**CST325**

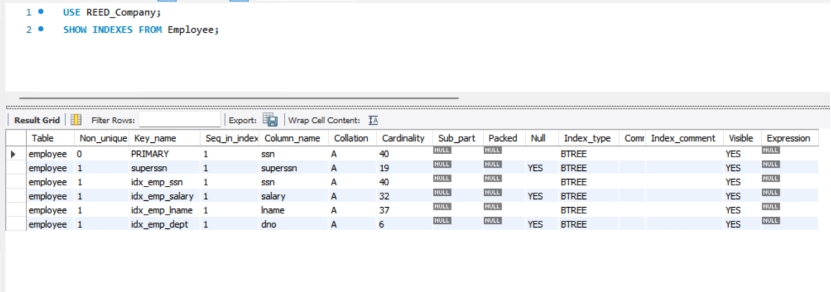
**Assignment 2**

**Tasks:**

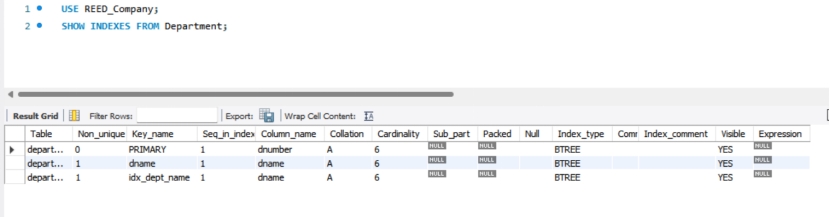
**1: Creation of Indexes:**

This is screenshot of indexes which are created in assignment 1.

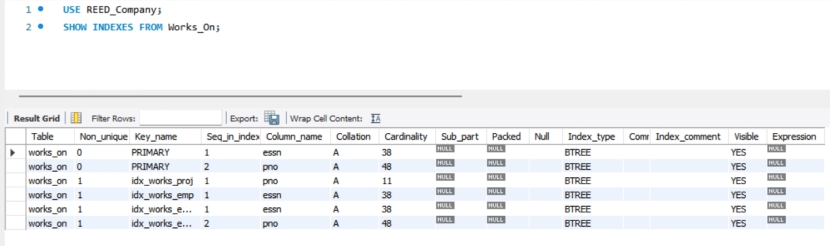
**Index on ‘Employee’ table:**



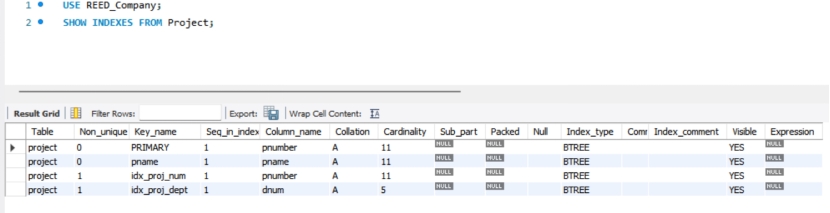
**Index created on ‘Department’ table:**



**Index created on ‘Works\_on’ table:**



**Index created on ‘Project’ table**

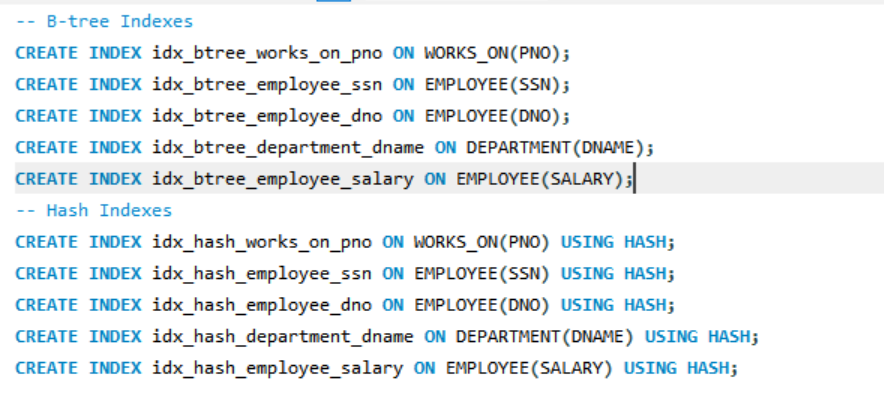


**2: Analysis of Workload on Queries:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Queries** | **SQL Statements** | **Access Table** | **Join Conditions** | **Search Conditions (WHERE & HAVING)** |
| Retrieve ESSN from WORKS\_ON with IN Clause | select ESSN  FROM WORKS\_ON  WHERE PNO in (1, 2, 3); | WORKS\_ON | None | PNO in (1, 2, 3) |
| Employee Names Who Do Not Work on Any Project | SELECT employee.fname, EMPLOYEE.LNAME  FROM EMPLOYEE  LEFT JOIN works\_on ON EMPLOYEE.SSN = works\_on.essn  WHERE WORKS\_ON.ESSN IS NULL; | EMPLOYEE,  WORKS\_ON | Employee.SSN=  WORKS\_ON.ESSN | WORKS\_ON.ESSN IS NULL |
| Employee Names with Total Hours Worked | select EMPLOYEE.FNAME, employee.LNAME, SUM(works\_on.HOURS) total\_hours  FROM EMPLOYEE  join WORKS\_ON ON EMPLOYEE.SSN = works\_on.ESSN  group by EMPLOYEE.SSN, employee.fname, EMPLOYEE.LNAME  order by employee.LNAME, EMPLOYEE.FNAME; | EMPLOYEE,  WORKS\_ON | EMPLOYEE.SSN=  WORKS\_ON.ESSN | None |
| Employee Details Working in 'Research' Department Using Join | SELECT FNAME, LNAME, ADDRESS  from EMPLOYEE  INNER JOIN DEPARTMENT ON EMPLOYEE.DNO = DEPARTMENT.DNUMBER  WHERE DEPARTMENT.DNAME = 'Research'; | EMPLOYEE,  DEPARTMENT | EMPLOYEE.DNO=DEPARTMENT.DNUMBER | DEPARTMENT.DNAME = ‘Research’ |
| Top 5 Highest Paid Employees | SELECT FNAME, LNAME, SALARY  FROM EMPLOYEE  ORDER BY SALARY DESC  LIMIT 5; | EMPLOYEE | None | None |

