1

(a) Factors to Consider While Choosing RAID Levels

When choosing a RAID level, several factors need to be considered to ensure the storage solution meets the needs of the application or system. These factors include:

1. Performance (Read and Write Speed):
   - The speed at which data can be read from and written to the storage array.
2. Fault Tolerance and Redundancy:
   - The ability to continue operating in the event of a disk failure.
3. Storage Efficiency:
   - The ratio of usable storage space to the total raw storage capacity.
4. Cost:
   - The cost of implementing and maintaining the RAID setup, including the number of disks required.
5. Complexity:
   - The complexity of setup, management, and recovery in case of failures.

Comparison of RAID Levels 0, 1, and 5

RAID 0 (Striping):

- Performance: High read and write speed since data is striped across multiple disks, allowing multiple reads/writes simultaneously.
- Fault Tolerance: None. If a single disk fails, all data is lost.
- Storage Efficiency: 100%, as all disk space is used for data storage.
- Cost: Low, as no redundancy is provided, hence fewer disks are needed.
- Complexity: Simple to set up and manage.

RAID 1 (Mirroring):

- Performance: Good read speed (reads can be performed from either disk), but write speed is slightly slower due to mirroring.
- Fault Tolerance: High. Data is mirrored on two disks, so if one disk fails, the other can be used.
- Storage Efficiency: 50%, as half of the storage is used for redundancy.
- Cost: High, as it requires double the number of disks for the same amount of usable storage.
- Complexity: Simple to set up and manage.

RAID 5 (Striping with Parity):

- Performance: Good read speed and moderate write speed. Writes are slower than RAID 0 due to parity calculations.
- Fault Tolerance: Good. Can tolerate the failure of one disk. Data can be rebuilt from parity information.
- Storage Efficiency: (N-1)/N, where N is the number of disks. For example, with 4 disks, efficiency is 75%.
- Cost: Moderate, as it provides redundancy with less overhead than RAID 1.
- Complexity: More complex to set up and manage, especially in the case of a disk failure and recovery.

(b) Deletion Techniques for a File of Records

Considering the deletion of record 5 from the given file:

i) Move record 6 to the space occupied by record 5, and move record 7 to the space occupied by record 6, and so on.

- Merits:
    - Maintains the order of records.
    - No need for additional markers or data structures.
- Demerits:
    - Requires multiple record movements, which can be time-consuming, especially for large files.
    - Can lead to fragmentation and inefficiency over time.

ii) Move record 10 to the space occupied by record 5.

