DBMS Interview Questions

1. Basics of DBMS

1. **What is DBMS? Why use it?** A **DBMS (Database Management System)** is software that allows users to create, define, manage, and query a database. Think of it as an organized digital filing cabinet, but with powerful tools to search, sort, and update your files. We use it to provide a structured way to store and retrieve data, ensuring data integrity, security, and concurrent access for multiple users.
    - **Example**: Instead of keeping student records in a bunch of disorganized Excel sheets (a manual system), a university uses a DBMS to store student data (name, ID, courses) in a structured manner. This makes it easy to find a student's record, enroll them in a new course, or generate a report of all students in a specific major.
2. **Difference between DBMS and RDBMS?** A **DBMS** is the foundational system for managing databases. An **RDBMS (Relational Database Management System)** is a specific type of DBMS based on the relational model. RDBMS stores data in tables with rows and columns and uses relationships between these tables. All RDBMS are DBMS, but not all DBMS are RDBMS.
    - **DBMS**: Can be any database system, including hierarchical, network, or NoSQL databases. Does not necessarily use a table-based structure or enforce relationships.
    - **RDBMS**: Strictly uses tables (relations) to store data. It enforces relationships and constraints like primary keys and foreign keys. Examples include MySQL, PostgreSQL, Oracle, and SQL Server.
3. **What are the advantages and disadvantages of DBMS?**
    - **Advantages**:
        - **Data Redundancy Control**: It helps avoid duplicate data, saving storage space and improving consistency.
        - **Data Integrity**: It enforces rules (constraints) to ensure the data is accurate and consistent.
        - **Data Security**: It provides mechanisms for access control, so only authorized users can access specific data.
        - **Concurrent Access**: Multiple users can access and modify data at the same time without interfering with each other.
        - **Backup and Recovery**: It has built-in tools to back up data and restore it in case of a system failure.
    - **Disadvantages**:
        - **Complexity**: DBMS software is complex and requires skilled professionals to manage.
        - **Cost**: High-end DBMS can be expensive, both in terms of software and hardware.
        - **Performance Overhead**: The system's features like security and concurrency control can add performance overhead.

        - **Size**: DBMS software requires a significant amount of disk space and memory.
4. **What is a database schema?** A **database schema** is the **logical design** of a database. It defines the structure of the database, including the tables, fields (columns), data types, and the relationships between them. It's the blueprint or skeleton of the database. A schema doesn't contain any data itself; it just defines what the data looks like.
    - **Example**: A `Library` database schema might define three tables: `Books` (with columns for `ISBN`, `Title`, `Author`), `Members` (with columns for `MemberID`, `Name`), and `Borrows` (linking `MemberID` and `ISBN`).
5. **What is a database instance?** A **database instance** is the **live, running state** of the database at a specific moment in time. It includes the database data in memory, the running processes, and the background processes that manage the database. While the schema is the static design, the instance is the dynamic, active snapshot of the data.
    - **Example**: When a user adds a new book to the `Books` table in the `Library` database, that change is part of the current database instance. The schema remains the same, but the data within it has changed.
6. **What are keys in DBMS? (Primary, Foreign, Candidate, Super, Alternate)** A **key** is an attribute or a set of attributes that uniquely identifies a row (or tuple) in a table. They are essential for defining relationships and enforcing data integrity.
    - **Super Key**: A set of one or more attributes that can uniquely identify a row in a table. It's a superset of all other keys. Every super key can be a candidate key, but not every candidate key is a super key.
        - **Example**: In a `Students` table with columns `(StudentID, RollNo, Name, Email)`, the super keys could be `(StudentID)`, `(RollNo)`, `(StudentID, Name)`, or `(RollNo, Email)`.
    - **Candidate Key**: A **minimal** super key. This means if you remove any attribute from it, it will no longer be unique.
        - **Example**: `(StudentID)` and `(RollNo)` are candidate keys in the `Students` table because they are unique, and you can't remove any attribute from them to make them smaller.
    - **Primary Key**: A **chosen candidate key** used to uniquely identify each row in a table. A table can have only one primary key. It cannot be `NULL` and must be unique.
        - **Example**: We'd likely choose `StudentID` as the primary key for the `Students` table.
    - **Alternate Key**: Any candidate key that is **not chosen** as the primary key.
        - **Example**: If `StudentID` is the primary key, then `RollNo` would be an alternate key.
    - **Foreign Key**: A column or a set of columns that establishes a link between the data in two tables. It references the primary key of another table. It's used to enforce referential integrity.

        - **Example**: In a `Courses` table, a `StudentID` column would be a foreign key referencing the `StudentID` primary key in the `Students` table, linking a student to their enrolled courses.
7. **Difference between primary key and unique key?**
    - **Primary Key**:
        - Identifies each row uniquely.
        - **Can't be `NULL`**.
        - There can be only **one** primary key per table.
        - Used for defining relationships with foreign keys.
    - **Unique Key**:
        - Ensures that all values in the column are unique.
        - **Can contain `NULL` values**.
        - A table can have **multiple** unique keys.
        - Not used for relationships as often as primary keys.
8. **What is a composite key?** A **composite key** is a primary key that consists of **two or more attributes** (columns) to uniquely identify a row. It's used when a single column isn't enough to guarantee uniqueness.
    - **Example**: In a `Grades` table with columns `(StudentID, CourseID, Grade)`, neither `StudentID` nor `CourseID` alone is unique (a student can take multiple courses, and a course can have multiple students). The **composite key** is `(StudentID, CourseID)`, as a specific student's grade for a specific course is unique.
9. **What is a surrogate key?** A **surrogate key** is an artificial or synthetic key, typically a single column with a system-generated unique number (like an `INT` or `BIGINT` with an `AUTO_INCREMENT` property), used to uniquely identify each row. It has no meaning to the user or the business and is used purely for database purposes. It's often used when a natural key is complex, large, or subject to change.
    - **Example**: Instead of using a composite key `(FirstName, LastName, DateOfBirth)` for a `Customers` table, which is long and could have duplicates, you would create a `CustomerID` column that auto-increments for each new customer. This `CustomerID` is the surrogate key.
10. **What is normalization in DBMS?Normalization** is the process of organizing data in a database to reduce data redundancy and improve data integrity. It involves breaking down large tables into smaller, related tables and defining relationships between them. The goal is to eliminate anomalies (insertion, update, and deletion anomalies) and ensure data is stored logically.
    - **Example**: Imagine a single `Employees` table with columns `(EmployeeID, EmployeeName, DeptID, DeptName, DeptLocation)`. If you update the `DeptName` for a department, you have to do it for every employee in that department, which is an **update anomaly**. Normalization would split this into two tables: `Employees (EmployeeID, EmployeeName, DeptID)` and `Departments (DeptID, DeptName, DeptLocation)`. Now, a department's name only needs to be updated in one place.

