DBMS Question Bank

Q: Explain decomposition using multivalued dependencies.

Keywords: Decomposition, Multivalued Dependency (MVD)

Meaning: How can we break down a table using multivalued dependencies to remove data redundancy?

- ✓Answer:
  - Multivalued Dependency (MVD) means one attribute determines a set of values independently of another attribute.
  - If a table has MVDs, it can lead to unnecessary data repetition.
  - Decomposition breaks the table into smaller relations
  - Each new relation contains only the related attributes, reducing redundancy.
  - Original relation can be recreated using natural joins.Example Inner, outer, left and light join
  - This process ensures data consistency and avoids anomalies.

Q: Explain in detail the design issues in the E-R model. How do extended E-R features help overcome some of these issues?

Keywords: Design issues, E-R model, Extended E-R features

Design issues are the challenges or problems that arise during the process of creating and structuring a database.

The Entity-Relationship (ER) model and Extended Entity-Relationship (EER) model are both conceptual data models used in database design.

The ER model provides a basic way to represent entities and their relationships, while the

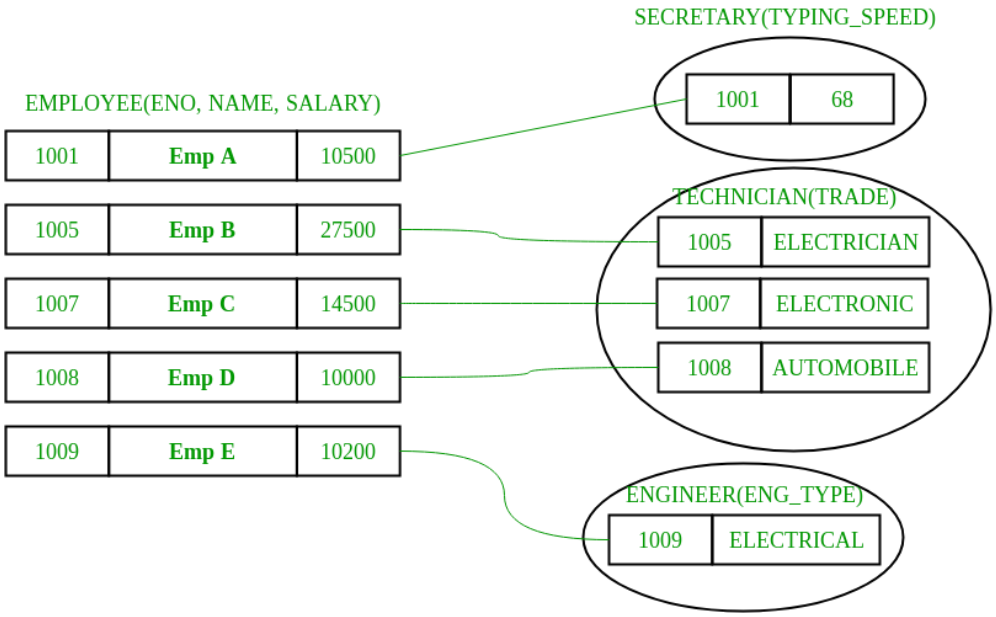
EER model extends the ER model with more features to handle more complex data structures.

Meaning: What are the problems when designing ER models, and how do advanced features help solve them?

- Answer: Design issues in database design are:

- Choosing between attributes vs entity sets can be confusing.
- Deciding the type of relationship may be unclear.
- Identifying correct primary keys is a challenge.
- Handling inheritance or classification is not always easy.

- Extended E-R features help:
  - -Generalization helps group similar entities (e.g., Student and Teacher into Person).
  - Specialization allows splitting a general entity into specific ones (e.g., Person into Student and Teacher).
  - Aggregation models relationships between relationships.
  - Categorization helps when a subclass belongs to more than one superclass.
  - - These improve clarity and support complex requirements.



Q: Describe weak entity sets with example and their representation in E-R diagrams.

