1. What are the Different Types of Joins in SQL? Explain Each with Examples.

- **Answer:**
    - **INNER JOIN:** Returns records that have matching values in both tables.
    - **LEFT JOIN (or LEFT OUTER JOIN):** Returns all records from the left table and the matched records from the right table. If no match, NULL is returned.
    - **RIGHT JOIN (or RIGHT OUTER JOIN):** Returns all records from the right table and the matched records from the left table.
    - **FULL OUTER JOIN:** Returns all records when there is a match in either left or right table. If there is no match, NULL values are returned.
    - **CROSS JOIN:** Returns the Cartesian product of both tables (every row of the first table is combined with every row of the second table).
    - **SELF JOIN:** Joins a table with itself.

2. What is a Primary Key, Foreign Key, and Unique Key?

- **Answer:**
    - **Primary Key:** A column or combination of columns that uniquely identifies each row in a table. It cannot contain NULL values and must have unique values.
    - **Foreign Key:** A column that creates a relationship between two tables. It refers to the primary key of another table and ensures referential integrity.
    - **Unique Key:** A constraint that ensures all values in a column are unique. Unlike a primary key, it can have a single NULL value.

3. What are ACID Properties in SQL? Why are They Important?

- **Answer:**
    - **Atomicity:** Ensures that all operations in a transaction are completed; if not, the transaction is aborted.
    - **Consistency:** Ensures that the database transitions from one valid state to another.
    - **Isolation:** Ensures that transactions are executed in isolation without interfering with each other.
    - **Durability:** Ensures that the results of a transaction are permanently stored in the database, even in case of system failure.
- **Importance:** ACID properties maintain the reliability and integrity of the database.

4. What is Normalization? Explain Different Normal Forms.

- **Answer:**
    - **Normalization:** The process of organizing data in a database to reduce redundancy and improve data integrity.

- **1NF (First Normal Form):** Ensures that each column contains atomic (indivisible) values.
- **2NF (Second Normal Form):** Meets 1NF requirements and removes partial dependencies (non-key attributes depend on the whole primary key).
- **3NF (Third Normal Form):** Meets 2NF requirements and removes transitive dependencies (non-key attributes depend only on the primary key).
- **BCNF (Boyce-Codd Normal Form):** A stricter version of 3NF, where every determinant is a candidate key.

5. What is an Index in SQL? Why and When Should You Use Indexes?

- **Answer:**
  - **Index:** A database object that improves the speed of data retrieval operations on a table by creating an entry for each value.
  - **Why Use Indexes:** To speed up query performance, especially for large datasets.
  - **When to Use:** When querying large tables with frequent search operations, or when using `JOIN`, `WHERE`, or `ORDER BY` clauses.
  - **Drawbacks:** Indexes can slow down `INSERT`, `UPDATE`, and `DELETE` operations due to the need to update the index.

6. What is the Difference Between `HAVING` and `WHERE` Clauses?

- **Answer:**
  - **WHERE:** Filters records before any groupings are made.
  - **HAVING:** Filters records after the grouping has been done. It is typically used with aggregate functions (e.g., `SUM`, `COUNT`).

7. Explain the Difference Between `UNION` and `UNION ALL`.

- **Answer:**
  - **UNION:** Combines the result sets of two or more queries and removes duplicate rows.
  - **UNION ALL:** Combines the result sets of two or more queries without removing duplicates (faster as it skips the duplicate-checking process).

8. What are Views in SQL? What are Their Advantages and Limitations?

- **Answer:**
  - **View:** A virtual table based on a result set of an SQL query. It does not store data itself but displays data from underlying tables.
  - **Advantages:** Simplifies complex queries, provides a layer of security by restricting access to specific columns or rows, and can be used to present data in a specific format.

- **Limitations:** Views can slow down performance for complex queries, especially when they involve multiple joins. They also cannot always be updated directly.

9. What is a Subquery? What is the Difference Between a Correlated and Non-Correlated Subquery?

- **Answer:**
  - **Subquery:** A query nested inside another query. It can be used in `SELECT`, `INSERT`, `UPDATE`, or `DELETE` statements.
  - **Non-Correlated Subquery:** A subquery that can be executed independently of the outer query.
  - **Correlated Subquery:** A subquery that depends on the outer query for its values and is evaluated once for each row processed by the outer query.