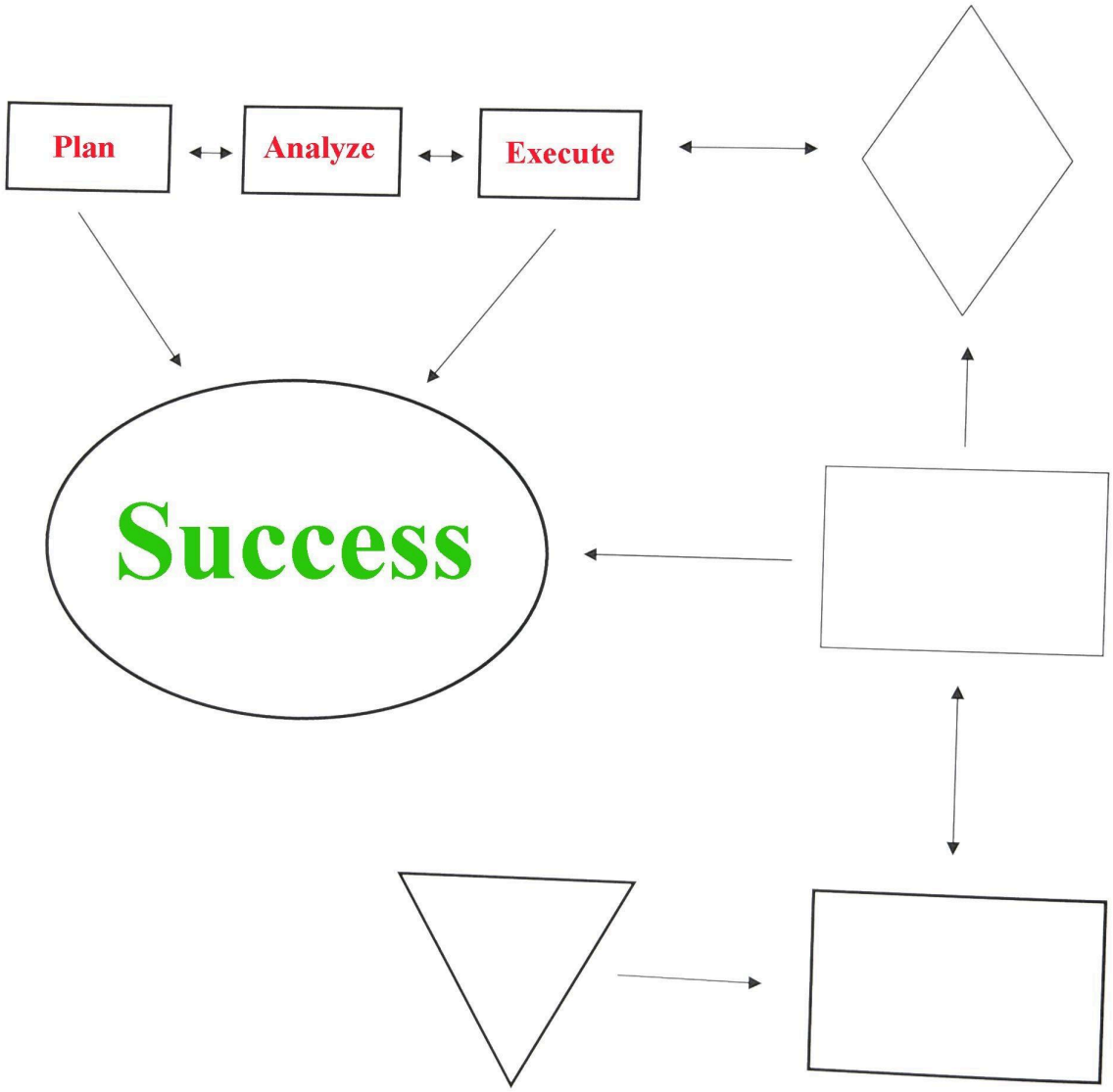
---

2. Normalization & Design

11. **What are the different normal forms (1NF, 2NF, 3NF, BCNF, 4NF, 5NF)?**
    Normalization is a set of rules for database design. The normal forms are a progression of these rules, with each level building on the previous one.
    - **First Normal Form (1NF)**: A table is in 1NF if:
        - Each column contains atomic (single) values. There are no repeating groups or lists.
        - Each column has a unique name.
        - The order of data doesn't matter.
    - **Second Normal Form (2NF)**: A table is in 2NF if it is in 1NF and all non-key attributes are **fully functionally dependent** on the primary key. This means no non-key attribute is dependent on only a **part** of a composite primary key.
    - **Third Normal Form (3NF)**: A table is in 3NF if it is in 2NF and there are **no transitive dependencies**. A transitive dependency occurs when a non-key attribute is dependent on another non-key attribute.
    - **Boyce-Codd Normal Form (BCNF)**: A stricter version of 3NF. A table is in BCNF if for every functional dependency `X -> Y`, `X` is a **super key**. This handles cases where a non-key attribute determines a part of the candidate key.
    - **Fourth Normal Form (4NF)**: A table is in 4NF if it is in BCNF and has no **multi-valued dependencies**.
    - **Fifth Normal Form (5NF)**: A table is in 5NF if it is in 4NF and has no **join dependencies**. This is rarely used in practice.
12. **What is denormalization? When to use it?Denormalization** is the process of intentionally adding redundant data to one or more tables. It's the inverse of normalization and is used to improve read performance by reducing the number of joins required to retrieve data.
    - **When to use it**:
        - When dealing with **reporting and analytics** (OLAP) where read performance is critical.
        - When queries involve a large number of joins, making them slow.
        - When the data is primarily read-only and updates are infrequent.
    - **Example**: In a `Products` table, if you frequently need the `CategoryName` for a product, you might store the `CategoryName` directly in the `Products` table instead of having to join with a separate `Categories` table.
13. **Difference between normalization and denormalization?**
    - **Normalization**:
        - **Goal**: Reduce data redundancy and improve data integrity.
        - **Process**: Splits large tables into smaller ones.
        - **Performance**: Improves write/update performance but may slow down read performance due to more joins.
        - **Use Case**: OLTP systems (transaction-heavy).
    - **Denormalization**:
        - **Goal**: Improve read performance by adding redundant data.
        - **Process**: Combines data from different tables into one.