Keywords: Weak entity, E-R diagram, Example

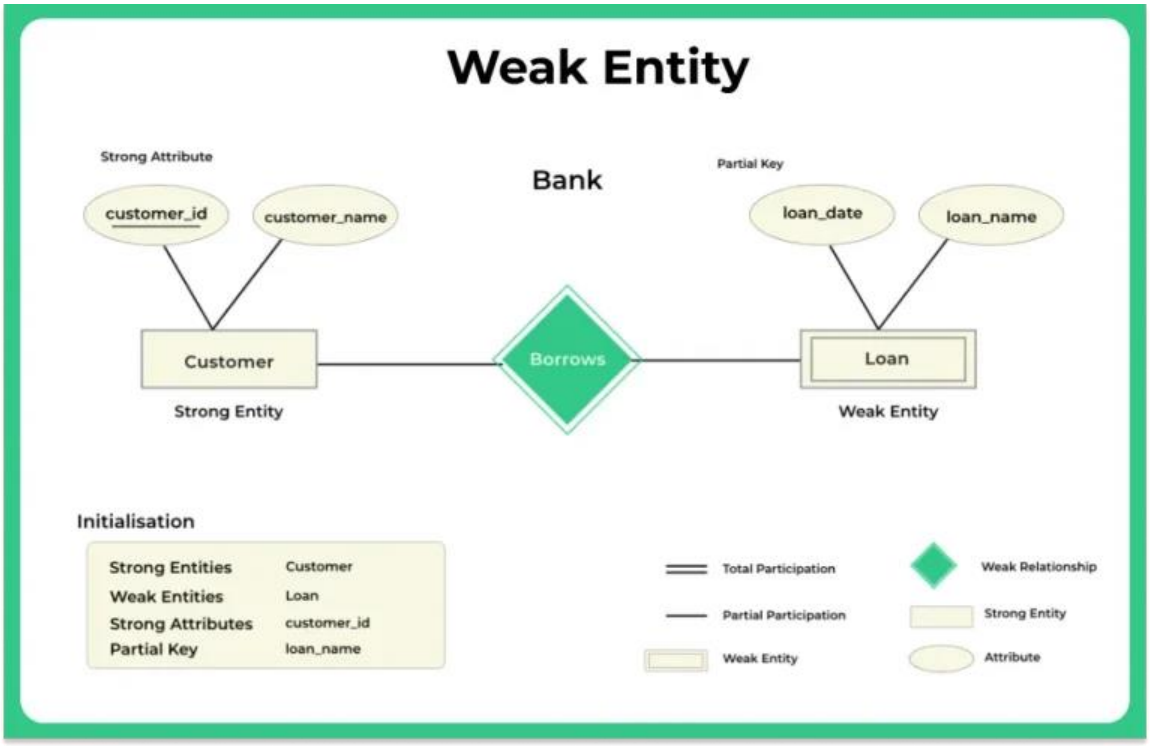
Meaning: What are weak entities, how are they shown in ER diagrams, and give a simple example.

- ✓Answer:

  - A weak entity cannot be uniquely identified by its attributes alone.
  - It depends on a strong entity for its identification.
  - It has a partial key and a total participation in the identifying relationship.
  - In ER diagrams:
    - Weak entities are shown using double rectangles.

- Their identifying relationship is shown using a double diamond.
- Partial keys are underlined with dashed lines.

- Example: 'Dependent' is a weak entity of 'Employee'. Each dependent depends on the employee for unique identification.



NENDA DEEP: https://prepinsta.com/dbms/weak-entity-and-strong-entity/

Q: Explain specialization and generalization with diagrams and examples. How are they useful in database modeling?

Keywords: Specialization, Generalization, Database modeling

Meaning: What do we mean by generalization and specialization in databases, and how do they help?

- Answer:

  - Generalization combines similar entities into one broad entity (e.g., Car and Truck into Vehicle).
  - Specialization splits one broad entity into more specific ones (e.g., Employee into Manager and Engineer).
  - These concepts support inheritance in data design.
  - They simplify complex data structures.
  - They help avoid redundancy by sharing common attributes in one place.

In diagrams:

- - A triangle labeled with ISA is used to show generalization/specialization.
- They improve clarity in ER modeling by showing real-world hierarchy.

Q: Describe the features of a good relational database design.

Keywords: Relational database, Design features

Meaning: What are the key features that make a relational database design effective and reliable?

A relational database is a system that organizes and stores data in a structured manner, using tables (also called relations) to represent relationships between different data points.

- Answer:
  - Minimal redundancy: No unnecessary repetition of data.
  - Data integrity: Rules are enforced to maintain accuracy and consistency.
  - Normalization: Tables are properly normalized to reduce anomalies.
  - Efficient access: Queries can run fast with indexes and optimized schema.
  - Clear primary and foreign keys: Ensures proper relationships between tables.
  - Scalability: Easy to modify or expand when needed.
  - Security: Data access is properly controlled.

Q: What is normalization? Explain all its types with suitable examples.