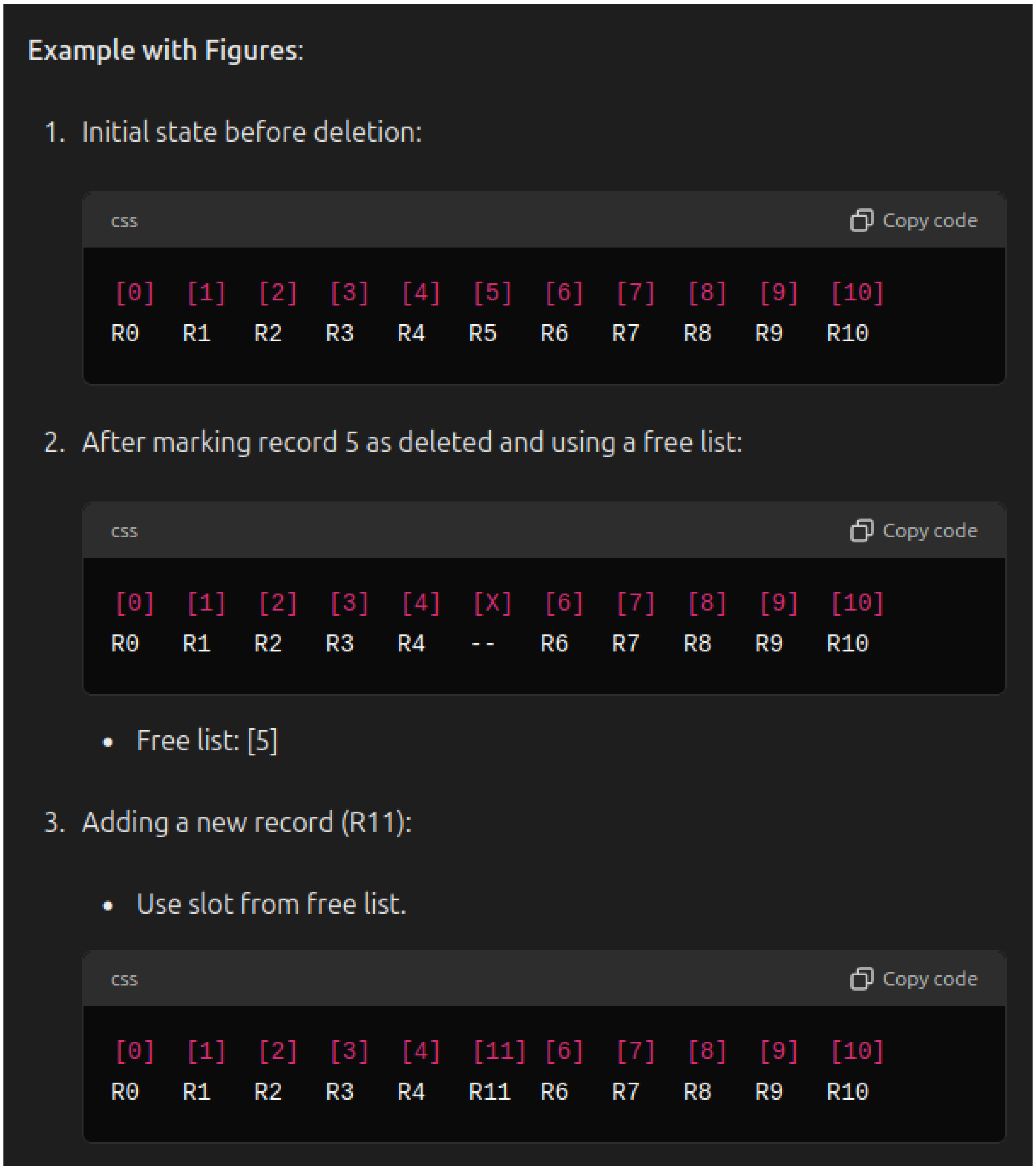
- Merits:
  - Fewer movements compared to the first method (only one record is moved).
  - Simple to implement.
- Demerits:
  - Disrupts the order of records, which might be important for certain operations.
  - Inconsistencies in sequential access.

iii) Mark record 5 as deleted, and move no records.

- Merits:
  - Very efficient as no records need to be moved.
  - Simple to implement and manage.
- Demerits:
  - Leaves gaps in the file, leading to wasted space.
  - Requires handling of "deleted" markers during searches and other operations.

Using a Free List:

- Free List:
  - A data structure that keeps track of all free (deleted) slots in a file.
  - When a new record is added, it can be placed in one of the free slots rather than at the end of the file.



Example with Figures:

1. Initial state before deletion:

css Copy code

```css
[0]  [1]  [2]  [3]  [4]  [5]  [6]  [7]  [8]  [9]  [10]
R0   R1   R2   R3   R4   R5   R6   R7   R8   R9   R10
```

2. After marking record 5 as deleted and using a free list:

css Copy code

```css
[0]  [1]  [2]  [3]  [4]  [X]  [6]  [7]  [8]  [9]  [10]
R0   R1   R2   R3   R4   --   R6   R7   R8   R9   R10
```

- Free list: [5]

3. Adding a new record (R11):

- Use slot from free list.

css Copy code

```css
[0]  [1]  [2]  [3]  [4]  [11] [6]  [7]  [8]  [9]  [10]
R0   R1   R2   R3   R4   R11  R6   R7   R8   R9   R10
```

(c) What is Metadata? How it Helps Manipulating a Database?

