SQL Developer Interview Questions with Answers

1. What is SQL? Explain its purpose.

SQL, or Structured Query Language, is a programming language that allows users to communicate with and manage relational databases. It's used to perform tasks such as:

- Storing data: SQL can be used to store information in a relational database.
- Updating data: SQL can be used to update data in a database.
- Retrieving data: SQL can be used to retrieve data from a database.
- Searching data: SQL can be used to search for data in a database.
- Maintaining databases: SQL can be used to maintain and optimize database performance.

2. What are the different types of SQL commands?

SQL commands are divided into five main categories:

- DDL (Data Definition Language): CREATE, ALTER, DROP, TRUNCATE
- DML (Data Manipulation Language): SELECT, INSERT, UPDATE, DELETE
- DCL (Data Control Language): GRANT, REVOKE
- TCL (Transaction Control Language): COMMIT, ROLLBACK, SAVEPOINT
- DQL (Data Query Language): SELECT

3. What is the difference between DELETE, TRUNCATE, and DROP?

- DELETE: Removes rows from a table based on a condition. Can be rolled back if within a transaction.
- TRUNCATE: Removes all rows from a table, but it cannot be rolled back. It's faster than DELETE.
- DROP: Deletes the entire table (along with its structure) from the database. This action is irreversible.

4. Explain the concept of normalization in SQL.

Normalization is the process of organizing data to minimize redundancy and improve data integrity.

There are several normal forms:

- 1NF (First Normal Form): Eliminates duplicate columns and ensures atomic values in fields.
- 2NF (Second Normal Form): Ensures 1NF is followed and removes partial dependencies.
- 3NF (Third Normal Form): Ensures 2NF is followed and removes transitive dependencies.
- BCNF (Boyce-Codd Normal Form): A stronger version of 3NF, ensuring no anomalies.

5. What are indexes in SQL, and how do they work?

Indexes are used to speed up the retrieval of rows from a database. There are two types:

- Clustered Index: Sorts and stores the data rows in the table based on the indexed column(s).
- Non-clustered Index: Maintains a separate structure that points back to the original table rows.

6. What is the difference between JOIN, LEFT JOIN, and RIGHT JOIN?

- JOIN (INNER JOIN): Returns only the rows where there is a match in both tables.
- LEFT JOIN (LEFT OUTER JOIN): Returns all rows from the left table and matched rows from the right table.
- RIGHT JOIN (RIGHT OUTER JOIN): Returns all rows from the right table and matched rows from the left table.

7. What is a view, and how is it used?

A view is a virtual table based on the result of a SELECT query. It is used to simplify complex queries,improve security, and make data access more convenient.

8. How do you handle performance tuning in SQL?

Performance tuning involves:

- Using indexes efficiently to speed up searches.
- Writing optimized SQL queries (e.g., avoiding SELECT *, using joins wisely).
- Denormalizing in cases where performance is crucial.
- Partitioning large tables to improve query performance.

9. Explain the ACID properties in the context of a transaction.

ACID stands for:

- Atomicity: Ensures that all operations in a transaction are completed or none at all.
- Consistency: Ensures that a transaction brings the database from one valid state to another.
- Isolation: Transactions should not interfere with each other.
- Durability: Once a transaction is committed, the changes are permanent.

10. What are stored procedures, and when would you use them?

A stored procedure is a precompiled collection of SQL statements stored in the database.

They are used to improve performance, encapsulate logic, and enhance security.

11. What is the difference between UNION and UNION ALL?

- UNION: Combines results of two SELECT queries but removes duplicate rows.
- UNION ALL: Combines results and includes all rows, even duplicates.

12. How do you optimize a query that is running slowly?

- Use indexes: Ensure the query uses indexed columns.
- Limit the result set: Avoid SELECT * and retrieve only necessary columns.
- Analyze the query execution plan to identify bottlenecks.
- Use joins instead of subqueries where possible.