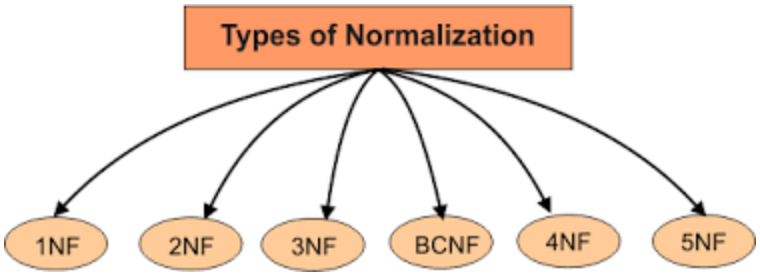
Keywords: Normalization, 1NF, 2NF, 3NF, BCNF, 4NF, 5NF

Meaning: What is normalization and how do we improve table design step by step?

- ✓Answer:
  - Normalization is the process of organizing data to reduce redundancy and improve data integrity.
  - Types of Normal Forms:
    - 1NF: Remove repeating groups. Each cell should have atomic values.
    - 2NF: Remove partial dependencies. Every non-key attribute should depend on the whole primary key. https://www.datacamp.com/tutorial/second-normal-form
    - 3NF: Remove transitive dependencies. Non-key attributes should depend only on the primary key. https://www.datacamp.com/tutorial/third-normal-form
    - BCNF: A stronger version of 3NF. Every determinant must be a candidate key.
    - 4NF: Remove multivalued dependencies.

- 5NF: Remove join dependencies.

- Example: Splitting a table with multiple phone numbers into separate tables with proper keys.



NENDA DEE : https://takeuforward.org/dbms/normalisation-and-its-types

Q: Discuss functional dependency theory and its role in relational design.

Keywords: Functional Dependency, Relational Design

Meaning: What is functional dependency and how does it help in designing relational databases?

- ✓Answer:

  - Functional dependency (FD) means one attribute uniquely determines another (A → B).
  - Functional dependency (FD) describes a relationship between two or more attributes (columns) within a table. It means that one attribute or a set of attributes uniquely determines the value of another attribute or set of attributes. If you know the value of the "determinant" (the attribute that determines), you can uniquely determine the value of the "dependent" (the attribute being determined).
  - FDs help in identifying candidate keys and ensuring data consistency.
  - They are used to detect anomalies and suggest better table structure.
  - FDs are key to applying normalization rules (2NF, 3NF, BCNF).
  - Helps in decomposition of relations while preserving data.
  - Makes the design less redundant and more structured.

Q: Explain decomposition using functional dependencies. How does it help in achieving normalization?

Keywords: Decomposition, Functional Dependency, Normalization

Meaning: How do we use functional dependencies to split tables and make them more efficient?

- ✓Answer:

- Decomposition breaks a relation into smaller relations using FDs.
  - This helps remove partial, transitive, or multivalued dependencies.
  - Ensures each relation is in a higher normal form like 3NF or BCNF.
  - Reduces data anomalies and improves consistency.
  - Preserves data without loss (lossless decomposition).
- Example: A relation with StudentID → Name, Course split into Student(StudentID, Name) and Enrollment(StudentID, Course).

Q: What is multivalued dependency? How do we use it for decomposition? Give an example.

Keywords: Multivalued Dependency, Decomposition, Example

Meaning: What is a multivalued dependency and how do we handle it in database design?

- ✓Answer:

  - MVD exists when one attribute determines multiple values of another attribute independently of other attributes.
    - Occurs when a table has two or more independent multi-valued facts about an entity.
  - Decomposition breaks the table into two to avoid repetition.
    - Used to achieve 4NF (Fourth Normal Form).
    - Preserves all data and maintains dependencies.
  - Example: Student with multiple phone numbers and multiple hobbies → create two tables: StudentPhones and StudentHobbies.

Q: Explain the different types of physical storage media used in databases. How do they vary in terms of speed, cost, and capacity?

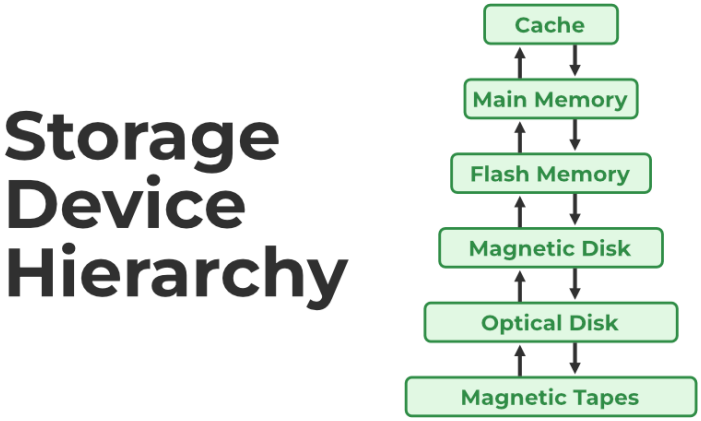
Keywords: Physical Storage Media, Speed, Cost, Capacity

Meaning: What are the main types of hardware used for storing databases and how do they compare?