Metadata:

- Metadata is data that describes other data. It provides information about a certain item's content.

Types of Metadata:

1. Descriptive Metadata: Information that describes the content, quality, condition, and other characteristics of data (e.g., title, author, keywords).
2. Structural Metadata: Information about how data is organized (e.g., tables, columns, indexes).
3. Administrative Metadata: Information needed to manage data, including rights, preservation information, and technical information (e.g., file type, access permissions).

How Metadata Helps in Database Manipulation:

1. Data Organization: Metadata helps in organizing data by providing a structure (e.g., table schemas, column data types).
2. Search and Retrieval: Descriptive metadata allows for efficient search and retrieval of data by providing keywords and descriptions.
3. Data Integrity and Quality: Administrative metadata ensures data integrity and quality by tracking changes, managing versions, and enforcing data standards.
4. Access Control: Metadata can include information on who has access to what data, ensuring security and proper access management.
5. Interoperability: Metadata facilitates interoperability between different systems and databases by providing standardized descriptions and formats.
6. Data Analysis: Metadata supports data analysis by providing context and additional information necessary for interpreting data correctly.

In summary, metadata is crucial for efficient database management, enhancing data retrieval, integrity, security, and overall usability.

(a) Improving Throughput and Response Time with Parallel Systems

Throughput:

- **Definition**: Number of queries processed per unit of time.
- **Improvement with Parallel Systems**: Multiple processors handle different queries simultaneously, increasing the overall number of queries processed.

Response Time:

- **Definition**: Time taken to execute a single query.
- **Improvement with Parallel Systems**: A single query can be divided into sub-queries and processed concurrently by multiple processors, reducing the execution time.

(b) Factors Affecting Speedup and Scaleup in Parallel Databases

Speedup:

- **Definition**: Measure of how much a parallel system improves performance over a single processor system.
- **Factors**:
  - **Parallel Overhead**: Coordination and communication costs between processors.
  - **Load Balancing**: Equal distribution of work among processors.
  - **Data Skew**: Uneven data distribution causing some processors to work longer than others.

Scaleup:

- **Definition**: Measure of a system's ability to handle increased workload by adding more resources.
- **Factors**:
  - **Linear Scaleup**: Doubling resources (processors, storage) should double the workload handled.

    - **System Bottlenecks**: Limits in network bandwidth, I/O performance, and inter-process communication.
    - **Software Efficiency**: The ability of database management software to utilize additional resources effectively.

(c) Comparison of Shared Memory and Shared Disk Architectures

Shared Memory:

- **Description**: Multiple processors share the same physical memory.
- **Benefits**:
    - Fast communication between processors.
    - Easier to implement and manage.
- **Limitations**:
    - Scalability issues due to memory access contention.
    - Limited by the physical size of memory.

Shared Disk:

- **Description**: Each processor has its own memory but all share the same disk storage.
- **Benefits**:
    - Better scalability compared to shared memory.
    - Easier to manage data consistency and availability.
- **Limitations**:
    - Potential bottlenecks at the disk controller.
    - More complex to implement.

Benefits and Limitations of Using Distributed Databases

Benefits:

- **Scalability**: Easier to scale horizontally by adding more nodes.
- **Availability**: Improved fault tolerance and high availability through data replication.
- **Performance**: Reduced latency and increased throughput by distributing the workload.

Limitations:

- **Complexity**: More complex to design, implement, and manage.
- **Consistency**: Challenges in maintaining data consistency across distributed nodes.
- **Network Dependence**: Performance heavily relies on network reliability and speed.