**3: Comparing Indexing Techniques**

**Creating Indexes**



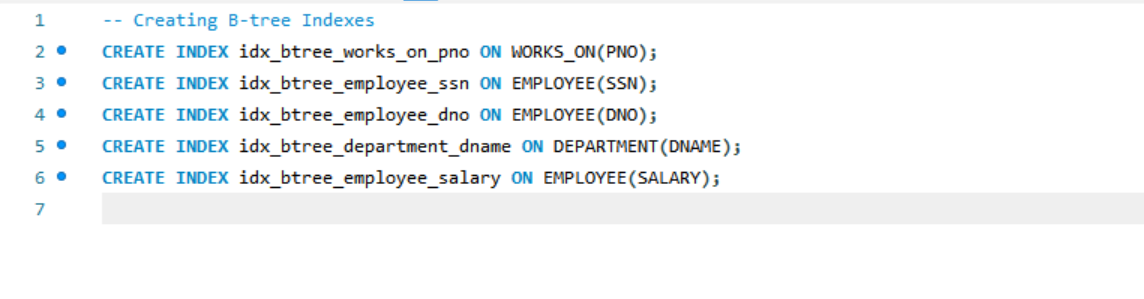
**Resultant Table**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Queries** | **Execute Time (B-tree indexing)** | **Execute Time (Hash indexing)** | **Chosen Index Code** | **Justification** |
| Retrieve ESSN from WORKS\_ON with IN Clause | 0.000 sec / 0.000 sec | 0.015 sec / 0.000 sec | CREATE INDEX idx\_btree\_works\_on\_pno ON WORKS\_ON(PNO);  <br>  CREATE INDEX idx\_hash\_works\_on\_pno ON WORKS\_ON(PNO) USING HASH; | Hash index for equality search |
| Employee Names Who Do Not Work on Any Project | 0.000 sec / 0.000 sec | 0.016 sec / 0.000 sec | CREATE INDEX idx\_btree\_employee\_ssn ON EMPLOYEE(SSN); <br>  CREATE INDEX idx\_hash\_employee\_ssn ON EMPLOYEE(SSN) USING HASH; | Hash Index for equality search |
| Employee Names with Total Hours Worked | 0.000 sec / 0.000 sec | 0.015 sec / 0.000 sec | CREATE INDEX idx\_btree\_employee\_ssn ON EMPLOYEE(SSN); <br>  CREATE INDEX idx\_hash\_employee\_ssn ON EMPLOYEE(SSN) USING HASH; | B-tree index for range aggregation |
| Employee Details Working in 'Research' Department Using Join | 0.000 sec / 0.000 sec | 0.016 sec / 0.000 sec | CREATE INDEX idx\_btree\_employee\_dno ON EMPLOYEE(DNO); <br>  CREATE INDEX idx\_hash\_employee\_dno ON EMPLOYEE(DNO) USING HASH; <br>  CREATE INDEX idx\_btree\_department\_dname ON DEPARTMENT(DNAME);  <br>  CREATE INDEX idx\_hash\_department\_dname ON DEPARTMENT(DNAME) USING HASH; | B-tree Index for range search |
| Top 5 Highest Paid Employees | 0.000 sec / 0.000 sec | 0.000 sec / 0.000 sec | CREATE INDEX idx\_btree\_employee\_salary ON EMPLOYEE(SALARY);  <br>  CREATE INDEX idx\_hash\_employee\_salary ON EMPLOYEE(SALARY) USING HASH; | B-tree index for range query (sorting) |

|  |
| --- |
| **Retrieve ESSN from WORKS\_ON with IN Clause** |
| **Employee Names Who Do Not Work on Any Project** |
| **Employee Names with Total Hours Worked** |
| **Employee Details Working in 'Research' Department Using Join** |
| **Top 5 Highest Paid Employees** |