- Answer:
- Physical storage media refers to the tangible materials used to store data, like hard drives, solid-state drives, or optical discs

  - Cache: Very fast, very expensive, small size.
  - Main Memory (RAM): Fast, costly, volatile (data lost on power off).
  - Solid State Drives (SSD): Fast access, more expensive than HDDs, no moving parts.
  - Hard Disk Drives (HDD): Cheaper, slower than SSD, large storage.
  - Optical Discs (CD/DVD): Cheap, slower, limited storage, mostly for backup.

- Magnetic Tapes: Very cheap, very slow, used for long-term backups.



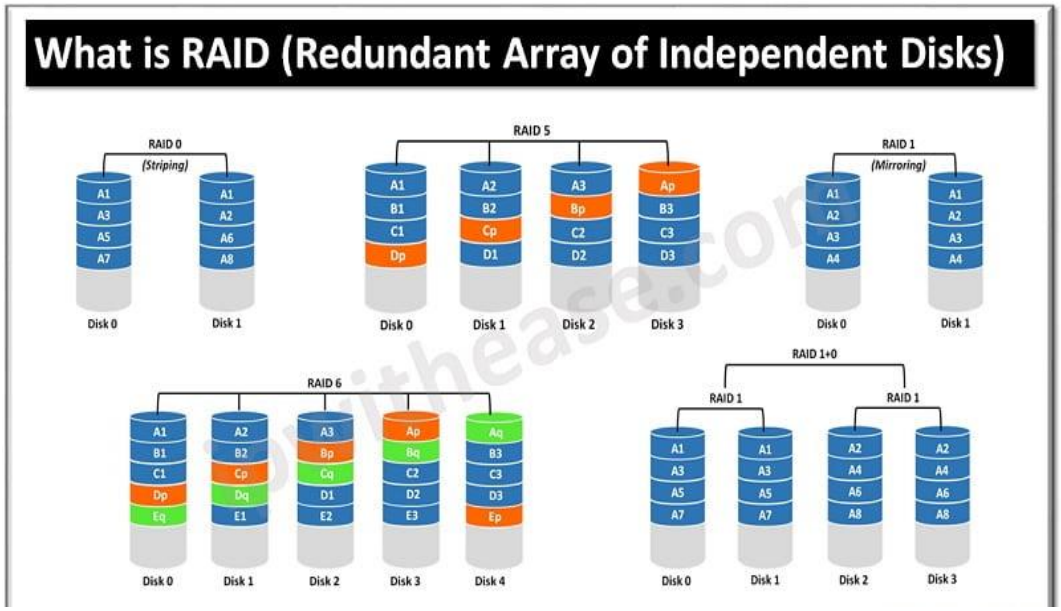
Q: Describe the various RAID levels and their advantages and disadvantages.

Keywords: RAID, Redundancy, Performance

Meaning: What are the different RAID methods for storing data and what are their pros and cons?

RAID (redundant array of independent disks) is a way of storing the same data in different places on multiple hard disks or solid-state drives (SSDs) to protect data in the case of a drive failure

- ✓Answer:
  - RAID 0: Stripes data, improves speed, no redundancy.
  - RAID 1: Mirrors data, good redundancy, uses more space.
  - RAID 5: Data + parity, balances speed and fault tolerance.
  - RAID 6: Double parity, survives 2 disk failures, slower writes.
  - RAID 10: Combines RAID 1 and 0, high performance and redundancy, costly.
- Trade-off: Higher RAID level means better fault tolerance but more cost and complexity.



Q: Discuss the different types of file organization in databases. What are the advantages and disadvantages of each?

Keywords: File Organization, Sequential, Indexed, Hashed

meaning: How is data physically stored in files and what are the pros and cons of each method?

File Organization refers to the the way in which data is stored in a file and the method(s) by which it can be accessed.

- Answer:
  - Sequential: Data stored in sorted order. Easy for batch processing, slow for random access.
  - Heap (Unordered): Fast insertion, slow searching.
  - Hashed: Fast for exact searches, not good for range queries.
  - Clustered: Related records stored close, fast access for groups.
  - Indexed: Uses indexes for faster searches, needs extra space.
- Each has use cases depending on access pattern and performance goals.

Q13: Compare fixed-length and variable-length records in terms of their structure, storage efficiency, and access time.