        - **Performance**: Improves read performance but can degrade write/update performance and introduce redundancy.
        - **Use Case**: OLAP systems (read-heavy, reporting).
14. **What is functional dependency?** A **functional dependency** is a constraint between two attributes or sets of attributes in a table. It states that if you know the value of one attribute (or set of attributes), you can determine the value of another attribute. It's written as `A -> B`, meaning `A` functionally determines `B`.
    - **Example**: In a `Students` table, `StudentID -> StudentName` is a functional dependency. If you know the `StudentID`, you can uniquely determine the `StudentName`.
15. **What is transitive dependency?** A **transitive dependency** occurs when a non-key attribute is functionally dependent on another non-key attribute. It violates 3NF. It's an indirect dependency.
    - **Example**: Consider a `Projects` table with `(EmployeeID, EmployeeName, DeptID, DeptName)`.
        - `EmployeeID -> DeptID` (functional dependency)
        - `DeptID -> DeptName` (functional dependency)
        - `EmployeeID -> DeptName` (transitive dependency)
    - Here, `DeptName` is transitively dependent on `EmployeeID` through `DeptID`. To fix this, you would move `DeptID` and `DeptName` to a separate `Departments` table.
16. **What is a database index?** A **database index** is a data structure (like a B-Tree) that improves the speed of data retrieval operations on a table. It's like an index in a book. Instead of scanning every page to find a topic, you look at the index to find the exact page number. In a database, it allows the system to quickly locate data without scanning the entire table.
17. **Types of indexes (clustered vs non-clustered)?**
    - **Clustered Index**:
        - Defines the **physical order** of the data in the table.
        - The table's rows are sorted based on the clustered index key.
        - A table can have only **one** clustered index.
        - Access is fast because the data is physically organized.
        - **Example**: The primary key on a table is often a clustered index.
    - **Non-Clustered Index**:
        - Does **not** define the physical order of the data.
        - It's a separate structure that contains the index key and a pointer to the physical location of the data row.
        - A table can have **multiple** non-clustered indexes.
        - **Example**: An index on a `LastName` column in a `Customers` table.
18. **What is a view in DBMS?** A **view** is a virtual table based on the result set of an SQL query. It doesn't store data itself; it's a saved query. When you query a view, the underlying query is executed, and the results are returned. Views are used to simplify complex queries, enforce security (by restricting access to certain columns), and provide a layer of abstraction.

    - **Example**: A `CurrentStudents` view could be created from the `Students` table to only show students with a `Status` of 'Active', hiding past students.
19. **Difference between materialized view and normal view?**
    - **Normal View (or Logical View)**:
        - Does **not** store data. It's a saved query.
        - Data is retrieved from the base tables **every time** the view is accessed.
        - Data is always **up-to-the-minute** accurate.
        - Performance is dependent on the complexity of the underlying query.
    - **Materialized View (or Snapshot View)**:
        - **Stores the data** from the query's result set.
        - Data is a **snapshot** of the base tables at a specific point in time.
        - Needs to be periodically refreshed to get the latest data.
        - Provides **faster query performance** because the data is pre-computed and stored.
        - Used for data warehousing and reporting.
20. **What is ER model (Entity-Relationship model)?** An **ER (Entity-Relationship) model** is a high-level conceptual data model that visually represents the relationships between entities in a database.



Licensed by Google

It's a powerful tool for designing a database. * **Entity**: A real-world object or concept (e.g., `Student`, `Course`). Represented by a rectangle. * **Attribute**: A property of an entity (e.g., `StudentName`, `CourseID`). Represented by an oval. * **Relationship**: An association between two or more entities (e.g., a `Student` `enrolls` in a `Course`). Represented by a diamond.

* **Example**: An ER model for a school might show an `Entity` `Students` and an `Entity` `Courses` connected by a `Relationship` `Enrolls`.

---

3. SQL & Queries

21. **What is DDL, DML, DCL, and TCL in SQL?** These are the four sublanguages of SQL.
    - **DDL (Data Definition Language)**: Used to **define** or modify the structure of the database.
        - **Commands**: `CREATE`, `ALTER`, `DROP`, `TRUNCATE`.
        - **Example**: `CREATE TABLE Students (ID INT, Name VARCHAR(255));`
    - **DML (Data Manipulation Language)**: Used to **manage** data within schema objects.
        - **Commands**: `INSERT`, `UPDATE`, `DELETE`, `SELECT`.
        - **Example**: `INSERT INTO Students (ID, Name) VALUES (1, 'Alice');`
    - **DCL (Data Control Language)**: Used to **control** access to data.
        - **Commands**: `GRANT`, `REVOKE`.
        - **Example**: `GRANT SELECT ON Students TO user1;`
    - **TCL (Transaction Control Language)**: Used to **manage** transactions.
        - **Commands**: `COMMIT`, `ROLLBACK`, `SAVEPOINT`.
        - **Example**: `COMMIT;`
22. **Difference between DELETE, TRUNCATE, and DROP?**
    - **DELETE**:
        - **DML command**.
        - Removes **specific rows** from a table.
        - You can use a `WHERE` clause to filter.
        - Keeps the table structure.
        - Is a transaction, so it can be `ROLLBACK`'ed.
        - Slower than `TRUNCATE` because it logs each deleted row.
    - **TRUNCATE**:
        - **DDL command**.
        - Removes **all rows** from a table.
        - **Can't** use a `WHERE` clause.
        - Keeps the table structure.
        - **Can't** be `ROLLBACK`'ed in most databases.
        - Faster than `DELETE` because it de-allocates the data pages and doesn't log individual row deletions.
    - **DROP**:
        - **DDL command**.
        - Removes the **entire table**, including its structure (schema), indexes, and constraints.
        - **Can't** be `ROLLBACK`'ed.
23. **What is the difference between HAVING and WHERE clause?**
    - **WHERE clause**:

        - Filters individual **rows** before they are grouped.
        - Used with `SELECT`, `UPDATE`, `DELETE`.
        - Can't contain aggregate functions (like `SUM`, `AVG`, `COUNT`).
    - **HAVING clause**:
        - Filters **groups** of rows after they have been created by the `GROUP BY` clause.
        - Used with `GROUP BY`.
        - **Can** contain aggregate functions.

Example:
SQL
-- WHERE: find employees who earn more than $50,000
SELECT Name, Salary FROM Employees WHERE Salary > 50000;

-- HAVING: find departments where the average salary is more than $60,000
SELECT Dept, AVG(Salary) FROM Employees GROUP BY Dept HAVING AVG(Salary) > 60000;

    -

24. **Difference between UNION and UNION ALL?** Both combine the result sets of two or more `SELECT` statements.
    - **`UNION`**:
        - Combines and returns **distinct** rows.
        - Automatically removes duplicate rows.
        - Slower than `UNION ALL` because of the overhead of checking for and removing duplicates.
    - **`UNION ALL`**:
        - Combines and returns **all** rows, including duplicates.
        - Faster because it doesn't perform a duplicate check.
25. **Difference between INNER JOIN, LEFT JOIN, RIGHT JOIN, and FULL JOIN?** These clauses are used to combine rows from two or more tables based on a related column.
    - **`INNER JOIN`**: Returns rows when there is a **match in both tables**. It's the most common type of join.
    - **`LEFT JOIN` (or `LEFT OUTER JOIN`)**: Returns all rows from the **left table**, and the matched rows from the right table. If no match is found, `NULL` values are returned for the columns from the right table.
    - **`RIGHT JOIN` (or `RIGHT OUTER JOIN`)**: Returns all rows from the **right table**, and the matched rows from the left table. If no match is found, `NULL` values are returned for the columns from the left table.
    - **`FULL JOIN` (or `FULL OUTER JOIN`)**: Returns all rows when there is a match in **either** the left or the right table. `NULL` values are returned for non-matched columns.
26. **What are subqueries? Types of subqueries?** A **subquery** (or nested query) is a query nested inside another SQL statement. It's often used to return data that will be used in the main query.