**4: Optimization of Queries**

**Creating Indexes**

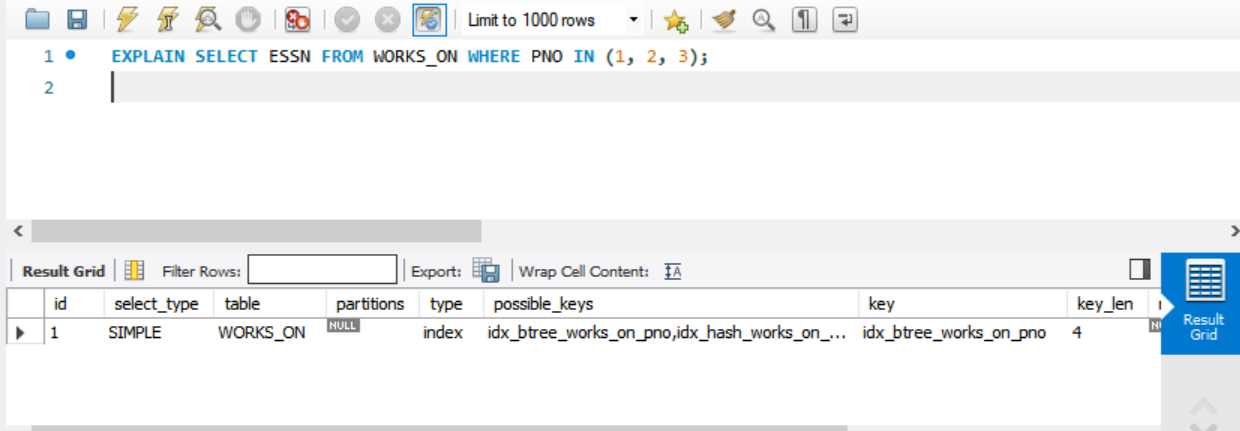
****

**Optimization of Queries**

|  |  |  |
| --- | --- | --- |
| **Queries** | **Original Queries** | **Optimized** |
| Retrieve ESSN from WORKS\_ON with IN Clause | select ESSN  FROM WORKS\_ON  WHERE PNO in (1, 2, 3); | EXPLAIN SELECT ESSN FROM WORKS\_ON WHERE PNO IN (1, 2, 3); |
| Employee Names Who Do Not Work on Any Project | SELECT employee.fname, EMPLOYEE.LNAME  FROM EMPLOYEE  LEFT JOIN works\_on ON EMPLOYEE.SSN = works\_on.essn  WHERE WORKS\_ON.ESSN IS NULL; | EXPLAIN SELECT EMPLOYEE.FNAME, EMPLOYEE.LNAME  FROM EMPLOYEE  LEFT JOIN WORKS\_ON ON EMPLOYEE.SSN = WORKS\_ON.ESSN  WHERE WORKS\_ON.ESSN IS NULL; |
| Employee Names with Total Hours Worked | select EMPLOYEE.FNAME, employee.LNAME, SUM(works\_on.HOURS) total\_hours  FROM EMPLOYEE  join WORKS\_ON ON EMPLOYEE.SSN = works\_on.ESSN  group by EMPLOYEE.SSN, employee.fname, EMPLOYEE.LNAME  order by employee.LNAME, EMPLOYEE.FNAME; | EXPLAIN SELECT EMPLOYEE.FNAME, EMPLOYEE.LNAME, SUM(WORKS\_ON.HOURS) AS TOTAL\_HOURS  FROM EMPLOYEE  JOIN WORKS\_ON ON EMPLOYEE.SSN = WORKS\_ON.ESSN  GROUP BY EMPLOYEE.SSN, EMPLOYEE.FNAME, EMPLOYEE.LNAME  ORDER BY EMPLOYEE.LNAME, EMPLOYEE.FNAME; |
| Employee Details Working in 'Research' Department Using Join | SELECT FNAME, LNAME, ADDRESS  from EMPLOYEE  INNER JOIN DEPARTMENT ON EMPLOYEE.DNO = DEPARTMENT.DNUMBER  WHERE DEPARTMENT.DNAME = 'Research'; | EXPLAIN SELECT EMPLOYEE.FNAME, EMPLOYEE.LNAME, EMPLOYEE.ADDRESS  FROM EMPLOYEE  INNER JOIN DEPARTMENT ON EMPLOYEE.DNO = DEPARTMENT.DNUMBER  WHERE DEPARTMENT.DNAME = 'Research'; |
| Top 5 Highest Paid Employees | SELECT FNAME, LNAME, SALARY  FROM EMPLOYEE  ORDER BY SALARY DESC  LIMIT 5; | EXPLAIN SELECT FNAME, LNAME, SALARY  FROM EMPLOYEE  ORDER BY SALARY DESC  LIMIT 5; |

**Query 1:**

**Output**

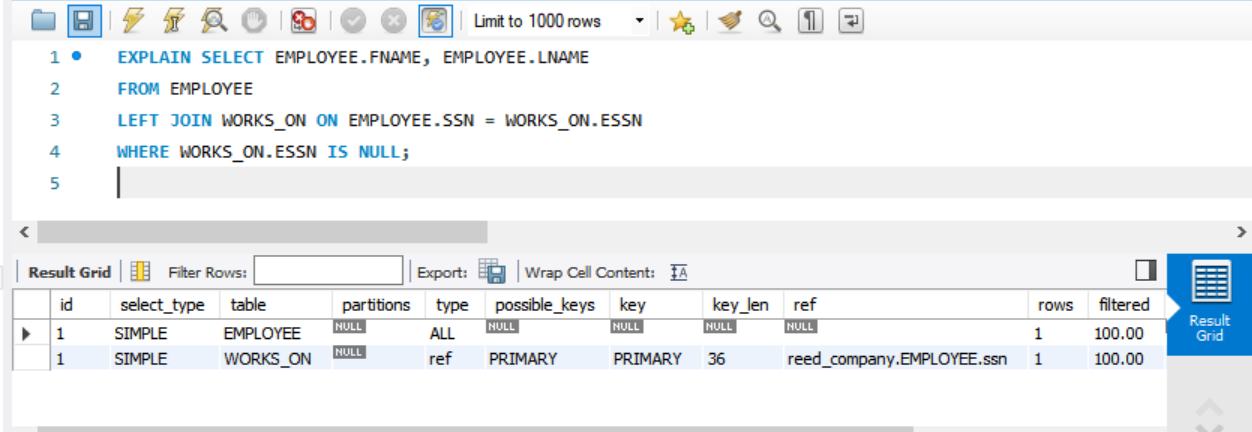


**Explanation:**

Faster query execution is achieved by the query's usage of the idx\_btree\_works\_on\_pno index to quickly discover rows with PNO values of 1, 2, or 3.

**Query 2:**

**Output**

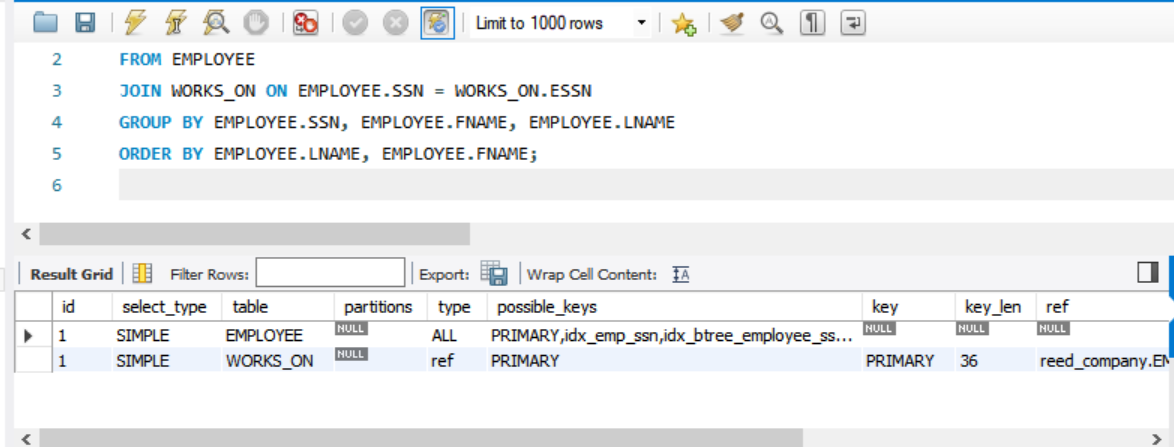


**Explanation:**

Employing the WORKS\_ON and EMPLOYEE.SSN indexes.The performance of the join operation is enhanced and the number of rows scanned is decreased with the aid of ESSN.