Keywords: Fixed-length, Variable-length, Record Storage, Performance

Meaning: How do these two types of records differ in size, efficiency, and speed?
Answer:

Feature	Fixed-Length Records	Variable-Length Records
Structure	Uniform size per record	Size varies per record
Access Time	Fast (direct addressable)	Slower (requires pointer traversal)
Storage Efficiency	Can waste space	Better for sparse or varying data
Complexity	Simple to implement	Needs delimiters or offset tables
Use Case	Banking, logs	Email systems, document storage

Q14: Discuss different record organization methods and their impact on performance.

Keywords: Record organization, Heap, Sequential, Hashed, Clustered

Meaning: What are the formats for storing records and how do they affect performance?
Answer:
Diagram: Record Organization Techniques

```
Heap       → [ R3 ][ R1 ][ R4 ] (unordered)
Sequential → [ R1 ][ R2 ][ R3 ] (sorted)
Hashed     → Bucket[3] → [ R4, R8 ]
Clustered  → Pages store physically related rows
Indexed    → Index → [pointer] → Data page
```

- **Heap**: Best for frequent inserts, poor for search.
- **Sequential**: Great for reporting, inefficient for random updates.
- **Hashed**: Very fast equality search, no range support.
- **Clustered**: Optimized for range queries and joins.
- **Indexed**: Balanced performance, supports fast lookups.
-

Q15: Explain the concept of indexing in databases and compare primary and secondary indexing.

Keywords: Indexing, Primary index, Secondary index

Meaning: How does indexing accelerate searches, and how do primary and secondary types differ?

Indexing in a database is the process of creating data structures (called **indexes**) that improve the speed of data retrieval operations on a database table, at the cost of additional storage and slower write operations.

An **index** is a data structure (like B-tree or hash table) that allows the database to find rows faster without scanning the entire table.

✅ Answer:
Definition: An index is a data structure that improves retrieval speed.

Index Type	Built On	Allows Duplicates	Ordering Required	Purpose
Primary Index	Primary Key (Unique)	❌	Yes	Fast direct access
Secondary Index	Non-key/Non-ordered field	✅	No	Conditional/multi-field access

Use primary index for key-based lookups, secondary for searching multiple columns.

Q16: Describe the structure and advantages of B+ Trees in database indexing.

Keywords: B+ Tree, Leaf-level data, Balanced search tree
Meaning: What makes B+ Trees powerful in indexing large datasets?
Answer:

A B+ tree is a specialized tree data structure designed for efficient data storage and retrieval, particularly in databases and file systems.

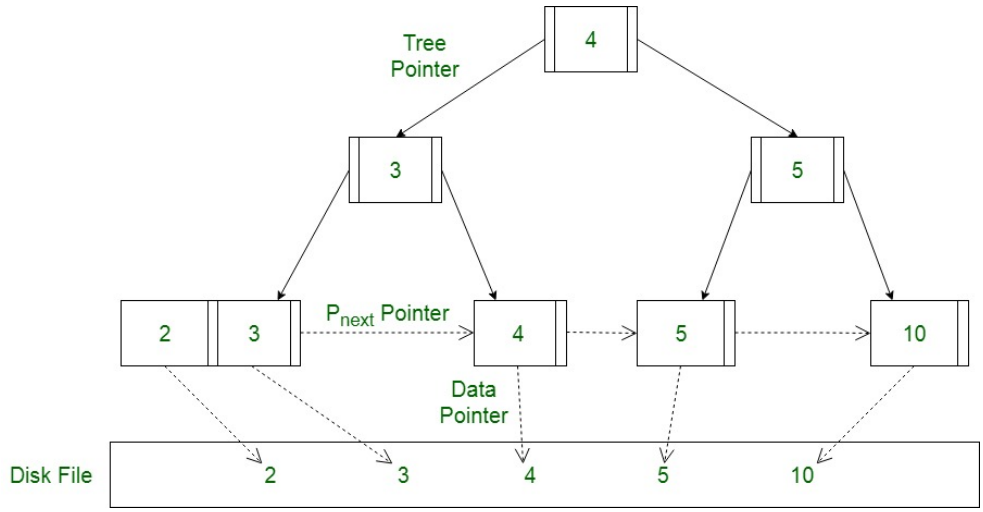
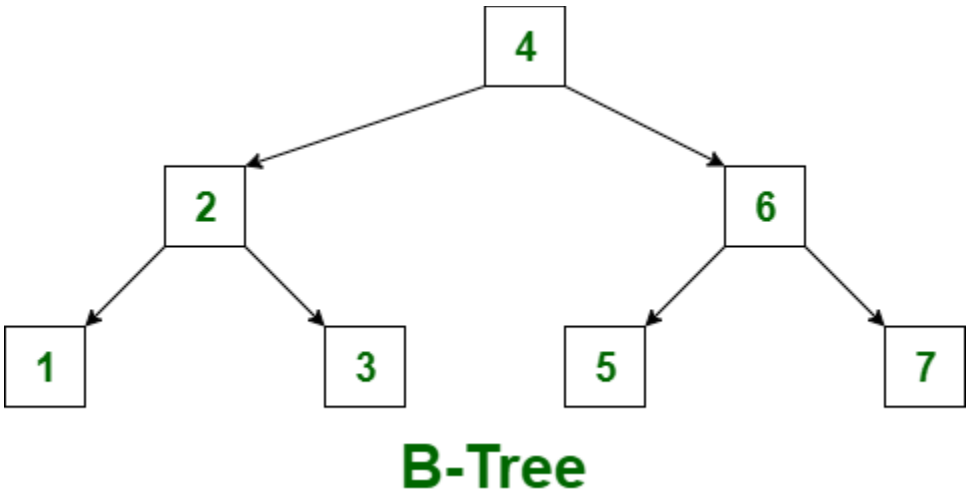