- **Types of subqueries**:
  - **Single-row subquery**: Returns one row. Used with single-value operators like `=`, `>`, `<`.
  - **Multi-row subquery**: Returns multiple rows. Used with operators like `IN`, `ANY`, `ALL`.
  - **Multi-column subquery**: Returns multiple columns.
  - **Correlated subquery**: A subquery that depends on the outer query for its values and is re-evaluated for each row of the outer query.

27. **Difference between correlated and non-correlated subquery?**
    - **Non-correlated Subquery**:
      - Executes **independently** of the outer query.
      - The results of the inner query are calculated once and then used by the outer query.
      - It's generally more efficient.
    - **Correlated Subquery**:
      - Depends on the outer query and **executes for each row** processed by the outer query.
      - It can be slow on large datasets because of this row-by-row processing.
      - It's used when a value from the outer query is needed to complete the inner query's WHERE clause.
28. **What is a stored procedure? Advantages?** A **stored procedure** is a prepared SQL code block that is stored in the database. You can execute it by calling its name.
    - **Advantages**:
      - **Performance**: The SQL is pre-compiled and optimized, leading to faster execution.
      - **Security**: You can grant users permission to execute a procedure without giving them direct access to the underlying tables.
      - **Reduced Network Traffic**: A single call to a procedure can replace multiple SQL statements.
      - **Reusability**: The same procedure can be called by multiple applications.
29. **What is a trigger in DBMS?** A **trigger** is a special type of stored procedure that is automatically executed in response to certain events on a table or view (e.g., an `INSERT`, `UPDATE`, or `DELETE` statement).
    - **Example**: An `audit_log_trigger` could be set to automatically record a log entry in an `AuditLog` table whenever an `UPDATE` happens on the `Salaries` table.
30. **Difference between procedure and function in DBMS?**
    - **Stored Procedure**:
      - Can return multiple result sets or an output parameter.
      - Can contain `DML` and `DDL` statements.
      - Can't be called from a `SELECT` statement.
      - Used to perform a series of actions.
    - **Function**:
      - Must return a single value (or a table).

    - Generally can only contain `SELECT` statements (it's often a `read-only` operation).
    - Can be called from a `SELECT` statement, `WHERE` clause, or other functions.
    - Used to perform a calculation and return a value.

---

4. Transactions & Concurrency

31. **What is a transaction in DBMS?** A **transaction** is a single logical unit of work that consists of one or more database operations. It must be treated as a whole; either all of its operations are completed successfully (committed), or none of them are (rolled back). This ensures data integrity.
    - **Example**: A money transfer from account A to B is a transaction. It involves two steps: `DEBIT` account A and `CREDIT` account B. If the credit fails, the debit must be undone (rolled back) to ensure the data is consistent.
32. **What are ACID properties?ACID** is an acronym for the four key properties of a reliable database transaction.
    - **Atomicity**: The "all or nothing" rule. A transaction is a single, indivisible unit. Either all operations succeed, or none do.
    - **Consistency**: A transaction must bring the database from one valid state to another. It ensures that the transaction follows all defined rules and constraints (e.g., referential integrity).
    - **Isolation**: Concurrent transactions should not interfere with each other. The result of a set of concurrent transactions should be the same as if they were executed one after another (serially).
    - **Durability**: Once a transaction is committed, its changes are permanent and will survive a system failure, such as a power outage.
33. **What are different types of transactions?** This question is a bit ambiguous, but typically refers to the state or outcome of a transaction:
    - **Active**: The initial state, where the transaction is being executed.
    - **Partially Committed**: After the last statement of the transaction has executed. It's waiting for the `COMMIT` command.
    - **Failed**: The transaction is running, but an error has occurred, and it can no longer proceed.
    - **Aborted**: The transaction has been rolled back due to failure.
    - **Committed**: The transaction has completed successfully and its changes are made permanent.
34. **What are concurrency control problems (Dirty Read, Lost Update, Phantom Read)?** These are common issues that arise when multiple transactions access and modify the same data concurrently without proper control.
    - **Dirty Read**: A transaction reads data that has been modified by another transaction that has not yet been committed. If the uncommitted transaction is rolled back, the first transaction has read "dirty" or incorrect data.

    - **Lost Update**: Two transactions read the same data, and then both update it. The update of the first transaction is overwritten (lost) by the second transaction.
    - **Phantom Read**: A transaction reads a set of rows that satisfy a `WHERE` clause. Another transaction then `INSERT`s or `DELETE`s rows that also satisfy the same clause. When the first transaction re-runs its query, it sees a new "phantom" row.
35. **What is locking in DBMS? Types of locks?Locking** is a mechanism to control concurrent access to data. It prevents other transactions from modifying or reading a resource that is currently being used by another transaction.
    - **Types of Locks**:
        - **Shared Lock (S-Lock)**: Allows multiple transactions to read a resource at the same time. No transaction can write to it.
        - **Exclusive Lock (X-Lock)**: Only one transaction can hold an exclusive lock on a resource at a time. No other transaction can read or write to it.
        - **Update Lock**: Prevents a deadlock situation by acting as a mix between a shared and exclusive lock.
36. **Difference between shared lock and exclusive lock?**
    - **Shared Lock (Read Lock)**:
        - Allows multiple transactions to read the same data concurrently.
        - Prevents any transaction from getting an exclusive lock and modifying the data.
    - **Exclusive Lock (Write Lock)**:
        - Grants exclusive access to a single transaction to modify the data.
        - Prevents any other transaction from getting a shared or exclusive lock.
37. **What is two-phase locking protocol?** The **Two-Phase Locking (2PL)** protocol is a concurrency control method that ensures serializability. It has two phases:
    - **Growing Phase**: The transaction can only acquire locks, but cannot release any.
    - **Shrinking Phase**: The transaction can only release locks, but cannot acquire any new ones.
38. This protocol guarantees that the schedule of transactions is serializable, preventing lost updates and dirty reads.
39. **What is deadlock in DBMS? How to prevent it?** A **deadlock** is a situation where two or more transactions are each waiting for the other to release a lock, and neither can proceed.
    - **Example**: Transaction A holds a lock on table X and wants a lock on table Y. Transaction B holds a lock on table Y and wants a lock on table X. Both are stuck.
    - **How to prevent it**:
        - **Deadlock Prevention**: Pre-ordering locks (always request locks in a specific order).
        - **Deadlock Detection**: The system detects a deadlock (e.g., using a wait-for graph) and then resolves it by aborting one of the transactions.