5

(a) Definitions

i) Interquery Parallelism:

- **Definition**: The ability to execute multiple independent queries simultaneously on different processors. This type of parallelism enhances system throughput by handling several queries at the same time, distributing the workload across multiple processors or nodes.

ii) Intraquery Parallelism:

- **Definition**: The ability to execute a single query in parallel by breaking it down into sub-queries that can be processed concurrently on different processors. This type of parallelism aims to reduce the response time of a single query by leveraging multiple processors.

(b) Range Partitioning Sort vs. Parallel External Sort-Merge Strategy

Range Partitioning Sort:

- **Strategy**:
  - Divide the data into ranges.
  - Each range is sorted independently in parallel.
  - Merge the sorted ranges to produce the final sorted output.
- **Key Characteristics**:
  - Suitable for sorting large datasets that can be naturally divided into ranges.
  - Efficient if ranges are well-distributed and balanced.

Parallel External Sort-Merge:

- **Strategy**:
  - Perform an initial sort on chunks of data that fit into memory (run generation phase).
  - Merge the sorted chunks in parallel (merge phase).
- **Key Characteristics**:
  - Uses external storage (disk) during sorting, making it suitable for very large datasets.
  - The merge phase can be performed in parallel to speed up the final sorting.

Differences:

- **Range Partitioning** focuses on dividing data into specific ranges and sorting each range independently, then merging the sorted ranges.
- **Parallel External Sort-Merge** involves creating sorted runs of data in-memory and then merging these runs in parallel, which is more disk I/O intensive and does not rely on predefined ranges.

(c) Fragment-and-Replicate Join

When to Use:

- **Suitability**: This join strategy is used when joining two large tables that cannot be easily partitioned based on join attributes, particularly when one table is much smaller than the other.
- **Data Distribution**: Ideal for scenarios where the data distribution makes it difficult to colocate related tuples on the same node.

Asymmetric Fragment-and-Replicate Join:

- **Definition**: A variant where only one of the tables (typically the smaller one) is fragmented and replicated across all nodes, while the larger table remains partitioned.
- **Use Case**: Useful when there is a significant size difference between the tables to avoid the overhead of replicating the larger table.

(d) Network Partition in Distributed Databases

Definition:

- **Network Partition**: A situation in a distributed database where the network is split into isolated segments, preventing nodes in different segments from communicating with each other. This can lead to inconsistency and availability issues in the database system.

Handling Network Partitions:

- **Partition Tolerance (CAP Theorem)**: Systems must choose between consistency and availability during a partition:
    - **Consistency**: Ensure all nodes see the same data at the cost of availability.
    - **Availability**: Ensure that the system continues to operate, potentially serving outdated data.
- **Strategies**:
    - **Consistency over Availability**: Use strong consistency models and deny operations that cannot guarantee consistency (e.g., waiting for partition resolution).
    - **Availability over Consistency**: Allow operations to continue, accepting that some nodes might have stale data, and resolve inconsistencies once the partition is resolved (e.g., eventual consistency).
- **Mechanisms**:
    - **Quorum-Based Approaches**: Require a majority of nodes to agree on updates, ensuring consistency during partition resolution.
    - **Conflict Resolution**: Use conflict detection and resolution strategies post-partition to reconcile divergent data states.
    - **Partition Detection and Handling**: Employ mechanisms to detect partitions and apply appropriate policies (e.g., split-brain detection and resolution).

(a) Differences Between Object-Relational Database (ORDBMS) and Object-Oriented Database (OODBMS)

Object-Relational Database (ORDBMS):

- **Data Model**: Extends the relational model to include object-oriented features.
- **Schema**: Tables can have rows and columns but also support complex data types, nested tables, and user-defined types (UDTs).
- **Query Language**: SQL is extended with object-oriented features (e.g., object methods, inheritance).
- **Integration**: Combines relational database capabilities with limited object-oriented features.
- **Usage**: Suitable for applications that primarily use relational data but require some object-oriented capabilities.