Illustration:

```
      [ 30 | 60 ]
     /     |     \
[10,20] [35,45] [70,90]  ← leaf nodes (contain all data)
     ↘  ↘  ↘ (linked)
```

- All keys in leaf nodes.
- Internal nodes store routing info only.
- Supports fast sequential and range access.
- Disk-friendly due to block size optimization.

Q17: Explain the structure and operations of B-Trees. How are B-Trees used in indexing?
Keywords: B-Tree, Balanced tree, Multi-level index.



Meaning: What is a B-Tree and how does it facilitate indexing operations?
Answer:
Explanation:

- B-Tree stores keys and pointers in all nodes.
- Height-balanced, allows logarithmic insert/search/delete.
- Maintains sorted order.
- Nodes split when full → dynamic rebalancing.
- Suitable for both range and point queries.

Basis of Comparison	B tree	B+ tree
Pointers	All internal and leaf nodes have data pointers	Only leaf nodes have data pointers
Search	Since all keys are not available at leaf, search often takes more time.	All keys are at leaf nodes, hence search is faster and more accurate.
Redundant Keys	No duplicate of keys is maintained in the tree.	Duplicate of keys are maintained and all nodes are present at the leaf.
Insertion	Insertion takes more time and it is not predictable sometimes.	Insertion is easier and the results are always the same.
Deletion	Deletion of the internal node is very complex and the tree has to undergo a lot of transformations.	Deletion of any node is easy because all node are found at leaf.
Leaf Nodes	Leaf nodes are not stored as structural linked list.	Leaf nodes are stored as structural linked list.
Access	Sequential access to nodes is not possible	Sequential access is possible just like linked list
Height	For a particular number nodes height is larger	Height is lesser than B tree for the same number of nodes
Application	B-Trees used in Databases, Search engines	B+ Trees used in Multilevel Indexing, Database indexing
Number of Nodes	Number of nodes at any intermediary level 'l' is 2^l.	Each intermediary node can have n/2 to n children.

Q18: Explain static hashing and its limitations in handling dynamic data.

Keywords: Static hashing, Buckets, Overflow
Meaning: What is static hashing and where does it fail with growing data?
Answer:
Diagram:

Static hashing is a hashing technique where the number of primary buckets is fixed at the time the hash table is created. A hash function maps keys to these fixed buckets.

Limitations in handling dynamic data:

1. **Fixed Size**: Cannot grow or shrink automatically as data size changes.
2. **Overflow Handling**: Excess data in a bucket leads to overflow chains or overflow blocks, which slow down search performance.
3. **Wasted Space**: If many buckets are underused, it leads to inefficient memory usage.
4. **Poor Scalability**: Not suitable for applications with frequent insertions and deletions or unpredictable data volume.

Q19: Explain dynamic hashing and how it resolves the limitations of static hashing.

Keywords: Dynamic hashing, Extendible hashing, Scalability

Meaning: How does dynamic hashing adapt to growing datasets?

Answer:

ChatGPT said:

Dynamic hashing is a technique where the hash table can grow or shrink dynamically based on the number of records. It adjusts the number of buckets and possibly the hash function as data changes.

How it resolves limitations of static hashing:

1. **Handles Growth**: It expands gracefully as more data is inserted.
2. **Avoids Long Overflow Chains**: Buckets are split as needed, reducing overflow.
3. **Efficient Space Use**: Only needed buckets and directory entries are created.
4. **Better Performance**: Maintains fast access time even with dynamic datasets.