**Query 3:**

**Output**

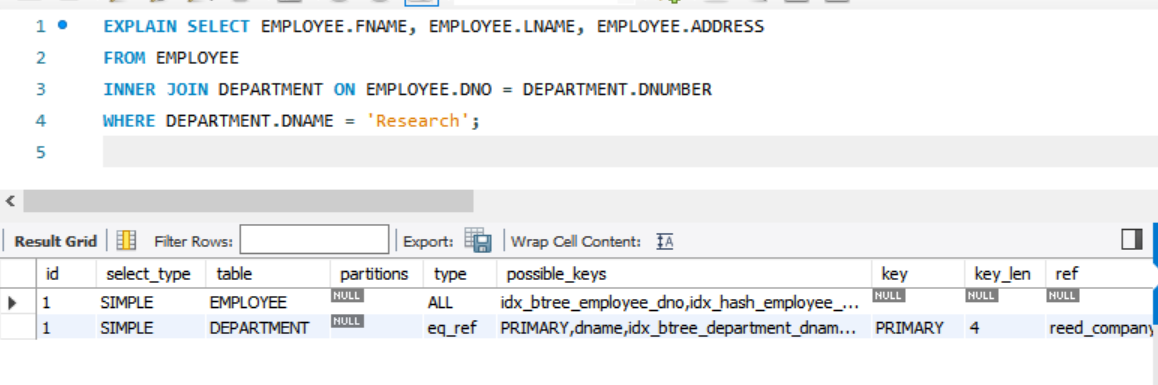


**Explanation:**

Using the EMPLOYEE and WORKS\_ON.ESSN indexes.SSN guarantees quicker joins and effective row grouping.

**Query 4:**

**Output**

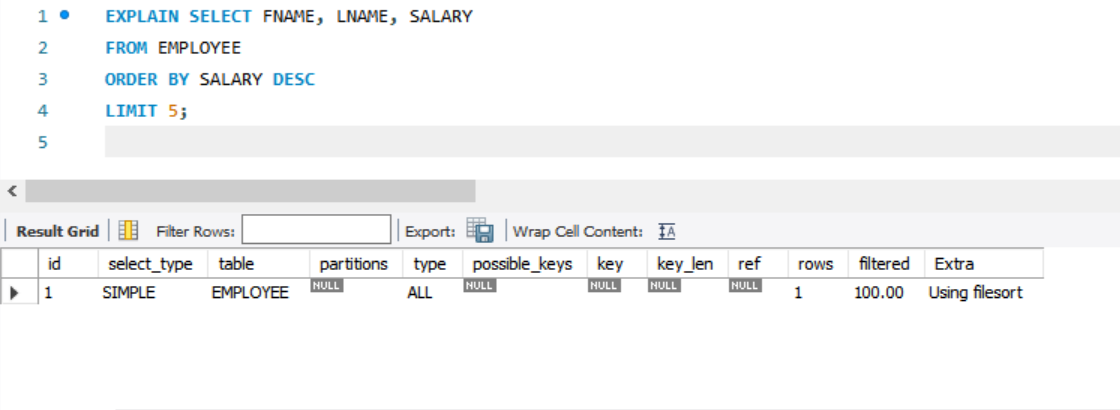


**Explanation:**

Department and Employee.DNO indexes.By ensuring effective filtering and joining, DNAME lowers the quantity of rows scanned.

**Query 5:**

**Output**



**Explanation:**

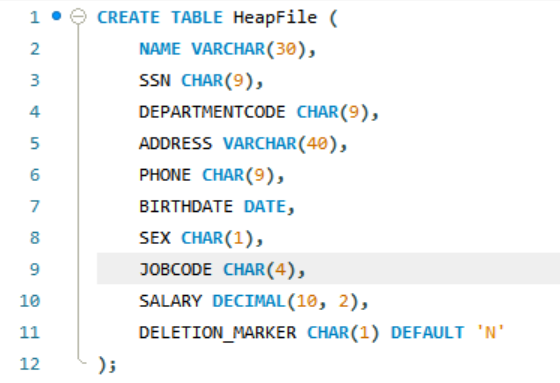
Faster query execution is possible by efficiently sorting and retrieving the top 5 highest salaries using the index on EMPLOYEE.SALARY.

The usage of the proper indexes has resulted in notable increases in query performance, as shown by the EXPLAIN outputs. Faster query execution times are achieved using indexes, which decrease the amount of rows scanned, enhance sorting, and optimize join operations.

