9/10/23

# ASSIGNMENT 3



Unit - 4&5 Data Storage, Indexing and Hashing & Query Processing and Query Optimization.

Q.1 Describe in detail RAID storage technique with diagrams.

→ RAID (Redundant Array of Independent Disks) is a storage technique that combines multiple disks into a single logical unit to improve data redundancy, performance, or both. There are several RAID levels, which are:

i) RAID 0 (Stripping)

This combines two or more disks to increase storage capacity and performance. Data is divided into blocks and distributed across the disks in a striped manner. Also, there is no data redundancy, so if one drive fails, all data is lost.

RAID 0

| DISK 0 | DISK 1 | DISK 2 | DISK 3 |
|---|---|---|---|
| 0 | 1 | 2 | 3 |
| 4 | 5 | 6 | 7 |
| 8 | 9 | 10 | 11 |
| 12 | 13 | 14 | 15 |

ii) RAID 1 (Mirroring)

RAID 1 involves mirroring data on two or more disks for redundancy. Each drive contains a duplicate copy of the data, so if one drive fails, the others can still provide data

RAID 1

| DISK 0 | DISK 1 | DISK 2 | DISK 3 |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
| 2 | 2 | 3 | 3 |
| 4 | 4 | 5 | 5 |

iii) RAID 5 (Block level ~~with~~ Striping with Parity)

This uses striping for improved performance and parity information for data recovery. Data is striped across into blocks and distributed across the disks. Parity is used to recover data in case of drive failure.

RAID 5

| DISK 0 | DISK 1 | DISK 2 |
|---|---|---|
| 1 | 2 | Parity |
| 4 | Parity | 5 |

Q.2 Explain Heuristic in Optimization. Discuss the main heuristics that are applied during query optimization.

→ Heuristics in optimization are problem-solving techniques that provides a practical approach to finding good solutions to complex problems. It does not guarantees optimal solutions but are used to determine the most effecient way to execute a database query by choosing a suitable query execution plan, in database query optimization's content.

-4 Some main heuristics applied during query optimization are:

i) Cost-Based Optimization

It evaluates the potential cost of executing various query plans and select the plan with the lowest estimated cost. It includes factors (for estimation) like I/O operations, CPU processing time and network transfer costs.

ii) Join Order Optimization

It focuses on determing the order in which tables are joined in a multi-table query. It includes strategies like left-deep and bushy join trees which aims to reduce the number of intermediate results and improve query performance.

iii) Index Selection

It helps to determine which indexes to use for efficient data retrieval. It allows the database system to quickly locate rows that satisfy certain conditions.

iv) Predicate Pushdown

It involves moving filtering conditions (predicates) as close to the data source as possible, which reduces the amount of data that needs to be processed by the query.

v) Memory Management
It considers the amount of available memory and determine the optimal memory allocation for sorting, joining and intermediate result storage.

Q.3 Explain file organization in detail with example
&
Q.4 Describe different ways of organizing records in a file

→ File organization refers to the way data is structured and stored within a computer file or a collection of ~~data~~ files. It determines how data is accessed, updated and managed.

→ Various file organization methods are:

i) Sequential File Organization.
Here, data is stored in a linear, sequential order, with each record following the previous one. Records can only be accessed sequentially, from the beginning of the file to the end.

eg: Consider a simple text file where each line represents a student's name and their corresponding grade.

ii) Indexed File Organization.
Here, records are stored sequentially, but an index structure is maintained separately which allows for direct access to records

based on a key-value.

eg: Consider a file with customer records which are stored sequentially. An index is maintained mapping customer IDs to their respective positions in the file, for direct access.

iii) Hashed (Direct) File Organization.

Here, records are not stored sequentially but are placed in specific locations using a hashing ~~alog~~ algorithm based on a key value. This allows for fast, direct access to records based on their key.

eg: Consider a file storing employee records. Each record is placed in a location determined by a hash function based on the employee's ID.

iv) Random File Organization.

Here, data records are stored in a way that doesn't follow any specific order. It is often used in scenarios where you need fast, direct access to records based on a specific key or attribute

eg: when you have a unique Student ID (Primary key), student's name and their grades then you can use a hashing function to determine where each record should be stored based on their Primary key.