10. What is a `TRIGGER` in SQL? When Would You Use It?

- **Answer:**
  - **Trigger:** A database object that is automatically executed or fired when certain events occur (e.g., `INSERT`, `UPDATE`, `DELETE`).
  - **Usage:** To enforce complex business rules, maintain audit logs, or automatically update related tables. For example, logging changes to a specific table or automatically updating a timestamp column when a record is modified.

Additional Theoretical Concepts You Should Be Familiar With:

- **Stored Procedures vs. Functions:** Differences in their purpose, how they are called, and their ability to return values.
- **SQL Injection:** What it is, its risks, and how to prevent it using prepared statements.
- **Transaction Control Statements:** Understanding of `COMMIT`, `ROLLBACK`, `SAVEPOINT`.
- **Difference Between `DELETE`, `TRUNCATE`, and `DROP`:** When to use each and their impact on data and table structure.
- **Entity-Relationship Diagram (ERD):** How to design and explain the relationships between tables in a database using ER diagrams.

11. What is a Composite Key?

- **Answer:**
  - A **composite key** is a combination of two or more columns that uniquely identify a row in a table. It is used when a single column is not sufficient to ensure uniqueness.
  - Example: In a table storing student course enrollments, a composite key could be (`student_id`, `course_id`) since a student can enroll in multiple courses, and a course can have multiple students.

12. What is Denormalization? When Would You Use It?

- **Answer:**
  - **Denormalization** is the process of combining normalized tables to improve read performance by reducing the number of joins. It involves adding redundant data to optimize query performance.
  - **When to Use:** In scenarios where read performance is critical and the overhead of maintaining multiple joins outweighs the benefits of normalization (e.g., in data warehousing).

13. What is a Partitioned Table? Why Use Partitioning?

- **Answer:**
  - A **partitioned table** divides a large table into smaller, more manageable pieces called partitions based on a specified column (e.g., date, range, or list).
  - **Why Use Partitioning:** Improves query performance by allowing the database to scan only relevant partitions. It also helps in managing large datasets by making maintenance tasks like archiving or purging faster.

14. Explain the Difference Between `Clustered` and `Non-Clustered` Indexes.

- **Answer:**
  - **Clustered Index:** Physically sorts the table's data based on the index key. Each table can have only one clustered index because it defines the physical order of the data.
  - **Non-Clustered Index:** Stores a separate structure for the index that references the physical table data. A table can have multiple non-clustered indexes.
  - **Use Case:** Clustered indexes are good for range-based queries (e.g., date ranges), while non-clustered indexes are useful for searching specific values.

15. What is a `WITH` Clause (Common Table Expression, CTE) in SQL?

- **Answer:**
  - The **`WITH` clause** defines a temporary result set that can be referenced within a `SELECT`, `INSERT`, `UPDATE`, or `DELETE` statement. It is also known as a **Common Table Expression (CTE)**.

Example:
sql
Copy code
```
WITH EmployeeCTE AS (
    SELECT employee_id, salary FROM employees WHERE department =
'Sales'
)
SELECT * FROM EmployeeCTE WHERE salary > 5000;
```

-
- **Use Case:** It simplifies complex queries by breaking them down into more manageable parts and can also be used for recursive queries.

16. Explain the Difference Between DELETE and TRUNCATE Commands.

- **Answer:**
  - **DELETE:** Removes rows from a table based on a condition. It is a DML command and can be rolled back using ROLLBACK.
  - **TRUNCATE:** Removes all rows from a table without logging individual row deletions. It is a DDL command and cannot be rolled back.
  - **Difference:** DELETE is slower and can be filtered with WHERE, while TRUNCATE is faster but removes all data unconditionally.

17. What is an Index Scan vs. Index Seek?

- **Answer:**
  - **Index Scan:** The database reads the entire index to find matching rows. It is less efficient for large datasets as it does not use the index's order.
  - **Index Seek:** The database directly navigates to the relevant part of the index to find matching rows. It is faster because it leverages the sorted nature of the index.
  - **Use Case:** Index Seek is preferred for selective queries with conditions that filter a small subset of rows.