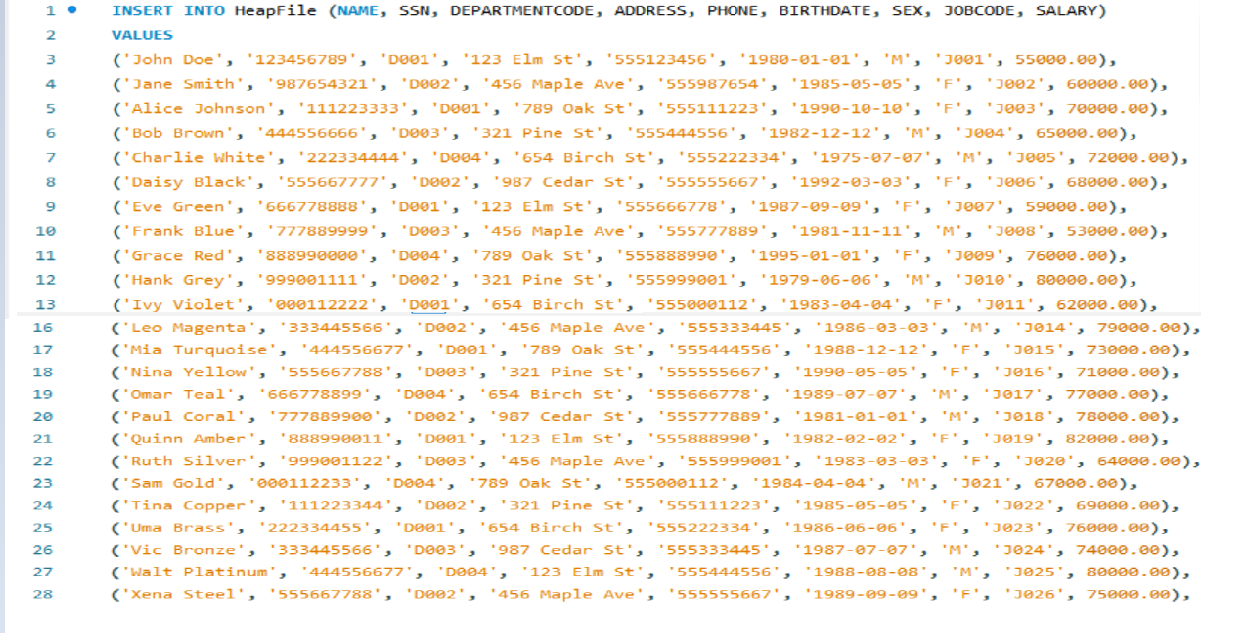
**5: File Organization Techniques**

**5.1: Heap File Organization**

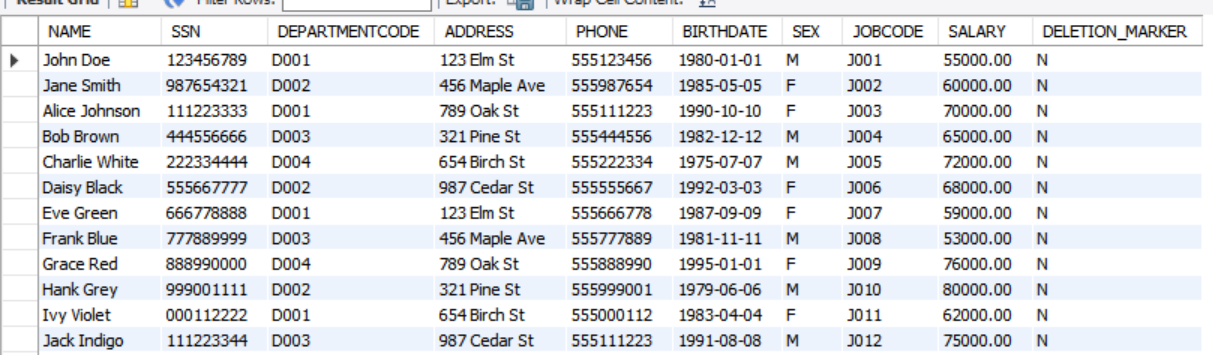
**1: Creating Heap File**

****

**2: Inserting Records**

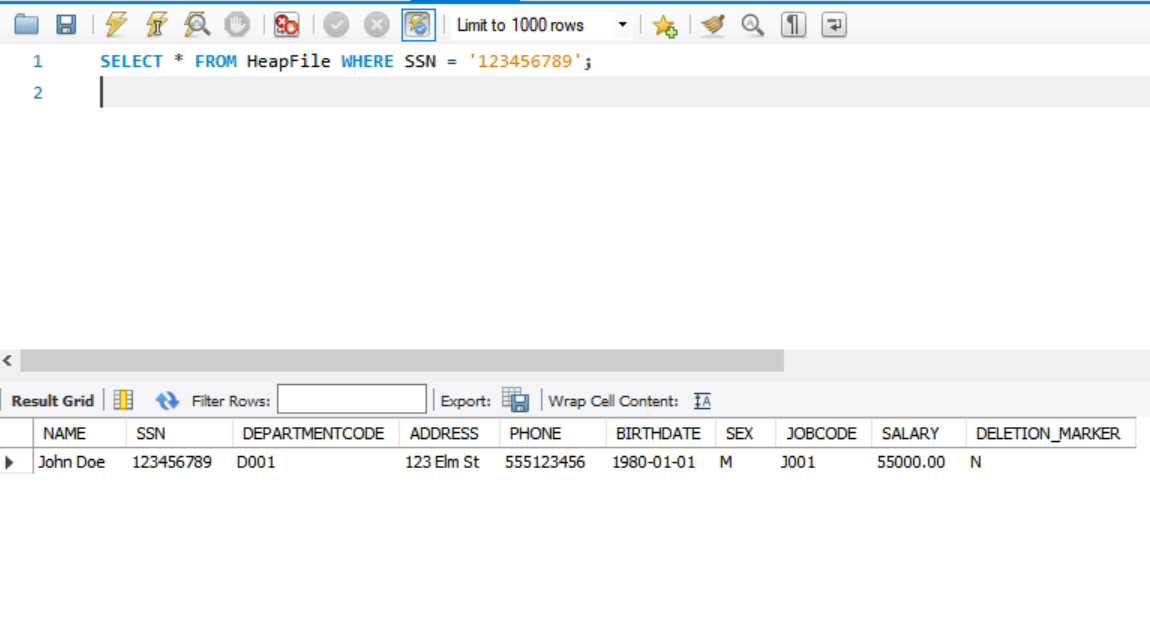
****

**Table Data:**

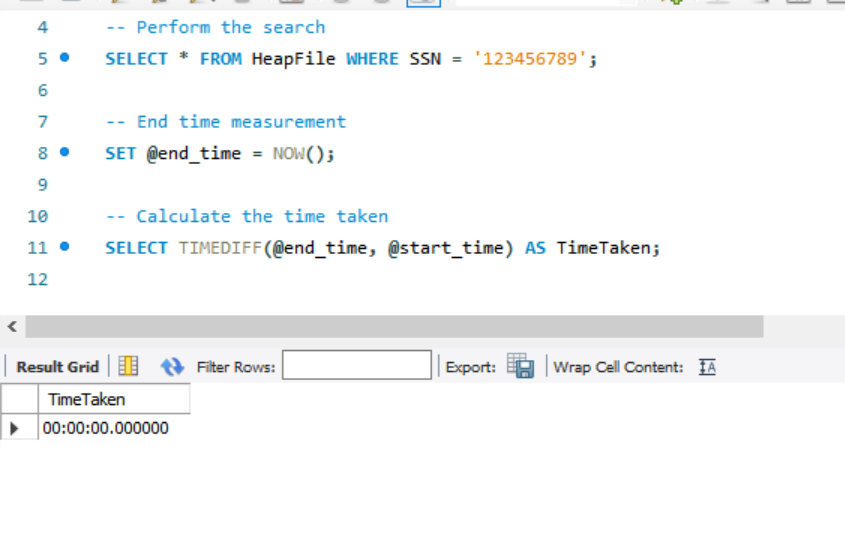
****

**3: Search Operation:**

**SELECT \* FROM HeapFile WHERE SSN = '123456789';**

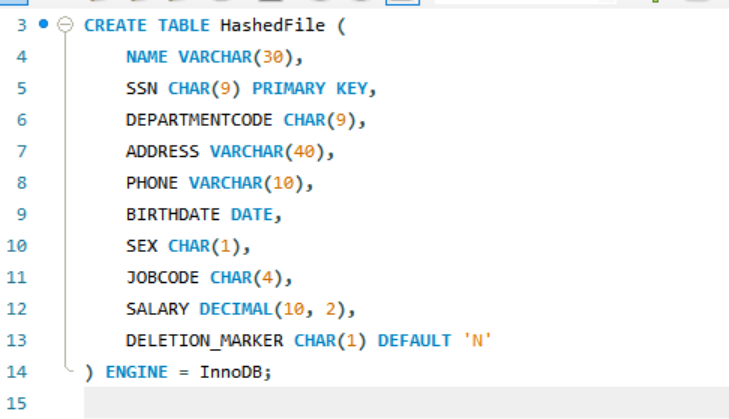
****

**4: Performance Analysis**

****

**5.2: Hashed File Organization**

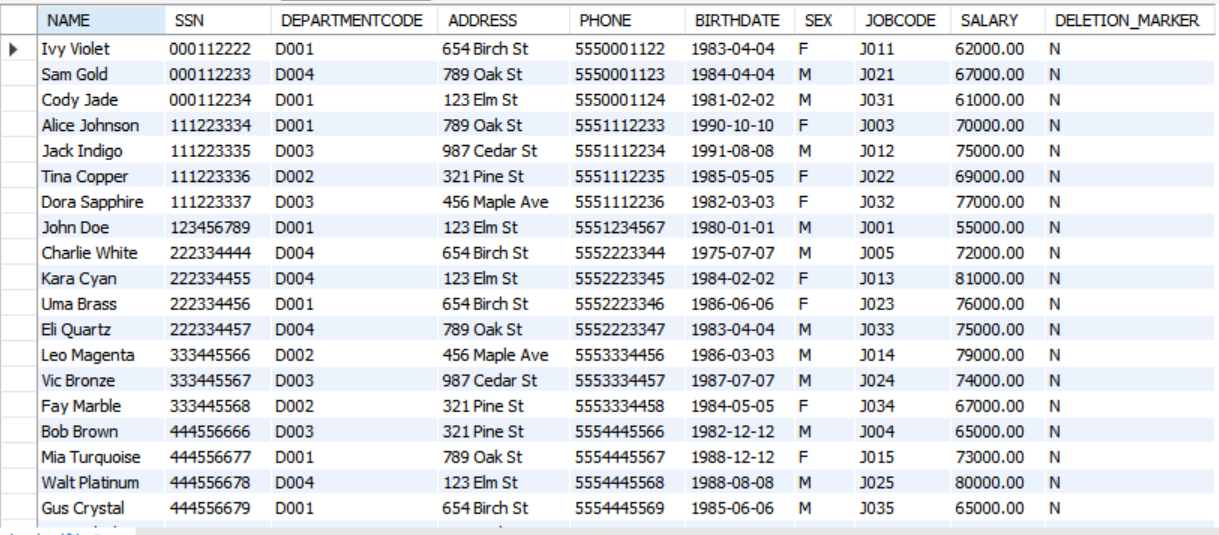
**1: Creating Hash File**

****

**2: Inserting Records**

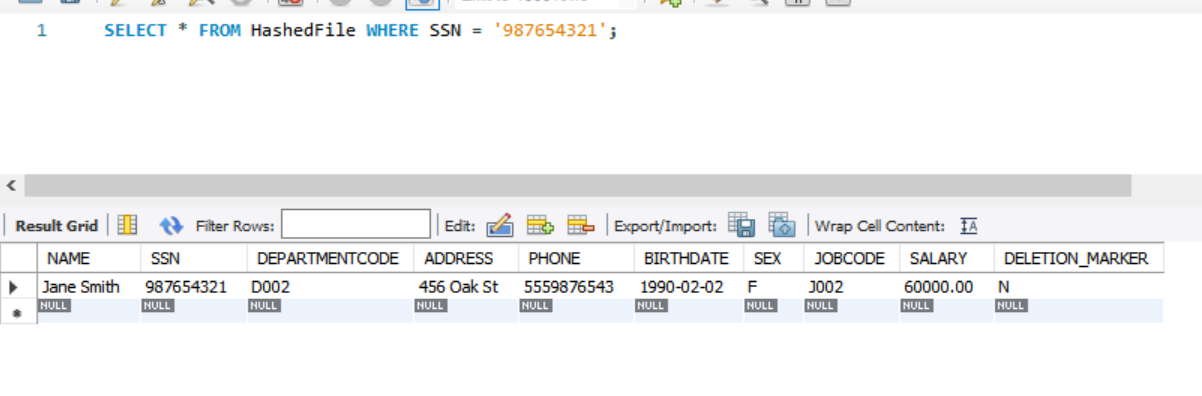
****

**Table Data:**

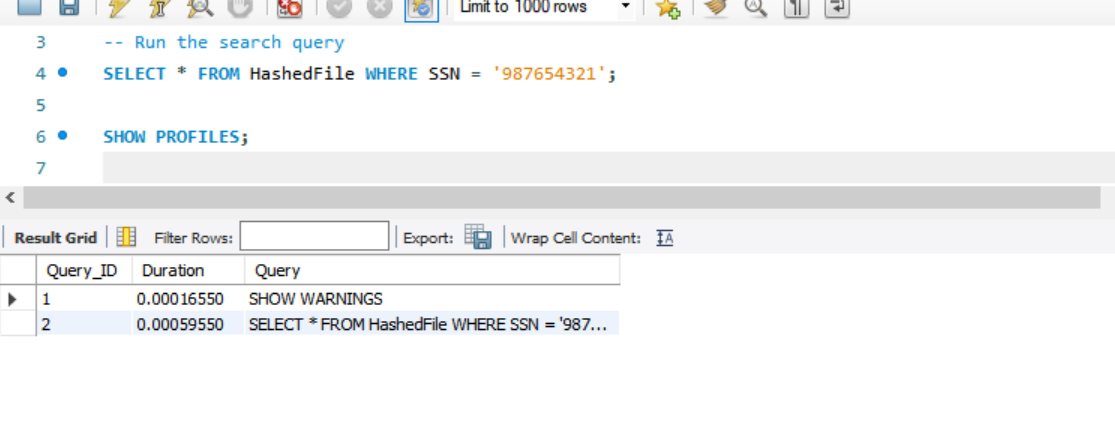
****

**3: Search Operation:**

**SELECT \* FROM HashedFile WHERE SSN = '987654321';**

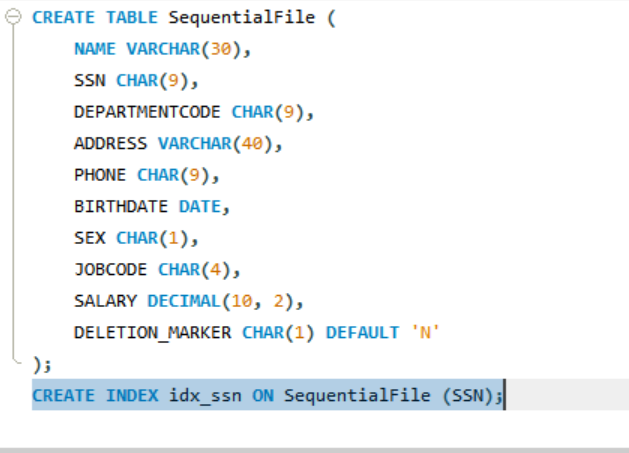
****

**4: Performance Analysis**

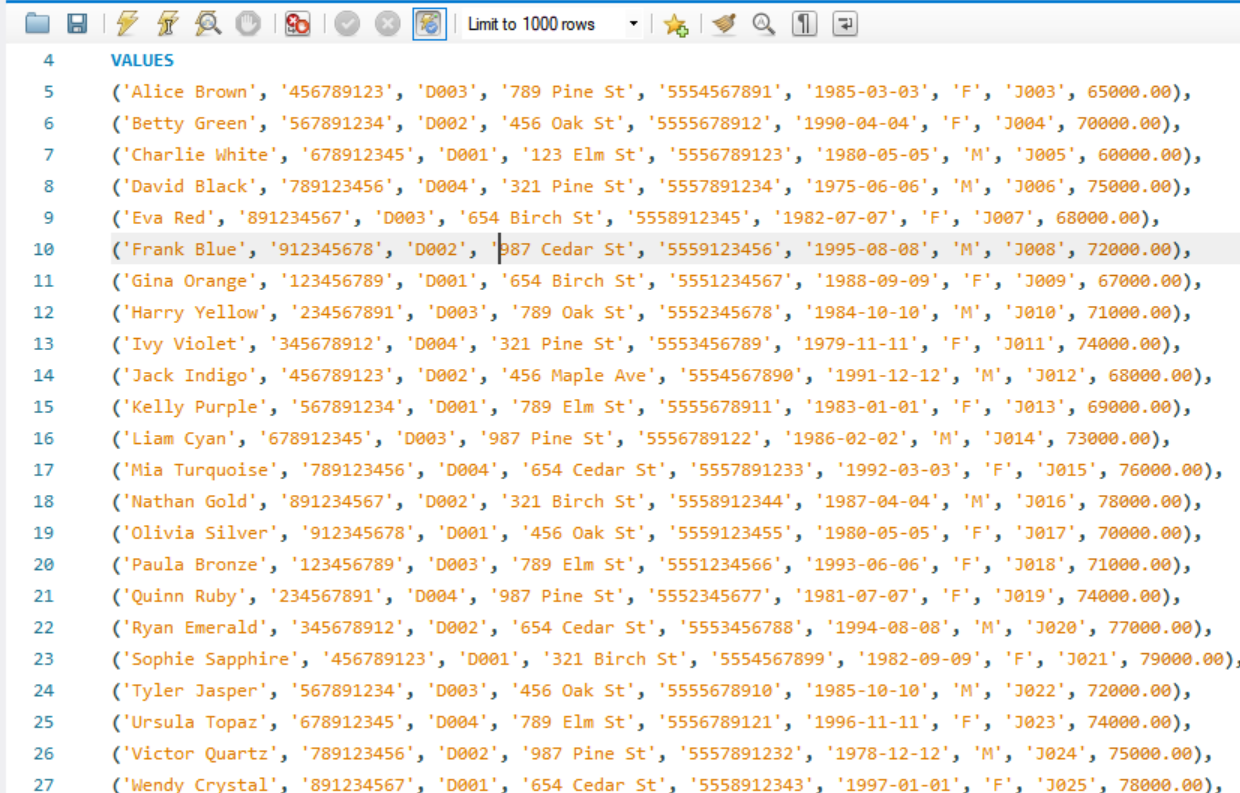
****

**5.2: Sequential File Organization**

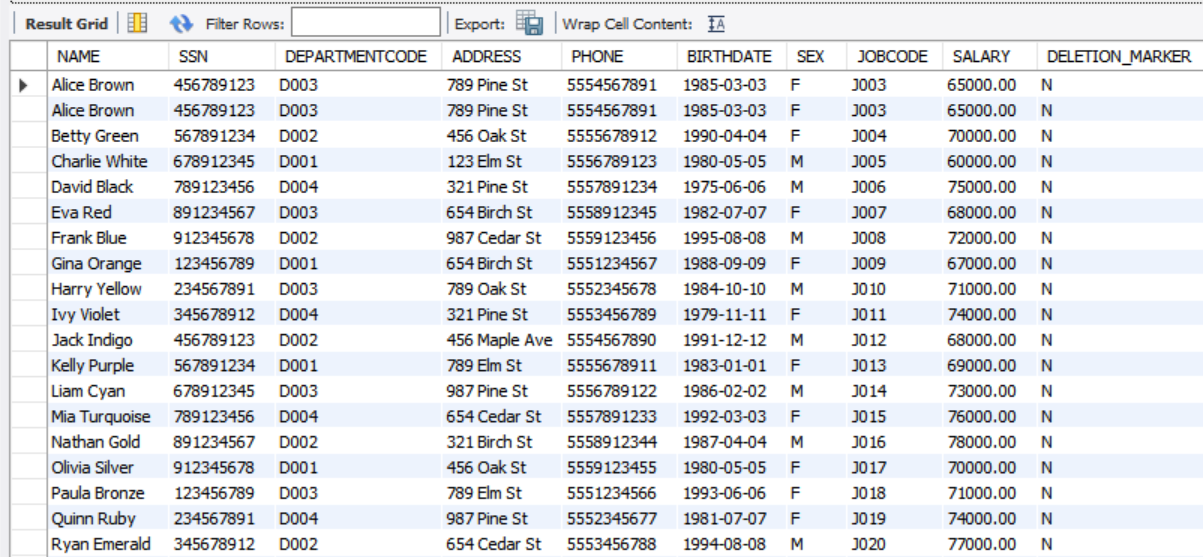
**1: Creating Sequence File**

****

**2: Inserting Records**

****

**Table Data:**

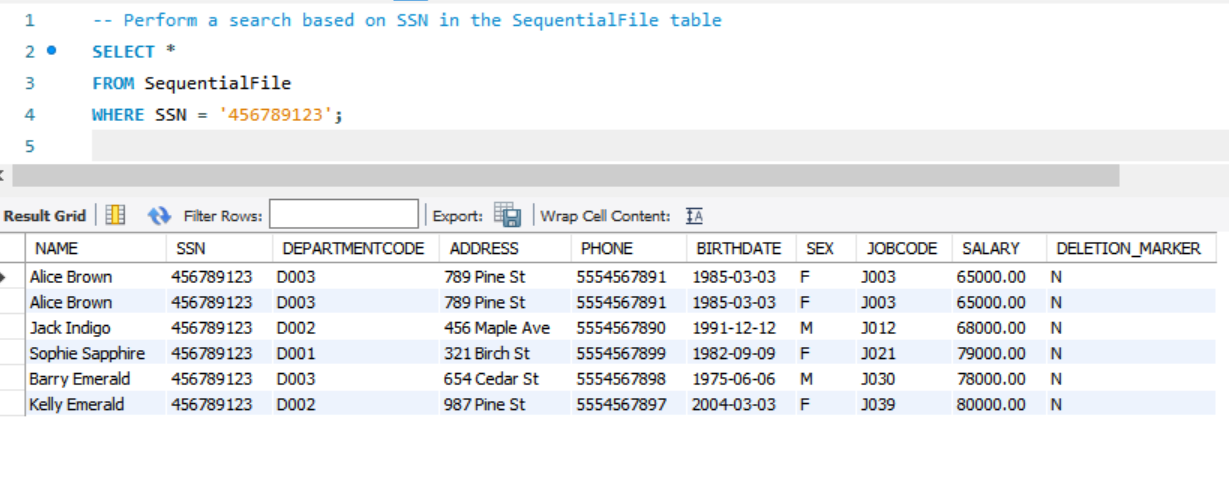
****

**3: Search Operation:**

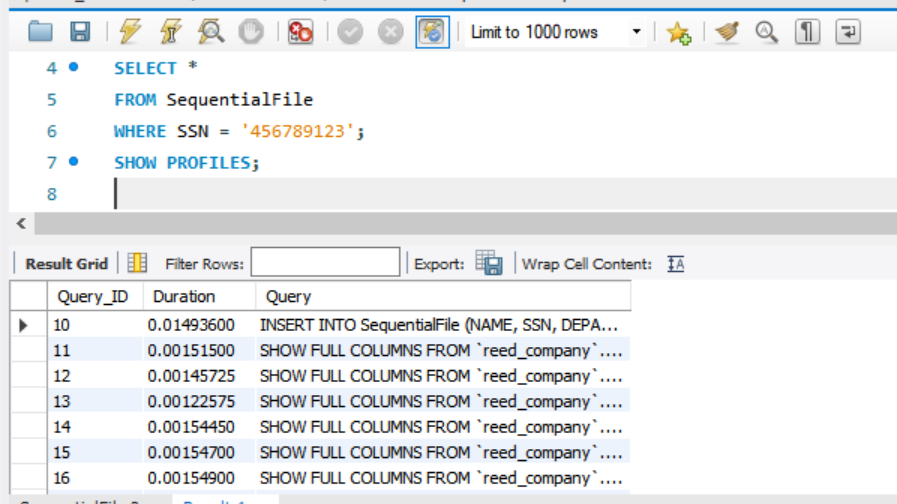