Q.5 Describe indexing in detail. Describe its Types in detail.

→ Indexing is a technique used to improve the speed of data retrieval operations on large datasets. It involves creating additional data structures, called indexes, that store a subset of the data in a more efficient form.

→ Types of Indexing are:

i) Primary Index

It is an index created on the primary key of a table. It enforces the uniqueness of the primary key and allows for efficient retrieval of data based on the primary key. Typically, implemented as B-Tree or B+Tree structures.

ii) Secondary Index

It is an index created on non-primary key columns of a table, which allows for fast retrieval of data based on attribute other than the primary key. It includes B-Tree, hash and bitmap indexes.

iii) Clustered Index

It determines the physical order of data in a table. In most RDBMS, a table can have only one clustered index. The leaf nodes contain the actual data rows and the index defines the order in which data is stored on disk. A table's primary key is typically implemented

as a clustered Index.

ii) Non-clustered Index

They are separate from the data rows and contain a copy of the indexed columns along with a pointer to the actual data row. Multiple non-clustered indexes can be created on a single data. They are generally used for columns that are frequently used in queries for filtering or sorting.

v) Bitmap Index

They uses a bitmap data structures to represent the values of indexed columns. Each bit corresponds to a unique value in the indexed columns. They are efficient for columns with low cardinality (a small number of distinct values). Used for data warehousing and decision support systems.

Q.6 Describe Primary and Secondary indexing.

→ i) Primary Indexing

A primary index is the means of organizing and accessing data in a database. It is created on the primary key of a table, which is a column or a set of columns that uniquely identifies each row in the table.

It enforces the uniqueness of values in the indexed columns, ensuring that no two rows can have the same values. A Table can

have only one primary index. It plays a fundamental role in database integrity, ensuring data uniqueness and allowing for efficient retrieval of specific rows.

eg: Consider a "Students" table, where the "StudentID" column is automatically becomes the primary index that allows for fast retrieval of individual records based on their unique IDs.

ii) Secondary Indexing.

It is used to enchance the performance of queries that involve columns other than the primary key. It allows for fast retrieval of rows based on values that are not part of the primary key.

Secondary indexes are not limited to unique values, multiple rows can have the same indexed value. A table can have multiple secondary indexes, each targeting different non-key columns. It is a trade-off between query performance and storage overhead, as indexes require additional drive space and maintenance.

eg: In the same table, create a secondary index on "LastName" column to quickly retrieve records based on their last name without affecting the primary index.

Q.7 State the differences between sparse index and dense index.

| Sparse Index | Dense Index |
|---|---|
| It is an indexing technique where only some of the data blocks are indexed | It is an indexing technique where every record has associated index |
| They are smaller in size | They are larger in size |
| They require more time to locate the data | They require less time to locate the data |
| Each entry points direct to a data block containing ~~multiple~~ records. | Each entry point directs to individual records |
| They are more storage efficient because they only store entries for the keys that are present in the data. | They consume more storage space because they store entries for every possible key value. |
| They are less efficient for searching record | They are more efficient for searching record. |

Q.8 Discuss data dictionary with an appropriate example.

→ A data dictionary is a centralized repository of metadata that provides information about the data and data structures within a database. It serves as a reference and documentation tool for data administrators, developers and users to understand the organization, structure and meaning of data elements.

- Data Dictionary includes components like:

i) Table information: Such as their names, owners and descriptions.

ii) Column information: Such as name, data type, length, description and any constraints.

iii) Relationships: Description of the relationships between tables like keys and joins condition.

iv) Index information: Details about indexes created on tables, including index names, columns and type.

v) Security information: Information related to data access and security, including user privileges and roles.

eg: Data dictionary for Employee information system

i) Table Name: Employee

> Description: This table contains information about company employees

ii) Columns: EmployeeID, Name.