18. What are SQL Window Functions? Provide an Example.

- **Answer:**
  - **Window functions** perform calculations across a set of table rows related to the current row, without collapsing the result set like aggregate functions.

Example:
sql
Copy code

```
SELECT employee_id, salary,
       RANK() OVER (ORDER BY salary DESC) as salary_rank
FROM employees;
```

-
- **Use Case:** Useful for ranking, calculating running totals, or moving averages.

19. What is a Deadlock in SQL? How Can It Be Prevented?

- **Answer:**
  - A **deadlock** occurs when two or more transactions block each other by holding a lock on a resource the other needs. This results in a situation where none of the transactions can proceed.
  - **Prevention:**
    - Ensure consistent locking order.
    - Use shorter transactions to minimize the time locks are held.
    - Implement proper error handling and retry logic for deadlocked transactions.

20. What is the Purpose of Using a `HAVING` Clause Instead of a `WHERE` Clause with Aggregate Functions?

- **Answer:**
  - The **`WHERE` clause** is used to filter rows before any grouping or aggregation happens. It cannot be used with aggregate functions like `SUM()`, `COUNT()`.
  - The **`HAVING` clause** is used to filter groups after the aggregation has occurred.

Example:
sql
Copy code
```
SELECT department, COUNT(*) as employee_count
FROM employees
GROUP BY department
HAVING COUNT(*) > 10;
```

-
- Here, `HAVING` filters departments with more than 10 employees after counting.

21. What is a Stored Procedure? What are Its Advantages?

- **Answer:**
  - A **stored procedure** is a precompiled collection of SQL statements stored in the database, which can be executed as a single unit.
  - **Advantages:**
    - Reduces client-server communication by executing multiple SQL statements in a single call.
    - Enhances performance due to precompilation.

        - Improves code reusability and maintainability.
        - Increases security by limiting direct access to underlying tables.

22. What is the Difference Between `HAVING`, `WHERE`, and `GROUP BY` Clauses?

- **Answer:**
    - **`WHERE`:** Filters rows before any grouping takes place.
    - **`GROUP BY`:** Groups rows sharing a property so that aggregate functions can be applied.
    - **`HAVING`:** Filters groups after the aggregation has been performed.

Example:
sql
Copy code
```
SELECT department, COUNT(*)
FROM employees
WHERE status = 'active'
GROUP BY department
HAVING COUNT(*) > 5;
```

23. What is a `CASE` Statement in SQL? Provide an Example.

- **Answer:**
    - The **`CASE`** statement is used to perform conditional logic in SQL.

Example:
sql
Copy code
```
SELECT employee_id,
       CASE
           WHEN salary > 10000 THEN 'High'
           WHEN salary BETWEEN 5000 AND 10000 THEN 'Medium'
           ELSE 'Low'
       END AS salary_level
FROM employees;
```

24. What are `Union` and `Union All`? Which is Faster and Why?

- **Answer:**
  - **`UNION`:** Combines result sets of two queries and removes duplicates.
  - **`UNION ALL`:** Combines result sets without removing duplicates.
  - **Which is Faster:** `UNION ALL` is faster because it does not perform the duplicate check.

25. What is a Materialized View? How is It Different from a Regular View?

- **Answer:**
  - A **materialized view** stores the result of a query physically on disk, unlike a regular view which is a virtual table.
  - **Advantages:** Faster query performance because the data is precomputed and stored.
  - **Disadvantages:** Requires maintenance and synchronization when the underlying data changes.

26. Explain Referential Integrity in SQL.

- **Answer:**
  - **Referential Integrity** ensures that a foreign key value always matches a primary key value in the referenced table. It prevents orphan records and maintains data consistency.
  - **Example:** If an `order` table has a `customer_id` foreign key referencing the `customers` table's primary key, any value in `customer_id` must exist in the `customers` table.