        - **Deadlock Avoidance**: The system checks resource allocation state to ensure that circular wait conditions never occur.
40. **What is serializability in DBMS?Serializability** is the main correctness criterion for concurrency control in a database. It ensures that the outcome of executing multiple transactions concurrently is the same as if they were executed one after another in some serial order. It is a key goal of transaction isolation.
41. **What is optimistic vs pessimistic concurrency control?**
    - **Pessimistic Concurrency Control**:
        - Assumes that conflicts are **likely** to occur.
        - Uses **locks** to prevent other users from accessing data until the transaction is complete.
        - Works well in high-contention environments but can cause deadlocks and slow down transactions.
    - **Optimistic Concurrency Control**:
        - Assumes that conflicts are **unlikely** to occur.
        - Allows transactions to proceed without locking.
        - Checks for conflicts only at the time of commit. If a conflict is detected, the transaction is rolled back.
        - Works well in low-contention environments and provides better performance but requires rollback mechanisms.

---

5. Indexing & Performance

41. **What is indexing in DBMS?Indexing** is a technique used to improve the performance of a database. It involves creating a data structure, like a B-Tree or hash table, that holds a small portion of the table's data and provides a quick lookup. The goal is to minimize the number of disk accesses required to find a particular row.
42. **Difference between B-Tree and B+ Tree indexing?**
    - **B-Tree**:
        - A tree data structure where each node can have multiple children.
        - Stores both keys and data pointers in **internal and leaf nodes**.
        - Data can be retrieved from any node, but this can lead to slower performance for range queries.
    - **B+ Tree**:
        - An improvement on the B-Tree.
        - Stores keys in internal nodes but stores **data pointers only in the leaf nodes**.
        - Leaf nodes are linked together in a sequential list. This makes **range queries** much more efficient because you can traverse the linked list.
        - This is the more common indexing structure in modern databases.
43. **What is hashing in DBMS?Hashing** is an indexing technique where the address of the data record is determined by a **hash function** applied to the key. It's excellent for **exact-match queries** because it provides a direct lookup. However, it's not suitable for range queries (e.g., `WHERE Price BETWEEN 10 AND 20`) because the data is not stored in a sorted order.

44. **What is clustered vs non-clustered indexing?**
    - **Clustered Index**:
        - **Physically orders** the data rows in the table.
        - There can be only one clustered index per table.
        - Faster for retrieving data from a specific range.
    - **Non-Clustered Index**:
        - Does **not** physically order the data.
        - Creates a separate index structure with pointers to the data rows.
        - You can have multiple non-clustered indexes.
45. **What is covering index?** A **covering index** (or a query-specific index) is a non-clustered index that includes all the columns needed to satisfy a query. When a query can be answered entirely by looking at the index, the database doesn't need to access the main table, which significantly speeds up performance.
    - **Example**: For a query `SELECT Name, Email FROM Employees WHERE ID = 123;`, a covering index on `(ID)` that includes `Name` and `Email` would be `CREATE INDEX idx_id_name_email ON Employees (ID) INCLUDE (Name, Email);`.
46. **What is composite index?** A **composite index** (or concatenated index) is an index on **multiple columns** of a table. It's useful for queries that frequently filter or sort by a combination of columns. The order of columns in a composite index matters.
    - **Example**: `CREATE INDEX idx_lastname_firstname ON Employees (LastName, FirstName);`. This index will be used for queries that filter by `(LastName)` or `(LastName, FirstName)` but not just `(FirstName)`.
47. **What is index fragmentation?Index fragmentation** occurs when the logical order of the index pages doesn't match their physical order on the disk. This happens over time as data is inserted, updated, and deleted. Fragmentation increases the time it takes to scan the index, as the disk head has to jump around to read the pages. You fix it by rebuilding or reorganizing the index.
48. **How to optimize SQL queries?**
    - **Use indexes wisely**: Create indexes on columns used in `WHERE`, `JOIN`, and `ORDER BY` clauses.
    - **Avoid `SELECT *`**: Only select the columns you need.
    - **Use `JOIN`s correctly**: Use `INNER JOIN` when possible and `EXISTS` instead of `IN` for subqueries.
    - **Avoid functions in `WHERE` clauses**: `WHERE YEAR(OrderDate) = 2023` prevents index usage. Instead, use a range: `WHERE OrderDate BETWEEN '2023-01-01' AND '2023-12-31'`.
    - **Denormalize for read-heavy workloads**: Reduce the number of joins.
    - **Use `EXPLAIN PLAN`**: Analyze the query execution plan to understand how the database is running your query.
49. **What is query execution plan?** A **query execution plan** (or query plan) is a sequence of steps that a database system uses to execute a SQL statement. It shows how the database will retrieve the data, including which indexes to use, how to perform joins, and in what order to execute operations. It's a crucial tool for **performance tuning**.

50. **What is sharding in databases?Sharding** is a method of horizontal partitioning, which involves dividing a large database into smaller, more manageable pieces called **shards**. Each shard is a separate database running on a separate server. This technique is used to scale a database horizontally to handle massive amounts of data and high traffic loads.
    - **Example**: A social media platform might shard its `Users` table by `country`. All users from the USA go to one shard, users from Europe go to another, and so on.

---

6. Advanced DBMS Concepts

51. **Difference between OLTP and OLAP?**
    - **OLTP (Online Transaction Processing)**:
        - **Purpose**: Handles day-to-day transactional data (e.g., e-commerce orders, bank transfers).
        - **Workload**: High volume of short, atomic transactions (`INSERT`, `UPDATE`, `DELETE`).
        - **Schema**: Normalized schema (3NF) to reduce redundancy.
        - **Performance**: Optimized for fast writes.
        - **Example**: A bank's ATM system, an airline booking system.
    - **OLAP (Online Analytical Processing)**:
        - **Purpose**: Supports business intelligence, reporting, and complex analysis.
        - **Workload**: Low volume of complex, read-heavy queries.
        - **Schema**: Denormalized schema (e.g., star schema) to speed up reads.
        - **Performance**: Optimized for fast reads and aggregations.
        - **Example**: A sales forecasting system, a market trend analysis tool.
52. **What is data warehouse?** A **data warehouse** is a central repository of integrated data from one or more disparate sources. It stores current and historical data in a single place for reporting and analysis. Data is typically extracted from OLTP databases, transformed to be consistent, and loaded into the data warehouse using an **ETL (Extract, Transform, Load)** process. It's the primary system for OLAP.
53. **What is star schema and snowflake schema?** These are popular models for data warehouses.
    - **Star Schema**:
        - Has a central **fact table** surrounded by multiple **dimension tables**.
        - The dimension tables are not normalized (denormalized).
        - Simple design, requires fewer joins, and is faster for queries.
    - **Snowflake Schema**:
        - An extension of the star schema where dimension tables are **normalized**.
        - This reduces data redundancy but increases the number of joins required for queries.
        - More complex to design and manage.