> Employee ID : Data-type ⇒ integer, size ⇒ 10 bytes, constraints ⇒ primary key.
> Name : Data-type ⇒ String, size ⇒ 25 bytes, constraints ⇒ not null.

iii) Indexes : Unique index on email column.

iv) Relationships : Employee (Department ID) → Department (Department ID)

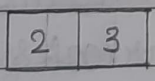
> Description : Each employee is associated with a department.

Q.9 Construct B+ Trees for the following ~~data~~ set of key values - (2, 3, 5, 7, 11, 17, 19, 23, 29, 31). Assume that the tree is initially empty and the number of pointers that will fit in one node is 4 and 6.

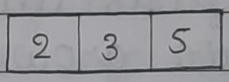
Constructing B+ Tree of order 4.

① Insert 2

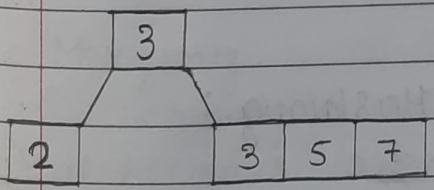

② Insert 3



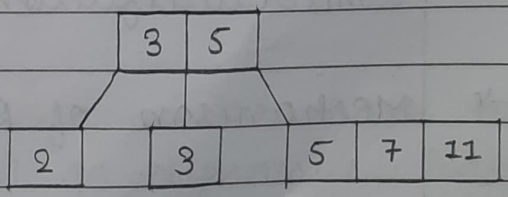

③ Insert 5



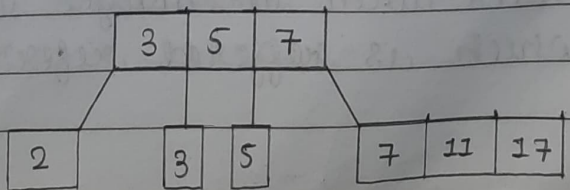

④ Insert 7




⑤ Insert 11

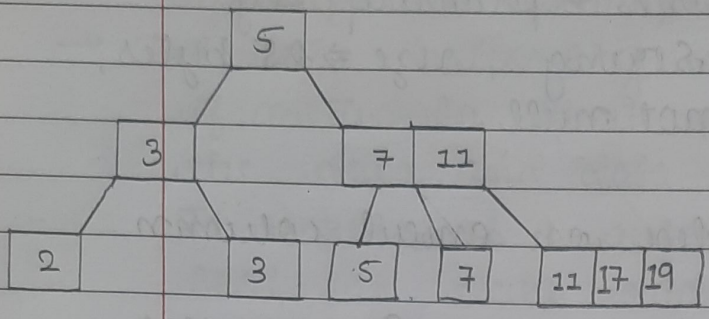



⑥ Insert 17

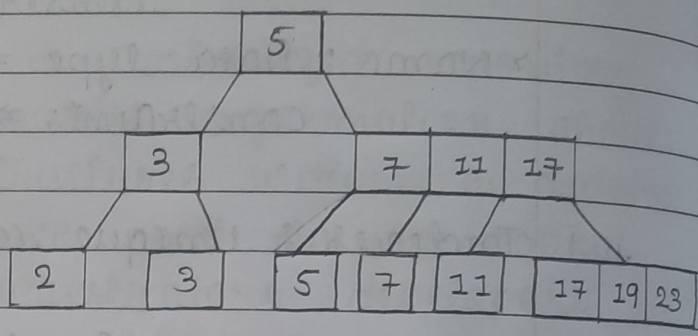


* Order - 6 structure is at pg - 82

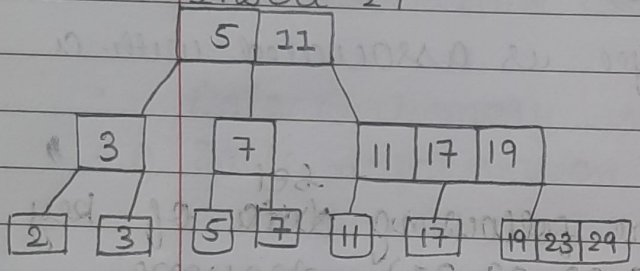
⑦ Insert 19



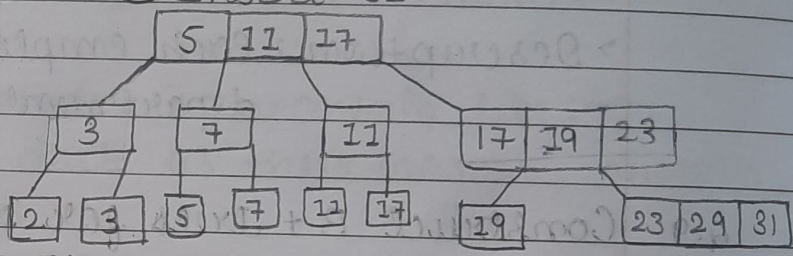

⑧ Insert 23




⑨ Insert 29




⑩ Insert 31




Q.10 Describe dynamic hashing and explain mechanism of dynamic hashing.

→ Dynamic hashing, also known as extendible hashing, is a technique used to efficiently manage data distribution and storage. It adjusts the number of buckets as data grows or shrinks. This approach helps to distribute data more evenly and minimizes the potential for bucket overflow or underutilization.

→ Mechanism of Dynamic Hashing:

i) Initialization

It starts with a single and fixed sized bucket which is referred to as the

root or primary bucket which contains all the data

ii) Data Insertion

When inserting a new element into the hash table, compute it's hash value. Use a portion of the hash value to determine which bucket the element should be placed in.

iii) Bucket Splitting

When data is inserted into the structure, if a collision occurs (i.e. multiple keys map to the same bucket), dynamic hashing takes action by splitting the full bucket. To split the bucket, you increase the number of bits used to determine the bucket index.

iv) Directory Expansion

When a bucket is split and the directory needs expansion, it effectively doubles in size. The existing directory is copied and one bit is added to each entry, which creates new entries for the expanded directory.

v) Merging

As data is removed from the hash tables, it becomes underutilized so dynamic hashing can also merge buckets to reduce the size of the directory.

Q.11 Discuss join order optimization and heuristic optimization algorithm.