Q20: Explain the query processing steps in a database system. What are the common measures used to estimate query cost?
Keywords: Query processing, Cost model, Evaluation plan
Meaning: How does a DBMS process a query and decide the best execution strategy?
✅ Answer:
Steps in Query Processing:

1. **Parsing** → Check syntax, create parse tree.
2. **Translation** → Convert to relational algebra.
3. **Optimization** → Generate multiple plans, choose cheapest.
4. **Execution** → Evaluate using chosen plan.

Query Processing Steps in a Database System

1. **Parsing and Translation**
   - The SQL query is checked for syntax and semantics.
   - It's then translated into an internal form (often a relational algebra expression).
2. **Query Optimization**
   - The system generates alternative execution plans.
   - It selects the most efficient plan using a cost-based optimizer.
3. **Query Execution Plan Generation**
   - The chosen plan is translated into a sequence of operations (like scans, joins, sorts).
4. **Query Execution**
   - The database engine executes the plan and retrieves the results.

Common Measures Used to Estimate Query Cost

1. **I/O Cost**
   - Number of disk reads/writes (major cost factor in large datasets).
2. **CPU Cost**
   - Processing time for operations (e.g., comparisons, hash computations).
3. **Communication Cost**
   - Relevant in distributed databases; involves data transfer between nodes.
4. **Memory Usage**
   - Space required for intermediate results or buffers.

Q21: Describe the selection operation in query processing and the methods used to improve its efficiency.

Answer:
Selection is the operation in a database query that filters records based on a given condition. It plays a key role in reducing the size of data processed in subsequent steps of query execution.

Techniques to Optimize Selection:

- **Indexing**: Use of B+ Trees or bitmap indexes allows faster lookup.
- **Predicate Pushdown**: Apply filter conditions as early as possible in the query plan.
- **Binary Search**: Efficient for sorted data files.
- **Partitioning**: Data is divided into segments, and only relevant partitions are scanned.
- **Statistics**: Used by the optimizer to choose the most efficient execution plan.

Q22: What is sorting, and why is it important in query processing? Describe an external sorting algorithm.

Answer: Sorting arranges data in a specified order, crucial for operations like `ORDER BY`, `GROUP BY`, and merge joins.

External Merge Sort Steps:

1. Divide input file into manageable-sized chunks.
2. Sort each chunk and store on disk as **sorted runs**.
3. Merge sorted runs using **multi-way merge** and buffers.
4. Produce the final sorted output.

Why Important:

- Required for producing ordered results.
- Prepares data for merge join.
- Handles datasets larger than main memory efficiently.

Q23: Explain different types of join operations and their performance.

Answer:
Join operations combine rows from two or more tables based on related columns.

Join Type	Best Use Case	Performance	Index Required
Nested Loop	Small tables or outer joins	Slow (O(n*m))	No
Index Nested Loop	One table indexed	Fast	Yes

Join Type	Best Use Case	Performance	Index Required
Sort-Merge Join	Inputs already sorted	Medium	No
Hash Join	Large, unsorted datasets	Fast	No
Cross Join	Cartesian product (rarely used)	Very slow	No

Q24: How are relational expressions evaluated? What are the factors influencing the choice of evaluation plan?

Answer:
Relational expressions represent queries in algebraic form, optimized by the query processor. They describe the logical structure of a query and serve as a blueprint for generating efficient execution plans that transform inputs into outputs in the least costly manner.

Evaluation Steps:

1. Parse query and convert into a tree form.
2. Apply algebraic transformation rules.
3. Generate multiple equivalent logical plans.
4. Select the best physical plan using:
    - **Cardinality estimation**: Predicts output size.
    - **Index usage**: Plans using indexes are often preferred.
    - **Cost model**: Considers CPU, I/O, memory cost.

Q25: Explain the Transaction Concept, its States, and Implementation in DBMS.

Answer:
A transaction is a logical unit of work that must be either fully completed or fully rolled back.

Transaction States:

- **Active**: Transaction is being executed.
- **Partially Committed**: Execution complete, awaiting commit.
- **Committed**: Changes are saved permanently.
- **Aborted**: Changes are rolled back due to failure.

ACID Properties:

- **Atomicity**: All or nothing.
- **Consistency**: Database remains in a valid state.
- **Isolation**: Concurrent transactions appear isolated.

- **Durability**: Committed changes persist.

Implementation:

- Uses **write-ahead logging (WAL)**.
- Employs locking mechanisms for concurrency.

Q26: Define Serializability and Explain Concurrency Control with Lock-based Protocols.