54. **What is fact table and dimension table?**
    - **Fact Table**: Contains the **facts** or measurements of a business process (e.g., `sales_amount`, `order_quantity`). It also contains foreign keys to the dimension tables. Fact tables are usually very large.
    - **Dimension Table**: Contains descriptive attributes related to the facts (e.g., `product_name`, `customer_city`, `date`). They provide context for the facts.
55. **What is database partitioning?Database partitioning** is the process of dividing a large table into smaller, more manageable parts. These parts, or partitions, can be stored in different locations (e.g., different disks). This can improve performance by allowing the database to query only the relevant partition, and it simplifies maintenance tasks.
    - **Types**:
        - **Horizontal Partitioning (Sharding)**: Dividing a table's rows into multiple tables.
        - **Vertical Partitioning**: Dividing a table's columns into multiple tables.
56. **What is CAP theorem?** The **CAP theorem** states that a distributed database system can only satisfy **two out of three** of the following guarantees:
    - **Consistency**: All nodes see the same data at the same time.
    - **Availability**: Every request receives a response, without a guarantee that it's the most recent write.
    - **Partition Tolerance**: The system continues to operate even if there is a network partition (communication failure) between nodes.
57. In a distributed system, you must choose between consistency and availability in the event of a network partition.
58. **Difference between vertical and horizontal partitioning?**
    - **Vertical Partitioning**:
        - Splits a table by **columns**.
        - Creates a new table for a subset of the original columns.
        - Used to improve performance for queries that only need a few columns from a very wide table.
    - **Horizontal Partitioning (Sharding)**:
        - Splits a table by **rows**.
        - Creates new tables with a subset of the original rows.
        - Used to handle massive amounts of data by distributing the load across multiple servers.
59. **What is replication in DBMS?Replication** is the process of creating and maintaining multiple copies of a database. It's used to:
    - **Improve availability**: If one database server fails, others can take over.
    - **Increase performance**: Read-heavy applications can distribute queries among multiple replica servers.
    - **Provide fault tolerance** and disaster recovery.
60. **Difference between master-slave and peer-to-peer replication?**
    - **Master-Slave Replication**:
        - One server is designated as the **master** (or primary). All writes happen on the master.
        - Other servers are **slaves** (or replicas). They receive copies of the data from the master.

        - Slaves are typically used for read operations.
        - Simple to set up but has a single point of failure (the master).
    - **Peer-to-Peer Replication (or Multi-Master)**:
        - All servers can be both a master and a slave.
        - Writes can be performed on any node.
        - This provides higher availability but is more complex to manage and can lead to **write conflicts**.
61. **What is database clustering?Database clustering** is the process of linking multiple database servers together so they act as a single system. This is done to achieve **high availability** and **load balancing**. If one server in the cluster fails, another takes over automatically (failover), ensuring continuous operation. It can also distribute the workload among multiple nodes to improve performance.

---

7. NoSQL vs RDBMS

61. **What is NoSQL? Difference from SQL databases?NoSQL** (Not Only SQL) databases are non-relational databases that provide flexible schemas and are designed for large volumes of data and high scalability.
    - **SQL (RDBMS)**:
        - **Schema**: **Rigid**, pre-defined schema (schema-on-write).
        - **Structure**: Stores data in tables with rows and columns.
        - **Scalability**: Primarily scales **vertically** (upgrading a single server), though horizontal scaling is possible.
        - **Transactions**: Follows **ACID** properties.
        - **Use Case**: When data structure is stable and consistency is critical.
    - **NoSQL**:
        - **Schema**: **Dynamic**, flexible schema (schema-on-read).
        - **Structure**: Stores data in key-value pairs, documents, wide columns, or graphs.
        - **Scalability**: Primarily scales **horizontally** (adding more servers).
        - **Transactions**: Often follows **BASE** properties, prioritizing availability over consistency.
        - **Use Case**: When dealing with large, unstructured data and high-velocity reads/writes.
62. **Types of NoSQL databases (Key-Value, Document, Column, Graph)?**
    - **Key-Value**: Stores data as a simple key-value pair. Fast for lookup but not for complex queries.
        - **Example**: Redis, Memcached.
    - **Document**: Stores data in flexible, semi-structured documents (like JSON or XML). A document can have different fields from another.
        - **Example**: MongoDB, CouchDB.
    - **Column (or Wide-Column)**: Stores data in columns rather than rows. Good for high-volume reads and writes of a few columns.
        - **Example**: Cassandra, HBase.

    - **Graph**: Stores data as nodes (entities) and edges (relationships) to represent complex relationships.
        - **Example**: Neo4j, ArangoDB.
63. **When to use SQL vs NoSQL?**
    - **Use SQL when...**:
        - Your data is structured and doesn't change frequently.
        - Data integrity and consistency are a top priority (ACID compliance).
        - You need complex queries with joins and aggregations.
        - The system requires a strict schema.
    - **Use NoSQL when...**:
        - Your data is unstructured or semi-structured.
        - You need to handle massive amounts of data and high-velocity traffic.
        - Your application requires extreme horizontal scalability.
        - Availability and partition tolerance are more important than immediate consistency.
64. **Difference between MongoDB and MySQL?**
    - **MongoDB**:
        - A **NoSQL, document-oriented** database.
        - Uses **JSON-like documents** for a flexible, schema-less structure.
        - Scales **horizontally** via sharding.
        - Best for unstructured data, real-time analytics, and content management.
    - **MySQL**:
        - A **relational** database (**RDBMS**).
        - Uses **tables** with a rigid, pre-defined schema.
        - Primarily scales **vertically**.
        - Best for structured data, transactional systems, and where referential integrity is crucial.
65. **What is BASE property in NoSQL?BASE** is a set of properties often associated with NoSQL databases, which prioritize availability and eventual consistency over the strict consistency of ACID.
    - **Basically Available**: The system is guaranteed to be available for queries.
    - **Soft state**: The state of the system can change over time even without input due to eventual consistency.
    - **Eventually Consistent**: The system will eventually become consistent once all data has been replicated and propagated throughout the system.
66. **Difference between consistency, availability, and partition tolerance (CAP theorem)?**
    - **Consistency**: A read operation on any node always returns the most recent write. All nodes in the system agree on the state of the data at the same time.
    - **Availability**: Every request to a non-failing node must receive a response, without exception. The system is always operational and responsive.
    - **Partition Tolerance**: The system continues to function even if some nodes can't communicate with others due to a network partition. This is a fundamental requirement for any distributed system.
67. The CAP theorem states that you can only have two of these three.
68. **What are advantages and disadvantages of NoSQL?**
    - **Advantages**:

        - **Scalability**: Easily scales horizontally to handle huge data volumes and traffic.
        - **Flexibility**: Supports semi-structured and unstructured data with flexible schemas.
        - **Performance**: Highly optimized for specific data models and use cases, often with high read/write speeds.
    - **Disadvantages**:
        - **Consistency**: Often provides "eventual consistency," which can be a problem for applications requiring strict data integrity (like banking).
        - **Querying**: Lack of a standardized query language (like SQL) and complex joins are difficult or impossible.
        - **Maturity**: Ecosystems and tools are less mature than RDBMS.
69. **What is sharding in MongoDB?Sharding** in MongoDB is a method for distributing data across multiple machines. It allows for horizontal scaling to handle large data sets and high throughput operations. MongoDB uses a **shard key** to determine how data is distributed across the shards in a cluster. A `mongos` router directs queries to the correct shard.
70. **Difference between schema-less and schema-based DB?**
    - **Schema-based (RDBMS)**:
        - Requires a **pre-defined, rigid schema** before you can insert data.
        - All rows in a table must conform to the same structure.
        - Ensures data integrity and consistency.
        - **Schema-on-Write**: The schema is enforced at the time data is written.
    - **Schema-less (NoSQL)**:
        - **No pre-defined schema**. Each document or record can have a different structure.
        - Allows for great flexibility and agility in development.
        - **Schema-on-Read**: The schema is defined by the application or query at the time of reading the data.
71. **What is polyglot persistence?Polyglot persistence** is the practice of using multiple types of database technologies within a single application. You choose the best database for a specific task.
    - **Example**: An e-commerce application might use a **relational database** for core transactional data (orders, users), a **document database** for the product catalog (which has a flexible structure), and a **graph database** for a "people who bought this also bought..." recommendation engine.

---

8. Security & Backup

71. **What are database security measures?**
    - **Authentication**: Verifying the identity of a user.
    - **Authorization**: Granting specific permissions to authenticated users (e.g., `SELECT` on a table but not `UPDATE`).
    - **Encryption**: Encrypting data at rest (on disk) and in transit (over the network).

- **Auditing**: Tracking and logging all database activity to detect suspicious behavior.
- **Data Masking**: Hiding or obfuscating sensitive data from unauthorized users.
- **Least Privilege Principle**: Giving users the minimum amount of access necessary to do their job.

72. **Difference between authentication and authorization in DBMS?**
    - **Authentication**: **Who are you?** The process of verifying a user's identity, typically using a username and password.
        - **Example**: A user logging in with `username = 'admin'` and `password = 'password123'`.
    - **Authorization**: **What are you allowed to do?** The process of granting or denying a user access to specific database resources or operations.
        - **Example**: After logging in, the `admin` user is granted `GRANT ALL` permissions on a specific database, while a `guest` user is only allowed `SELECT` on a public table.
73. **What is SQL injection? How to prevent it?SQL Injection** is a code injection technique where an attacker exploits vulnerabilities in an application's database layer by inserting malicious SQL code into input fields. This can allow the attacker to read, modify, or delete data they are not supposed to access.
    - **Example**: A login form where the user enters `' or 1=1 --`. The resulting query becomes `SELECT * FROM Users WHERE username = '' or 1=1 --' AND password = '...`, which always returns true and bypasses authentication.
    - **How to prevent it**:
        - **Prepared Statements with Parameterized Queries**: The most effective method. The query is pre-compiled, and the user's input is treated as a literal value, not as executable code.
        - **Input Validation**: Sanitize and validate all user input.
        - **Principle of Least Privilege**: The database user account used by the application should have limited permissions.
74. **What is database encryption?Database encryption** is the process of converting data into an unreadable format using an encryption key. It's a key security measure that protects sensitive data from being read by unauthorized parties, even if they gain access to the physical database files. It can be applied to data at rest (on disk) or in transit (over the network).
75. **What is role-based access control in DBMS?Role-Based Access Control (RBAC)** is a security model where permissions are not assigned directly to individual users but instead to **roles**. Users are then assigned to one or more roles. This simplifies security management, as you only need to manage the permissions of the roles, not each user.
    - **Example**: You create a `Manager` role with `UPDATE` and `DELETE` privileges and an `Employee` role with `SELECT` privileges. You can then assign new users to the appropriate role.
76. **What is database auditing?Database auditing** is the process of monitoring and recording a specific set of database activities. The goal is to track who did what,

when, and from where. Auditing logs are used for security analysis, compliance requirements, and troubleshooting.

77. **What is data masking?Data masking** is the process of obfuscating or scrambling sensitive data to protect it from unauthorized access. The masked data looks realistic but has no real value. This is useful for providing a production database copy to developers or testers without exposing sensitive information.
    - **Example**: Changing a real credit card number `1234-5678-9012-3456` to a masked number like `xxxx-xxxx-xxxx-3456`.
78. **What is backup and recovery in DBMS?**
    - **Backup**: The process of creating a copy of the database and saving it in a secure location. Backups are essential for disaster recovery.
    - **Recovery**: The process of restoring a database to a consistent state from a backup after a failure. This can involve restoring a full backup and then applying transaction logs to bring the database back to its state just before the failure.
79. **Difference between logical backup and physical backup?**
    - **Logical Backup**:
        - Creates a backup of the data in a human-readable format (e.g., SQL statements).
        - Is platform-independent.
        - Slower to restore because it involves re-creating the database objects and re-inserting the data.
        - **Example**: `mysqldump` in MySQL.
    - **Physical Backup**:
        - Creates a direct copy of the database files and folders on disk.
        - Is platform-dependent.
        - Faster to restore because it's a simple file copy operation.
        - **Example**: Copying the database's data files.
80. **What are high-availability databases?** A **high-availability database** is a system designed to operate continuously with minimal downtime. It uses techniques like **replication**, **clustering**, and **failover** to ensure that if one component fails, another can immediately take over, providing uninterrupted service.

---

9. Distributed Databases

81. **What is a distributed database?** A **distributed database** is a database system where the data is stored on multiple computers or sites that are interconnected via a network. It appears as a single logical database to the user, but the data is physically distributed.
82. **Difference between centralized and distributed database?**
    - **Centralized Database**:
        - Data is stored on a **single** central computer.
        - Easy to manage and control.
        - Can become a **performance bottleneck** and has a single point of failure.