→ i) Join Order Optimization

It involves finding the optimal order in which to perform table joins in a multi-table SQL query. The goal is to minimize the total cost of query execution, which typically includes I/O, CPU, Network and Memory cost.

The challenge in join order optimization is that there are many possible join orders and evaluating all of them to find the best one can be computationally expensive. Also, it is essential for improving query performance in database queries.

ii) Heuristic Optimization Algorithm

They are the methods that provide approximate solutions to complex problems, often based on a set of predefined rules. These algorithms may not guarantee an optimal solution but they aim to find a good solution with limited computational resources.

They offer a practical and efficient way to address the combinatorial complexity of join order optimization which can significantly improve query performance by finding good join orders, even in complex and large database environments.

Q.12 Discuss hashing along with its types.

→ Hashing is a process used to convert input data (like key) into a fixed-size string of characters, which is typically a numeric value. The output, known as hash code/value, is generated by mathematical function called hash function.

- Types of hashing :-

× i) Cryptographic Hashing

It is used in security applications to ensure data integrity, authentication and confidentiality. It have specific properties including collision resistance, pre-image resistance and avalanche effect.

× ii) Perfect Hashing

It is used to eliminate collisions entirely in certain scenarios. It's often used in scenarios where the set of keys is known and static.

(1) iii) Static Hashing

It is also known as fixed or closed hashing, is a technique in which the number of hash buckets is predetermined and data not change during the lifetime of the hash table. Each data item is hashed to a specific bucket using a hash function and if there's a

collision, various methods are used to handle these collision.

② Dynamic Hashing

It is also known as open or extendible hashing, is a technique in which the number of hash buckets can change dynamically as the dataset grows or shrinks. The goal is to maintain a more adaptive data structure that can efficiently accomodate varying numbers of elements.

Q.13 Discuss and describe bitmap indices with its usage.

→ Bitmap indices are a type of data structure used to efficiently store and retrieve data for specific attributes or columns in a table. It uses bitmaps (binary strings) to represent the presence or absence of values within a column.