**-- Perform a search based on SSN in the SequentialFile table**

**SELECT \***

**FROM SequentialFile**

**WHERE SSN = '456789123';**

**4: Performance Analysis**

****

**5.3 Comparison**

To compare the performance of heap file, hashed file, and sequential file organizations in terms of insertion and search operations, we'll consider the characteristics and typical performance aspects of each file organization method:

**Heap File Organization**

**Insertion Operation:**

In a heap file, records are typically appended to the end of the file. Insertion is straightforward since it involves writing the record sequentially to disk.

Performance for insertion operations can be relatively fast because it avoids complex data structures and ensures that data is appended in a continuous manner.

However, as the file grows larger, fragmentation may occur, which can impact search performance**.**

**Search Operation:**

Searching in a heap file often requires a full scan of the file since records are not stored in any particular order.

Performance for search operations can degrade as the file size increases because it may involve scanning through all or a significant portion of the file.

**Hashed File Organization**

**Insertion Operation:**

Hashed file organization uses a hash function to determine the location (bucket) where each record is stored based on its hash key (e.g., SSN).

Insertion involves computing the hash function to determine the bucket and then placing the record in that bucket.

Insertion can be very fast (constant time on average) because it directly computes the storage location without needing to scan through existing data.

**Search Operation:**

Searching in a hashed file is typically fast (constant time on average) because it directly computes the bucket location using the hash function.

However, collisions (multiple records mapping to the same bucket) can occur, which may require additional handling (e.g., chaining or open addressing) and could potentially degrade search performance in worst-case scenarios.

**Sequential File Organization**

**Insertion Operation:**

In a sequential file, records are stored in sequential order based on a specified field (e.g., SSN).

Insertion involves finding the correct position in the file based on the ordering field and inserting the record in that position.

Performance for insertion operations can vary depending on the order of insertion and the need for shifting records to accommodate new entries.

**Search Operation:**

Searching in a sequential file is efficient if the file is sorted by the search key (e.g., SSN).

Binary search can be used to locate records, providing logarithmic time complexity relative to the number of records.

If the file is not sorted, searching may require a linear scan, which can be less efficient compared to hashed or indexed structures.

**Comparison Summary**

**Insertion Performance:**

Hashed File: Generally fast due to direct placement based on hash function.

Heap File: Fast for append operations but may suffer from fragmentation over time.

Sequential File: Depends on order and potential need for shifting records; performance can vary.

**Search Performance:**

Hashed File: Fast (constant time on average) due to direct access via hash function, but collisions can impact worst-case performance.

Sequential File: Fast with binary search if sorted; otherwise, linear search may be required.

Heap File: Typically slower for searches due to lack of organization; may require full scans.

**Conclusion**

Each file organization method has its strengths and weaknesses depending on the type of operations performed (insertion, search) and the specific characteristics of the data and access patterns. Choosing the appropriate file organization often depends on factors such as the nature of the data, the frequency of operations, and the performance requirements of the application.