Object-Oriented Database (OODBMS):

- **Data Model**: Fully based on object-oriented principles.
- **Schema**: Stores objects directly, supporting complex objects, inheritance, polymorphism, and encapsulation.
- **Query Language**: Often uses object query languages (OQL) that allow for querying objects directly.
- **Integration**: Provides full support for object-oriented programming languages and their features.
- **Usage**: Ideal for applications that heavily rely on complex objects and object-oriented programming, such as CAD/CAM, multimedia, and engineering applications.

(b) Complex Data Types in SQL Extensions

Extensions to SQL handle complex data types that the usual relational model does not support. These include:

1. **Array**:

    - **Example**: An employee might have multiple phone numbers stored in an array.

```
CREATE TABLE Employee (

        EmployeeID INT,

        Name VARCHAR(100),

        PhoneNumbers VARCHAR(15)[]

);
```

2. Nested Tables:

- **Example**: A department might have a nested table of employees.

```
CREATE TYPE EmployeeType AS OBJECT (

        EmployeeID INT,

        Name VARCHAR(100)

);


CREATE TYPE EmployeeTable AS TABLE OF EmployeeType;


CREATE TABLE Department (

        DepartmentID INT,

        DepartmentName VARCHAR(100),

        Employees EmployeeTable

) NESTED TABLE Employees STORE AS NestedEmployees;
```

3.**User-Defined Types (UDTs)**:

- **Example**: Creating a complex type for an address.

```
CREATE TYPE AddressType AS OBJECT (
        Street VARCHAR(100),
        City VARCHAR(50),
        ZipCode VARCHAR(10)
);


CREATE TABLE Customer (
        CustomerID INT,
        Name VARCHAR(100),
        Address AddressType
);
```

(c) Nesting and Unnesting

Nesting:

- **Definition**: Refers to the inclusion of complex data structures within a single column or attribute, such as arrays, nested tables, or user-defined types.
- **Example**: A nested table of phone numbers within an employee record.

```
CREATE TABLE Employee (
    EmployeeID INT,

    Name VARCHAR(100),

    PhoneNumbers VARCHAR(15)[]

);
```

Unnesting:

- **Definition**: The process of flattening nested structures into a simpler, flat relational format.
- **Example**: Converting nested phone numbers into separate rows.

```
SELECT EmployeeID, Name, PhoneNumber

FROM Employee, UNNEST(PhoneNumbers) AS PhoneNumber;
```

(d) Persistent Programming Language vs. Embedded SQL

Persistent Programming Language:

- **Definition**: A programming language that natively supports the persistence of objects, allowing objects to outlive the program execution.
- **Characteristics**:
  - Objects can be stored and retrieved from a database without explicit database commands.
  - Examples include languages with built-in database support like Java with Java Persistence API (JPA) or C++ with ObjectStore.
  - Persistence is managed by the language runtime.

Embedded SQL:

- **Definition**: The integration of SQL statements within a general-purpose programming language, typically using predefined libraries or APIs.
- **Characteristics**:
    - SQL commands are embedded directly within the host language code.
    - Requires explicit database connection and SQL command execution.
    - Examples include using SQL within C, C++, Java, or Python through APIs like JDBC, ODBC, or SQLite.
    - Provides a clear separation between application logic and database operations.

Differences:

- **Integration**: Persistent programming languages have integrated persistence mechanisms, while embedded SQL requires explicit SQL command execution.
- **Ease of Use**: Persistent languages simplify object persistence, reducing boilerplate code. Embedded SQL involves more explicit database management.
- **Flexibility**: Embedded SQL offers more direct control over SQL execution and optimization, while persistent languages abstract many details for simplicity.

Perform Patient ⋈ Doctor using the merge join algorithm when relations are not sorted according to ID (consider the cost of the last write operation for sorting).
Compare the cost of the merge join algorithm with the nested and block nested-loop join algorithms.
Assume: M = 4 blocks, tS = 5 msec. and tT = 0.2 msec.