- Structure: It consists of one bitmap for each distinct value within the indexed column. Each bitmap contains one bit for each row in the table, with each bit indicating 'set(1)' the presence or 'clear(0)' the absence of the corresponding value for that row.

- Usages of Bitmap Indices:-

i) Boolean Queries: They are highly efficient for

Boolean queries, where the condition involve the logical AND, OR or NOT operation.

iii) Low Cardinality Columns: They are well-suited for columns with low cardinality, meaning they have a small number of distinct values

iii) Data Warehousing: They are commonly used in data warehousing and analytics applications. They are useful for decision support systems, as they can significantly speed up query performance for specific types of queries

iv) Data Mining and OLAP: They are valuable for online analytical processing systems and data mining applications. They allow for efficient filtering and aggregation of data based on specific attributes.

v) Categorical Data: They are useful for indexing and searching categorical data, such as product categories, gender or status flags.

vi) Combining Multiple Indices: They can be combined to create composite indices that accelerate complex queries

Q.14 State the comparison between B Tree and B+ Tree.

—4

| B Tree | B+ Tree |
|---|---|
| > No duplicate of keys is maintained in the tree | Duplicate of keys are maintained and all nodes are present at leaf. |
| > Insertion takes more time and it is not predictable sometimes | Insertion is easier and the results are always the same |
| > Deletion of the internal node is very complex, and the Tree has to undergo a lot of transformation | Deletion of any node is easy because all nodes are found at leaf |
| > They involve random disk access, making them less effective for sequential access | They are more efficient for sequential access as only leaf stores data. |
| > B Trees are used in Databases, search engines | B+ Trees are used in Multilevel indexing, Database indexing |
| > Number of nodes at any intermediary level 'l' is $2^l$. | Each intermediary node can have n/2 to n children |

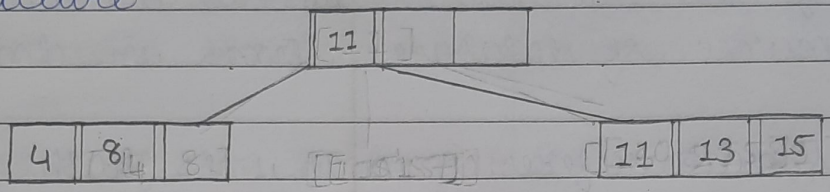
Q.15 Differentiate indexing and hashing.
—4

| Indexing | Hashing |
|---|---|
| > It is a technique that allows to quickly retrieve records from database file | It is a technique that allows to search location of desired data without using index structure. |
| > It is generally used to optimize or increase performance of database | It is generally used to access and retrieve items in database. |
| > It uses data reference to hold address of ~~th~~ disk block | It uses hash functions to calculate direct location of records on disk. |
| > It is considered best for small databases | It is considered best for large databases. |
| > Types of indexing include ordered, primary, secondary, clustered | Types of hashing includes static and dynamic hashing. |
| > Its main purpose is to provide basis for both rapid random lookups and efficient access of ordered records. | Its main purpose is to use math problem to organize data into easily searchable buckets. |

Q.16 Demonstrate the searching of given element in B+ trees with necessary example.

→ Searching for an element in a B+ Tree involves traversing the tree from the root node to find leaf node where the element is located. They are particularly well-suited for range queries and sequential access.

eg: Suppose we have a B+ tree with the following structure




Here, we will search for value '15'.

- Searching Process:
  - i) Begin the search at the root node, which contains the keys '11'.
  - ii) Compare the search key (15) with the keys in the root node. Here, "15" is greater than '11', so we move to the right child.
  - iii) In the internal node, compare the search key (15) with the keys in current node.
  - iv) Since, we have found the key '15', we can conclude the search is successful.

Q.17 Consider a B Tree with key size = 10 bytes, block size of 512 bytes, data pointer is of size 8 bytes and block pointers is 5 bytes. Find the order of B Tree.