13. How do you handle errors in SQL?

- Use TRY...CATCH blocks to catch exceptions.
- Handle errors gracefully by logging them and either rolling back transactions or alerting the user.

14. What are the types of joins in SQL?

- INNER JOIN: Returns records with matching values in both tables.
- LEFT JOIN: Returns all records from the left table and matched records from the right table.
- RIGHT JOIN: Returns all records from the right table and matched records from the left table.

-FULL OUTER JOIN: Returns all records when there is a match in either left or right table.

15. What is a trigger in SQL?

A trigger is a SQL code that automatically executes in response to certain events on a table.

16. What is a primary key and a foreign key?

- A primary key is a column that uniquely identifies each row in a table.

- A foreign key is a column that references the primary key in another table to ensure referential integrity.

17. How do you handle NULL values in SQL?

- Use IS NULL and IS NOT NULL to check for NULL values.

- Use COALESCE, IFNULL, or NVL to replace NULL values with defaults.

- Use NULLIF to convert specific values to NULL.

- Understand how NULL values are handled in aggregate functions (COUNT, SUM, etc.).

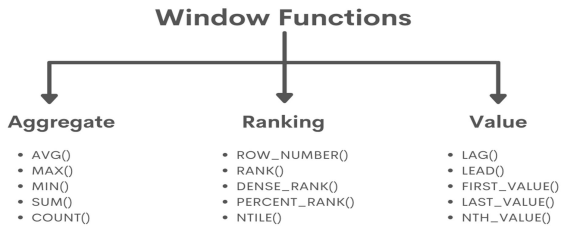
- Use CASE expressions to handle NULL values conditionally.

- Control the placement of NULL values in sorting with ORDER BY.

- Be cautious when using NULL with IN or NOT IN.

18. Explain window functions and their use cases?

Window functions in SQL are used to perform calculations on sets or rows of data, also known as "windows". They are used for analytical purposes, such as calculating running totals, averages, and rankings.



Here are some examples of window functions and their use cases:

- LAG() and LEAD(): These functions compare values in a row with values in a preceding or following row, respectively.

- NTILE(): This function divides a result set into a number of equal groups.
- SUM(): This function calculates the total of a numeric column in a given window.
- Last_value(): This function returns the last value in a group.
- Cumulative Sum: Also known as running totals, this function shows the accumulation or growth of a metric over time.

19. Given a table of sales data, Write a SQL query to compute the cumulative sum of sales for each product, sorted by product_id and date.

```
SELECT  s1.product_id,s1.date,SUM(s2.price) AS cumulative_sum
FROM sales s1 JOIN sales s2
ON s1.product_id = s2.product_id AND s1.date >= s2.date
GROUP BY s1.product_id, s1.date
ORDER BY s1.product_id, s1.date;
```

20. Describe a situation where you had to debug a complex query or database issue. How did you resolve it?

- **Identify Core Components:** Start by isolating the main parts of your query. ...
- **Simplify Joins:** Break down multiple joins into simpler, individual join statements. ...
- **Incremental Development:** Build your query incrementally, adding one component at a time.
- Refined the Query: To further optimize the query.
- **Partitioning (Long-Term Solution):** Since the sales table was growing rapidly, I suggested partitioning the table by month based on the sale_date. This strategy would optimize queries that filtered by date ranges. After implementing partitioning, queries filtering by dates saw a significant performance boost, as only relevant partitions were scanned.
- **Added Missing Indexes**: I added indexes on columns that were frequently used in WHERE, JOIN, and GROUP BY clauses

21. How do you handle a situation where a query performs well in development but is slow in production?

By optimizing the query design, you can improve the performance and the readability of your queries. You can apply some general best practices, such as using indexes, avoiding unnecessary columns, using appropriate data types, limiting the result set, using parameters, and simplifying the logic.