(b) With examples, show the differences among schedules, serial schedules, and serializable schedules.

(c) Consider the below-given schedule S. Proof and check whether it is conflict serializable or not? Also, check for the view serializability of the schedule.

Transaction 1	Transaction 2
Read (A) A:= A - 50 Write (A)	

	Read(B)
	B:= B - B * 0.1
	Write (B)

Read (B) B:= B + 50 Write (B)	
	Read (A) A:= A + A * 0.1 Write (A)

4 (a) 'Throughput' and 'response time' are two important performance measures of a database. How parallel systems help to improve these measures with respect to query processing?

(b) Explain the factors that affect speedup and scaleup in parallel databases.

(c) Compare 'shared memory' and 'shared disk' as parallel database architecture. [illegible]

(d) What are the benefits and limitations of using distributed database?

5 (a) Define the terms: i) interquery parallelism ii) intraquery parallelism.

(b) Describe the strategy for Range Partitioning Sort. How it differs from Parallel External Sort-Merge strategy?

(c) When should we use Fragment-and-Replicate join? What is Asymmetric Fragment-and-Replicate join?

(d) What is network partition in distributed databases? How it can be handled?

6 (a) What are the purposes of using a data warehouse? Describe benefits and drawbacks of a source-driven architecture for gathering of data at a data warehouse, as compared to a destination-driven architecture.

(b) What do you understand by OLAP? Distinguish between the OLAP operation:
i) Slicing and dicing
ii) cube and rollup

(c) How do you define data mining? Briefly mention some of the applications of data mining.

7 (a) What are the differences between object-relational database and object-oriented database?

(b) What different complex data types are handled by the extensions to SQL that usual relational model does not support. Explain with a suitable example.

(c) What do you understand by 'nesting' and 'unnesting'?

(d) What is persistent programming language? How it differs from embedded SQL?

1 (a) What factors should be considered while choosing RAID levels? Compare RAID level 0, 1 and 5 with respect to these factors.

(b) Consider the deletion of record 5 from the file of **Fig 1**. Compare the relative merits of the following techniques for implementing the deletion:

stop

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 11	98345	Kim	Elec. Eng.	80000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000

Fig. 1

i) Move record 6 to the space occupied by record 5, and move record 7 to the space occupied by record 6 and so on.
ii) Move record 10 to the space occupied by record 5.
iii) Mark record 5 as deleted, and move no records.

How free list can help to solve the problem? Explain with figures.

(c) What is metadata? How it helps manipulating a database?

2 (a) Consider the relation given below and the table X. Construct a B+ tree with N = 4, for indexing the table entries to perform the following query efficiently:

Select * from X where Account Number = "AD0200001"

Account Number	Customer Name	Branch Name	Balance (Million)
AD0200019	ABC	C.DU	5
AD0200027	ABD	T.DU	7
AD0200001	ABB	N.K.D	7
AD0200021	ABE	RB.DU	9
AD0200025	ABBCD	C.DU	3
AD0200003	ABER	T.DU	5
AD0200009	YAJ	T.DU	6
AD0200008	KAL	RB.DU	77
AD0200007	PLL	C.DU	89
AD0200010	BPL	C.DU	90
AD0200018	IPL	C.DU	23
AD0200015	MPL	N.K.D	24
AD0200013	MMPL	RB.DU	54
AD0200030	XPL	N.K.D	57

(b) Analyze the computational complexity of performing a query in a B+ tree with the node size N. Calculate the cost of performing the query stated in 2(a) with respect to the number of nodes accessed in a B+ tree during the query. [5

3 (a) Let relations Patient (ID, Name, Address, Age) and Doctor (ID, Dept, Salary, Degree) have the following properties [7]
The patient has 50000 tuples and needs 800 blocks in the secondary storage.
The doctor has 20000 tuples and needs 500 blocks in the secondary storage.