→ Here, Key Size = 10 bytes ⇒ K
Block Size = 512 bytes ⇒ B D
Data pointer size = 8 bytes ⇒ R
Block pointer size = 5 bytes ⇒ B

If the order of B tree is 't', then

$$t \times B + (t-1)(K) \leq D$$
$$\therefore t \times 5 + (t-1)(10) \leq 512$$
$$\therefore 5t + (10t - 10) \leq 512$$
$$\therefore 5t + 10t - 10 \leq 512$$
$$\therefore 15t \leq 512 + 10$$
$$\therefore 15t \leq 522$$
$$\therefore t \leq 522/15$$
$$\therefore t \leq 34.8$$
$$\therefore \boxed{t = 34}$$

So, the order of B tree is '34'.

**Q.18** Describe various factors used to evaluate the indexing.

→ Evaluating Indexing methods is essential as it helps optimize query performance and data retrieval. Various factors are considered when assessing the effectiveness and efficiency of indexing techniques, some key factors are:

i) Query Performance: The primary goal of indexing is to improve query performance. Evaluate

how well an indexing methods accelerates the execution of common and critical queries.

ii) Index Size: The size of indexes affects storage ~~of insertion, update and deletion operations~~ requirements. Smaller indexes consume less disk space and memory, which can improve query performance by reducing I/O.

iii) Selectivity: Index selectivity measures the uniqueness of the indexed values. Highly selective indexes are more efficient because they reduce the number of rows accessed during retrieval.

iv) Concurrency: Analyze the impact of indexing on concurrent access. Consider how well the indexing method handles concurrent queries and update.

v) Maintenance Overhead: Determine how quickly indexes ~~or~~ can be rebuilt or reorganised when needed. Minimizing maintenance overhead is crucial for keeping indexes up-to-date.

Q.19 Explain how an expression can be evaluated with the help of materialization and pipeline approach.

→ Evaluating an expression can a computational

system can be done using either the materialization or the pipeline approach, depending on the specific requirements and constraints of the system.

i) Materialization Approach.

Here, the system computes and stores intermediate results or the final result in memory before further processing. Here's how evaluation works with this approach:

a) Parse and Compile : The expression is parsed and an abstract syntax tree (AST) is construct and then compiled into an executable program.

b) Evaluation : The executable program run on the input data. Then the result are computed and stored in persistent storage.

c) Post-Processing : Futher processing can be performed on the result if needed. Results are returned to the user or stored in a database or file system.

ii) Pipeline Approach.

Here, the system processes data and expressions incrementally as they flow through the pipeline, without storing intermediate results in persistant storage. Here's how evaluation works with this approach :

a) Stream Processing : Data is ingested into the system as a continuous stream (eg: event data or sensor readings).

b) Expression Evaluation : Evaluated incrementally as the data flows through the pipeline. The results are computed and passed downstream without being stored.

c) Final Processing : If necessary, intermediate result are combined to produce final result. Results are continuously generated and can be delivered in real-time.

Q.20 Discuss the procedure to evaluate an expression in query processing.

—4 Evaluating an expression in query processing is a fundamental task in data retrieval. The procedure to evaluate an expression follows :

i) Parsing : Firstly, The expression is parsed to understand its structure and semantics. This involves tokenizing the expression and building an AST that represents its structure.

ii) Optimization : Before the evaluation, it may undergo optimization to improve its efficiency by minimizing the number of operations.

iii) Type Checking : It is performed to ensure that

the expression is well-formed and that the data types are compatible.

iv) Evaluation: Actual evaluation begins. This step depends on the type of expression and the available evaluating mechanisms. Some common types of expressions and their evaluation procedures a) Arithmetic, b) Boolean, c) Comparison, d) Aggregate, e) String

v) Error Handling: During evaluation, errors may occur, such as division by zero or undefined values. Proper mechanisms are essential to identify and report errors to users.

vi) Result Generation: The results of the expression evaluation are generated. It can be a single value, a set of values, boolean values, etc.

vii) Output and Presentation: The final results may need to be formatted, presented and delivered to users in a meaningful way. Structures like tables, JSON or other data structures.

Q.21 Explain typical query processing strategies of DBMS.

—↳ Same answer as Question (20).

Q.22 Explain materialized views.

—↳ A Materialized views is a database object that

stores the results of a precomputed query as a static table. It is a database feature used to improve query performance and reduce the overhead of repeatedly executing complex queries.

