University (SEU), and you need to retrieve data for the **Dammam branch only**, using relation algebra and SQL query. The data to be retrieved includes the course ID and name, the faculty member's name and department number, and the student's name.

- Create an SQL query to retrieve the required data.
- Translate the SQL query into relational algebra.
- Discuss the benefits of translating the SQL query into relational algebra.

Assumed Database Schema

1. **Courses**:
    - CourseID
    - CourseName
    - Branch
2. **FacultyMembers**:
    - FacultyID
    - FacultyName
    - DepartmentNumber
    - CourseID
3. **Students**:
    - StudentID
    - StudentName
    - CourseID

Question Four

```
SELECT
   c.CourseID,
   c.CourseName,
   f.FacultyName,
   f.DepartmentNumber,
   s.StudentName
FROM
   Courses c
JOIN
   FacultyMembers f ON c.CourseID = f.CourseID
JOIN
   Students s ON c.CourseID = s.CourseID
WHERE
   c.Branch = 'Dammam';
```

2. Relational Algebra

The corresponding relational algebra expression for the SQL query would be as follows:

1. **Selection**: Select courses from the Courses table where the branch is Dammam.
2. **Join**: Join the resulting relation with the FacultyMembers and Students tables based on the CourseID.

Relational Algebra Expression:

Let:

- C=Courses
- F=FacultyMembers
- S=Students

The relational algebra expression can be represented as:

πCourseID, CourseName, FacultyName, DepartmentNumber, StudentName((σBranch= ′Dammam′(C))⋈F⋈S)

3. Benefits of Translating SQL Queries into Relational Algebra

1. **Formal Semantics**: Relational algebra provides a mathematical foundation for relational databases, allowing for precise definitions of operations. This can help clarify what the SQL query is doing.
2. **Optimization**: Understanding the query in terms of relational algebra can help database administrators and developers optimize SQL queries. Certain operations can be more efficiently implemented depending on their algebraic representation.
3. **Verification**: By translating SQL into relational algebra, it becomes easier to verify the correctness of the query logically. You can examine the steps in relational algebra to ensure that all required data is correctly represented.
4. **Database Design**: Relational algebra can aid in database design and normalization by emphasizing the relationships between different entities in the database schema.
5. **Educational Purposes**: Teaching database concepts often involves relational algebra as a way to introduce students to the principles of querying without the complexities of SQL syntax.
6. **Performance Analysis**: Analyzing the relational algebra representation of a query can provide insights into the performance characteristics of different operations (e.g., joins, selections, projections) involved in the query.