    - **Distributed Database**:
        - Data is stored on **multiple** computers.
        - Provides **higher availability**, scalability, and performance.
        - More complex to manage due to data fragmentation, replication, and concurrency control across multiple sites.
83. **What is fragmentation in distributed DBMS?Fragmentation** is the process of dividing a single relation (table) into multiple fragments that are then stored at different sites. This is a core concept in distributed databases and is used to improve performance and data locality.
84. **What is replication in distributed DBMS?Replication** is the process of storing multiple copies of the same data at different sites. It's used to increase data availability, improve query performance (by directing queries to the closest replica), and provide fault tolerance.
85. **What is two-phase commit protocol?** The **Two-Phase Commit (2PC)** protocol is a distributed algorithm that ensures all nodes in a distributed transaction either **commit** or **abort** the transaction. It's a mechanism for achieving atomicity in distributed systems.
    - **Phase 1 (Vote)**: The coordinator sends a `PREPARE` message to all participants. Each participant votes `YES` if it can commit or `NO` if it can't.
    - **Phase 2 (Commit/Abort)**: If all participants voted `YES`, the coordinator sends a `COMMIT` message. If any voted `NO` or a timeout occurred, the coordinator sends an `ABORT` message.
86. **What is BASE vs ACID in distributed DBMS?**
    - **ACID** is the standard for transactional integrity in **centralized** databases and some distributed ones. It prioritizes strong consistency.
    - **BASE** is a model for **distributed** databases that prioritizes availability over immediate consistency. It acknowledges that in a large-scale, partitioned system, guaranteeing strong consistency across all nodes at all times is often impossible without sacrificing availability.
87. **What is eventual consistency?Eventual consistency** is a consistency model where data changes will eventually propagate through the system, and all replicas will eventually become consistent. It doesn't guarantee that a read will return the most recent write, but it ensures that after some time with no new writes, all reads will be consistent.
88. **What is quorum in distributed databases?** A **quorum** is the minimum number of nodes that must respond to a read or write request for the operation to be considered successful. By requiring a quorum, distributed systems can maintain a certain level of consistency and availability even if some nodes are down.
    - **Example**: In a 5-node cluster, a write quorum of 3 means at least 3 nodes must acknowledge the write for it to be considered successful.
89. **What is Cassandra database?Cassandra** is a highly scalable, fault-tolerant, wide-column NoSQL database. It was designed by Facebook to handle massive amounts of data with high availability and no single point of failure. It uses a peer-to-peer architecture and prioritizes partition tolerance and availability (BASE properties) over strong consistency.
90. **What is Google Spanner / CockroachDB?**

- **Google Spanner**: A globally distributed, transactional database service developed by Google. It provides strong consistency and high availability at a global scale. It achieves this by using atomic clocks to ensure a globally consistent view of time, which allows for true serializable transactions across datacenters.
- **CockroachDB**: An open-source, distributed SQL database inspired by Google Spanner. It provides ACID transactions and horizontal scalability, making it a "NewSQL" database. It is designed to survive disasters and maintain data integrity.

---

10. Latest Trends

91. **Difference between SQL, NewSQL, and NoSQL?**
    - **SQL (RDBMS)**: The classic relational database. Strong consistency, rigid schema, scales vertically.
    - **NoSQL**: Non-relational, flexible schema, scales horizontally, and prioritizes availability over consistency (BASE).
    - **NewSQL**: A class of relational databases that aims to provide the best of both worlds: the horizontal scalability of NoSQL with the ACID properties and relational model of traditional SQL databases.
92. **What is graph database? Example?** A **graph database** is a NoSQL database that uses graph structures (nodes, edges, and properties) to represent and store data. It's highly effective for managing highly interconnected data and navigating relationships.
    - **Example**: **Neo4j**. Used for social networks (finding friends of friends), recommendation engines ("people who like this also like..."), and fraud detection.
93. **What is time-series database? Example?** A **time-series database (TSDB)** is a database optimized for storing and querying data that is timestamped or time-ordered. It's designed for high-volume ingest of data points over time.
    - **Example**: **InfluxDB**, **Prometheus**. Used for monitoring and logging systems, IoT sensor data, and financial trading data.
94. **What is columnar database? Example?** A **columnar database** stores data by column rather than by row. This is highly efficient for analytical queries (OLAP) that involve aggregations over a subset of columns, as the system only needs to read the relevant columns from disk.
    - **Example**: **Cassandra** (wide-column), **ClickHouse**, **Snowflake**. Used in data warehousing and business intelligence.
95. **What is in-memory database (Redis, H2, Memcached)?** An **in-memory database** stores all or most of its data in the main memory (RAM) instead of on a disk. This drastically reduces latency and provides extremely fast data access. They are often used for caching and real-time data processing.
    - **Redis**: A popular key-value store, often used as a cache.
    - **Memcached**: A high-performance, distributed memory caching system.
    - **H2**: A Java-based in-memory RDBMS.

96. **What is database federation?Database federation** is a technology that allows you to access data from multiple, disparate data sources as if they were a single, centralized database. It creates a virtual database on top of existing databases, allowing you to run queries that join data from different systems without needing to physically move or consolidate the data.
97. **What is multi-tenant database architecture?Multi-tenant architecture** is a design where a single instance of a software application (and its database) serves multiple tenants (customers or organizations). The data from each tenant is isolated from the others.
    - **Shared Database, Shared Schema**: All tenants share one database and one set of tables, but their data is isolated by a `tenant_id` column.
    - **Shared Database, Separate Schemas**: All tenants share one database, but each has their own set of tables.
    - **Separate Database, Separate Instance**: Each tenant has their own dedicated database instance.
98. **Difference between schema-on-write vs schema-on-read?**
    - **Schema-on-Write**: The schema is defined and enforced **at the time data is written** to the database. This is the model used by traditional RDBMS.
    - **Schema-on-Read**: The schema is **defined by the application at the time of reading** the data. This is the model used by NoSQL databases, allowing for more flexible data structures.
99. **What is eventual vs strong consistency?**
    - **Strong Consistency**: Guarantees that a read operation on any node will always return the most recent committed write.
    - **Eventual Consistency**: Guarantees that if no new writes are made to the item, all reads will eventually return the last written value. It prioritizes availability and performance over immediate consistency.
100. **Future trends in DBMS (Cloud DBs, Serverless, AI + DB)? * Cloud Databases**: The shift to cloud-native databases is a major trend. Services like Amazon RDS, Aurora, and Google Cloud Spanner provide managed, scalable, and highly available database solutions. * **Serverless Databases**: Databases that automatically scale up and down based on demand and where you pay only for the resources you use. `AWS Aurora Serverless` is a prime example. * **AI/ML Integration**: Databases are increasingly integrating AI and Machine Learning capabilities for tasks like query optimization, predictive indexing, and automated performance tuning. This moves the database towards a self-managing, autonomous system. * **Multi-model Databases**: Databases that support multiple data models (e.g., relational, document, graph) in a single platform