- Key Characteristics :-

i) Precomputed Results : Materialized views store the computed results of a query that retrieves and aggregates data from one or more base tables.

ii) Storage : They are physically stored in database, occupying disk space.

iii) Query Performance : They significantly improve query performance for complex, time-consuming or frequently executed queries.

- Advantages :-
  - > Query Performance
  - > Reduced Resource Consumption
  - > Complex Query Simplification
  - > Offline Processing

- Limitations :-
  - > Storage Overhead
  - > Data consistency
  - > Maintenance Overhead
  - > Query Relevance

Q.23 Explain various steps involved in query processing with functioning of each step.

→ Same answer as Question (20)

Q.24 Write a short note on intrusion detection.

→ Intrusion Detection is a critical security measure, aimed at identifying and responding to unauthorized or suspicious activities within a database environment. It is a vital aspect of maintaining data integrity, confidentially and availability as well as protecting against security breaches and data theft.

- Some key aspects of intrusion detection :-

i) Types of Intrusion : It is designed to detect various types of intrusion, including unauthorized access, data breaches, SQL injection attacks, etc.

ii) Monitoring and Analysis : It continuously monitor network and system activities and analyze the data for anomalies and known attacks patterns.

iii) Alerting : When it identifies suspicious activities, it generates alerts and they are sent to system administrators for investigation.

iv) Event logging : It maintains detailed logs of all

detected events, providing an audit trail for forensic analysis and incident response.

(v) Signature-Based Detection: It compares observed system activities against a database of known attack signatures. If a match is found, it triggers an alert.

(vi) Real-Time Response: It can take real-time actions, such as blocking network traffic or terminating processes, to prevent potential intrusions.

Q.25 let Relation R1(A,B,C) & R2(C,D,E) have following properties. R1 has 10,000 tuples & R2 has 15,000 tuples where 20 tuples of R1 on one block and 15 tuples of R2 on one block. Estimate no. of block access required using each of the following join strategies of R1 & R2:

→ R1 has 10,000 tuples — so, R1 needs 500 blocks
R2 has 15,000 tuples — so, R2 needs 1000 blocks.

let us assume M pages of memory.

- If $M > 500$, then join can easily be done in 1000 + 500 disk access

So, we consider only case where $M \leq 500$ pages.

a) Hash Join

Here R1 < R2 and assuming that no overflow occurs

- If, $M > \frac{500}{M}$, then cost is $3(1000+500) = 4500$

or else the cost will be,

$$2(1000+500)[\log_{M-1}(500-1)] + 1000 + 500$$

b) Block Nested loop Join

- If R1 is the outer relation,

$$\left[\frac{500}{M-1}\right] \times 1000 + 500 \text{ disk access}$$

- If R2 is the outer relation,

$$\left[\frac{1000}{M-1}\right] \times 500 + 1000 \text{ disk access}$$

c) Nested Loop Join

- using R1 as the outer relation,
  $10000 \times 1000 + 500 = 10000500$ disk access
- using R2 as the outer relation,
  $15000 \times 1000 + 500 = 15000500$ disk access.

Q.26 Discuss the problems in query optimization.

→ Common issues and problem in query optimization are:

i) Combinatorial Explosion

The number of possible query execution plans grows exponentially, so it makes it very challenging to consider all the possibilities.

ii) Cost Estimation:

It can be challenging, especially for complex queries and in the presence of varying data distributions and statistics.

iii) Data Skew

Uneven data distribution, where certain values or keys appear more frequently than others, can lead to suboptimal execution plans.

iv) Index Selection

Choosing the right indexes for query predicates and join conditions is crucial because selecting inappropriate indexes can lead to poor performance.

v) Resource Constraints

limited system resources, such as CPU, memory and disk I/O, can influence the choice of execution plans.

vi) Parallelism and Consistency.

Optimizing queries in parallel and multi-user environments requires dealing with issues related to resource contention, locking and coordination of concurrent queries.

vii) Data Partitioning

In distributed databases, query optimization needs to consider data partitioning and data placement to minimize data movement across nodes.