27. What are SQL Aggregate Functions? List Some Common Ones.

- **Answer:**
  - **Aggregate functions** perform calculations on multiple rows and return a single result.
  - **Common Functions:**
    - `COUNT()`: Counts the number of rows.
    - `SUM()`: Returns the total sum of a numeric column.
    - `AVG()`: Returns the average value.
    - `MAX()`: Returns the maximum value.
    - `MIN()`: Returns the minimum value.

28. What is an Index in SQL? How Does It Improve Query Performance?

- **Answer:**
  - An **index** is a data structure that improves the speed of data retrieval operations on a table.

    - It reduces the amount of data the database needs to scan to find specific records, making searches faster.

29. Explain the Concept of NULL in SQL. How is It Different from Zero or an Empty String?

- **Answer:**
    - **NULL** represents a missing or unknown value. It is different from zero (0) or an empty string ('').
    - **Example:** `NULL` indicates the absence of a value, while `0` and `''` are actual values.

30. What is a `SELF JOIN`? When Would You Use It?

- **Answer:**
    - A **SELF JOIN** is a join of a table with itself. It is used when you need to compare rows within the same table.
    - **Example:** Finding employees who report to the same manager in an employee table.

31. What is a `Cross Join`? Provide an Example.

- **Answer:**
    - A **CROSS JOIN** returns the Cartesian product of two tables (i.e., every row from the first table is combined with every row from the second table).

Example:
sql
Copy code
```
SELECT * FROM table1
CROSS JOIN table2;
```

    -

32. What is a Recursive CTE (Common Table Expression)? Provide an Example.

- **Answer:**
    - A **recursive CTE** is used to perform recursive queries, such as traversing hierarchical data (e.g., organizational charts).

Example:
sql

Copy code
```
WITH RECURSIVE EmployeeHierarchy AS (
    SELECT employee_id, manager_id, 1 AS level
    FROM employees
    WHERE manager_id IS NULL
    UNION ALL
    SELECT e.employee_id, e.manager_id, eh.level + 1
    FROM employees e
    INNER JOIN EmployeeHierarchy eh ON e.manager_id = eh.employee_id
)
SELECT * FROM EmployeeHierarchy;
```

33. Explain the Concept of Transactions in SQL. Why Are They Important?

- **Answer:**
    - A **transaction** is a sequence of SQL operations executed as a single unit of work. It ensures data consistency through **ACID properties**.
    - **Importance:** Transactions help maintain data integrity, especially in scenarios where multiple operations need to be completed successfully together.

34. What is a `Savepoint` in SQL? How Is It Used?

- **Answer:**
    - A **savepoint** is a point within a transaction to which you can roll back without affecting the entire transaction.

Example:
sql
Copy code
```
BEGIN TRANSACTION;
SAVEPOINT sp1;
UPDATE employees SET salary = salary * 1.1;
ROLLBACK TO sp1;
```

35. What are SQL Wildcards? Explain With Examples.

- **Answer:**
    - **Wildcards** are used with the `LIKE` operator to search for patterns.

        - `%` matches any number of characters.
        - `_` matches a single character.
        - **Example:** `SELECT * FROM employees WHERE name LIKE 'J%';` finds names starting with 'J'.

36. What is Data Integrity? How Do You Ensure It in a Database?

- **Answer:**
    - **Data Integrity** ensures accuracy, consistency, and reliability of data over its lifecycle.
    - **Methods:** Using constraints (Primary Key, Foreign Key, Unique, Check), triggers, and ACID-compliant transactions.

37. What is an Execution Plan in SQL? How Can It Help Optimize Queries?

- **Answer:**
    - An **execution plan** shows the steps the SQL engine uses to execute a query. It helps identify bottlenecks, such as full table scans or inefficient joins, and provides insights for optimizing queries.

38. What are the Differences Between OLTP and OLAP?

- **Answer:**
    - **OLTP (Online Transaction Processing):** Focuses on fast query processing, data integrity, and supporting day-to-day operations (e.g., banking transactions).
    - **OLAP (Online Analytical Processing):** Focuses on complex queries, data analysis, and decision-making (e.g., data warehousing).