Answer:
Serializability ensures that concurrent transactions produce the same result as some serial (one-after-another) execution.

Two-Phase Locking (2PL):

- **Growing Phase**: Acquire all required locks.
- **Shrinking Phase**: Release locks.

Lock Types:

- **Shared Lock (S)**: Read access.
- **Exclusive Lock (X)**: Write access.

Deadlock Handling:

- **Detection**: Wait-for graph.
- **Prevention**: Timeout or order-based locking.

Q27: Explain Recovery System, Failure Classification, and Storage Structure.

Answer:
Failures and recovery are managed to maintain the consistency and durability of transactions.

Failure Classifications:

- **Transaction Failure**: Logical errors, invalid inputs.
- **System Failure**: Crash or power outage.
- **Media Failure**: Disk crash or corruption.

Storage Structure:

Type	Volatile	Persistent	Examples
RAM	Yes	No	Cache, Buffers
Disk	No	Yes	HDD, SSD
Stable Storage	No	Yes	RAID, Cloud Backup

Recovery Tools:

- Logs and checkpoints for system and transaction recovery.

Q28: Discuss Log-based Recovery, Buffer Management, and Recovery with Concurrent Transactions.

Answer:
Log-based recovery maintains consistency by storing all changes to a durable log.

Components:

- **Log Entries**: Record before and after image of data.
- **Write-Ahead Logging (WAL)**: Changes are logged before being applied.
- **Checkpointing**: Periodic snapshots to speed up recovery.
- **Buffer Management**: Handles in-memory data pages efficiently.

Concurrent Recovery:

- Ensures **recoverability** and **isolation** across transactions.

Q29: Explain Failure with Loss of Non-Volatile Storage and Role of Remote Backup Systems.

Answer:
Loss of non-volatile storage requires robust backup strategies for full recovery.

Backup Types:

- **Full Backup**: Entire database snapshot.
- **Incremental Backup**: Only changes since last backup.
- **Differential Backup**: All changes since last full backup.

Recovery Strategy:

- Use **remote or cloud-based storage**.

- Combine **logs** and **backups** to restore full state.
- Maintain **high availability** and disaster recovery readiness.

13. Compare fixed-length and variable-length records in terms of their structure, storage

Structure	Fiaed-Langth Records	Variable-Length Records
Structure	Uniform size	Varying length
Storage Efficiency	Simple to manage	Uses space efficienlly, with potential gaps
Access Time	Fast, direct access	Slower due to varied sizes

14. Impacts different record organizaation methods

Impact on Performance

Indexing

- Creating data structure for fast access
- Primary Indexing. Index on a key attr.
- Secondary Indexing. Index on a hori-key

A d-r Trae

- Primary Indexing. Fast accesby key-limited to key search
- Secondary Indexing. Allows searching by attributes than key

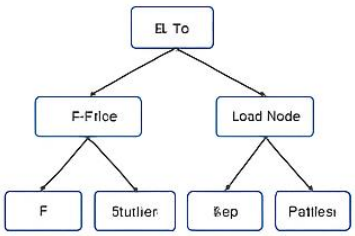
16. Static hashing and its limitations in handling oy/ingen_eladq Steu ne Hashiing

Static Hashing

- Using a fixedsize ible of buckets

Limitations

- Fixed size, connot grow or thrink
- Poor handling of dynamic data changes
- Leads to overflow or linderuse



21. Explain different types of join operations droring imitations of static hashing

Selection Operation

- Choosing rowsbased a condition

Efficiancy Methods

- Use indexing on th selection attribute.
- Apply selection early to reduce ta volume
- Use hash-based techniques
- Optimiize selection predicars for better performance

	Types	Performance
Parsing	Parsing	High
Left outer	Fixed size or out grow or shiink	Affects performate due to sults
Right outer	Poor handling of dynams data things	Usually slewer, returnt alrrows
Full outer	Run optmiize on sonted order	Usually slewer, returns allrows

22. Sorting, and why is it importanit ni imitations of query processing

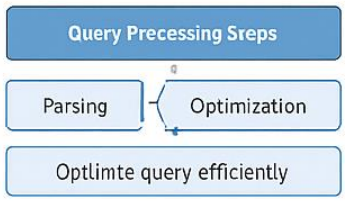
Sorting

- Sortting records by attribute values

Importance Facilitate efficient query processinkage

- Addressing overflow to avoid rehashing
- Better handung of growing data
- Use a hash-based technidues for be resiz

Commor: Measures



23. What is sorting, and why is it imporant on the imitations of static hashing?

Importance

Types	Performance