Q.27 list the techniques to obtain the cost of a

query and discuss any of them.

—> The cost of query is typically an estimation of the resources and time required to execute the query. Some of the common techniques used to obtain the cost of a query:

i) Cost - Based Optimization (CBO)
ii) Query Parsing and Analysis
iii) Cardinality Estimation
iv) Cost Models
v) Cost Functions
vi) Statistics and Metadata
vii) Join Order Analysis : For queries with multiple joins, determing the order in which tables are joined is critical. The query optimizer explores different join order possibilities and estimates the cost of each. Join Order Analysis can have a significant or impact on query performance.

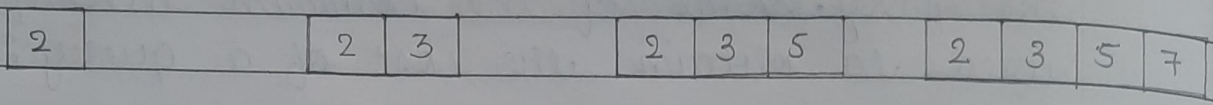
Q.28 Explain how an expression can be evaluated with the help of materialization and pipeline approach.

—> Same answer as Question (19)

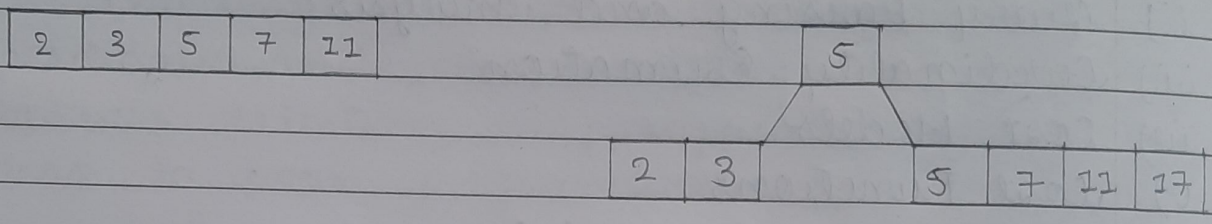
~~~ x ~~~ x ~~~

Answered 1+1+623

V. good
~~~

Q.9_b Constructing B+ tree of order 6.

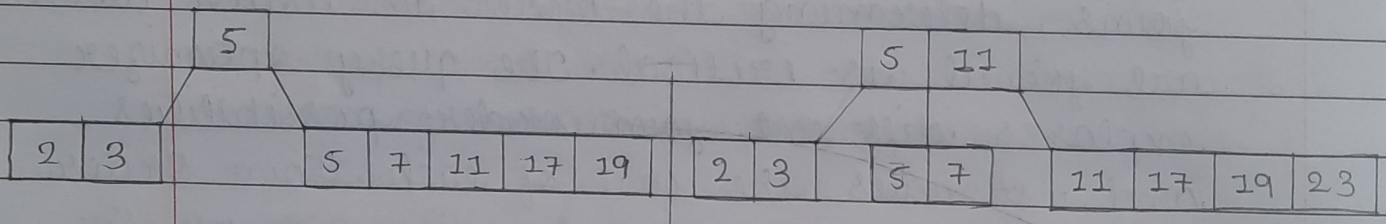
① Insert 2 ② Insert 3 ③ Insert 5 ④ Insert 7



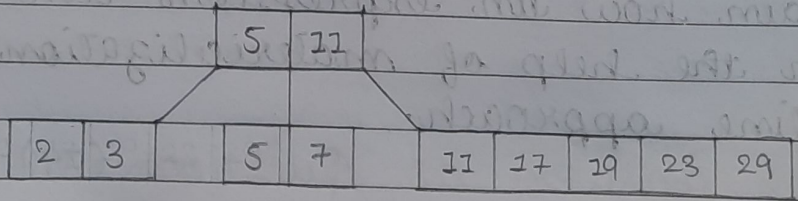

⑤ Insert 11 ⑥ Insert 17




⑦ Insert 19 ⑧ Insert 23




⑨ Insert 29




⑩ Insert 31.