39. What is a `Temporary Table` in SQL? When Should It Be Used?

- **Answer:**
    - A **temporary table** is a table created in a session to store intermediate results. It is automatically dropped when the session ends.
    - **Use Case:** When you need to store temporary data for complex calculations or reporting without affecting the main tables.

40. What is a Cursor in SQL? What Are Its Types?

- **Answer:**
    - A **cursor** is a database object used to retrieve a set of rows one at a time.
    - **Types:**
        - **Static Cursor:** Provides a snapshot of the result set.
        - **Dynamic Cursor:** Reflects changes in the data while the cursor is open.
        - **Forward-Only Cursor:** Moves in one direction through the result set.

        - **Keyset-Driven Cursor:** The set of keys is fixed, but the data can change.

41. What is Database Caching? Why is it Used?

- **Answer:**
    - **Database caching** is a technique that stores a subset of data in memory to reduce the time it takes to access that data.
    - **Use Case:** It improves read performance, reduces latency, and alleviates the load on the primary database, making it especially useful in high-traffic applications.

42. How Can You Implement Caching in a Database Application?

- **Answer:**
    - **In-Memory Caching**: Using caching solutions like **Redis** or **Memcached** to store frequently accessed data in memory.
    - **Database Query Caching**: Utilizing built-in caching mechanisms like MySQL's query cache (deprecated in recent versions) or enabling result caching in Oracle.
    - **Application-Level Caching**: Caching data at the application layer using frameworks like **Spring Cache** in Java.

43. What Are the Different Types of Indexes in SQL?

- **Answer:**
    - **Primary Index**: Automatically created on the primary key of a table.
    - **Unique Index**: Prevents duplicate values in the indexed column.
    - **Clustered Index**: Determines the physical order of data in a table (only one per table).
    - **Non-Clustered Index**: Creates a separate structure that points to the data (multiple can exist per table).
    - **Full-Text Index**: Used for efficient text search on large text columns.

44. When Should You Avoid Using an Index?

- **Answer:**
    - When the table is small (a full table scan might be faster).
    - When the column has low cardinality (many duplicate values).
    - For columns that are frequently updated (as indexes need to be maintained).
    - When the overhead of maintaining the index outweighs its performance benefits.

45. How Does Indexing Affect the Performance of `INSERT`, `UPDATE`, and `DELETE` Operations?

- **Answer:**

  - **Insertions**: Slower because the index needs to be updated every time a new record is added.
  - **Updates**: Slower if the indexed columns are updated, as the index needs to be maintained.
  - **Deletions**: Slower because the index entries for the deleted rows must be removed.

46. Explain the Concept of an Index Scan and an Index Seek.

- **Answer:**
  - **Index Scan**: The database engine scans the entire index to find the required data. It is less efficient, typically used when no selective filter is available.
  - **Index Seek**: The database engine uses the index to directly find the specific record. It is faster and more efficient.

47. What is a Covering Index? Provide an Example.

- **Answer:**
  - A **covering index** is an index that contains all the columns needed by a query, so the query can be satisfied entirely using the index without accessing the table.

Example:
sql
Copy code
```
CREATE INDEX idx_employee ON employees (department_id, name);
SELECT name FROM employees WHERE department_id = 5;
```

  - Here, the query can use the index `idx_employee` without accessing the full table.

48. How Does Caching Affect Database Performance?

- **Answer:**
  - **Positive Impact**: Reduces latency for frequently accessed data, lowers database load, and speeds up response times for read-heavy operations.
  - **Negative Impact**: If not properly managed, cache can become stale (serve outdated data) or lead to cache thrashing (frequent invalidations).

49. What Are the Downsides of Using Too Many Indexes?

- **Answer:**
  - Increased storage requirements.
  - Slower `INSERT`, `UPDATE`, and `DELETE` operations due to index maintenance overhead.

    - Potential performance degradation if queries use suboptimal indexes or the wrong index is selected by the query optimizer.

50. What is Query Plan Caching in Databases?

- **Answer:**
    - **Query plan caching** stores the execution plan of a previously executed query in memory. This allows the database to reuse the plan for identical queries, avoiding the overhead of recalculating the plan and speeding up query execution.