22. What experience do you have working with large datasets and optimizing SQL for high-volume transactions?

I have extensive experience working with large datasets and optimizing SQL queries for high-volume transactions in both transactional (OLTP) and analytical (OLAP) environments. Here are some key areas where I have applied optimization techniques:
Database Indexing Strategies, Query Optimization Techniques, Partitioning Large Tables ,Managing High-Volume Transactions, Caching and Materialized Views, Query Execution Plan Analysis.

23. How do you use SQL in ETL processes?

Using SQL in ETL (Extract, Transform, Load) processes is a common practice for managing and transforming data efficiently. Here's how I have used SQL at each stage of the ETL process, along with best practices and examples:

1. Extract:

SQL in the Extract Phase:

- Data Retrieval: SQL is primarily used to extract data from various sources, such as relational databases, data warehouses, and sometimes even flat files or APIs when combined with ETL tools.
- Query Optimization: Writing efficient SQL queries to retrieve only the necessary data can help reduce the amount of data transferred and improve extraction speed.

  e.g For example, the query extracts customer data created in 2023. This selective extraction helps minimize the volume of data processed downstream.

2. Transform:

SQL in the Transform Phase:

- Data Cleaning: SQL is used to clean the data by filtering out invalid entries, handling NULL values, and formatting data types.
- Data Transformation: Applying business rules and calculations, such as aggregations, joins, and case statements to derive new metrics or reshape the data.
- Data Integration: Combining data from multiple sources through joins and unions to create a unified dataset.
  e.g. In this transformation, the data is cleaned and aggregated to show the number of customers registered each month. The COALESCE function is used to replace NULL phone numbers with 'N/A', and DATE_TRUNC is used to group data by month.

3. Load:

SQL in the Load Phase:

- **Data Insertion**: SQL is used to insert transformed data into the target data warehouse or database. This can be done with bulk inserts for efficiency.
- **Handling Duplicate Records:** SQL can be used to ensure that duplicate records are managed appropriately, either by ignoring them or updating existing records.

e.g. For example to insert monthly summary data into a table and updates existing records if the month already exists.

24. **What are materialized views, and how are they different from regular views?**

Feature	Regular Views	Materialized Views
Storage	Does not store data (virtual view)	Stores data physically on disk
Data Refresh	Always reflects the current data	Must be refreshed manually or on a schedule
Performance	May slow down performance for complex queries as it runs the underlying query every time	Provides faster access to precomputed results
Use Cases	Suitable for dynamic data that needs to be always current	Ideal for reporting and analytical queries where performance is critical
Update Behavior	Changes in underlying tables are immediately visible	Requires refresh to show updated data

Advantages of Materialized Views

1. **Performance Improvement**: Since materialized views store precomputed results, they can dramatically reduce query execution time, especially for complex aggregations and joins.
2. **Simplified Queries**: They allow users to work with simplified queries without having to repeatedly compute complex calculations.
3. **Reduced Load on Source Tables**: By querying materialized views instead of the underlying tables, you can reduce the load on the source tables, especially during peak times.

Use Cases for Materialized Views

- **Data Warehousing**: Frequently used in data warehouses for reporting purposes where aggregated data is required for analysis.
- **High-Performance Dashboards**: Used in business intelligence dashboards that need to display real-time metrics without querying the underlying data sources directly.
- **Reporting**: Ideal for generating reports where the underlying data changes infrequently but the report needs to be generated quickly.

25. **Explain your understanding of HiveQL and its differences with traditional SQL?**

Differences Between HiveQL and Traditional SQL

Feature	HiveQL	Traditional SQL
Execution Model	Converts queries into MapReduce jobs	Typically executes queries directly on relational databases
Schema Handling	Schema on read (data can be unstructured)	Schema on write (data must fit predefined schema before storage)
Performance	Optimized for batch processing; slower response times for ad-hoc queries	Optimized for transactional processing; faster response times for individual queries
Data Types	Limited set of data types (e.g., ARRAY, MAP, STRUCT)	Comprehensive set of data types, including complex types
Joins	Supports joins but can be less efficient for large datasets	Highly optimized joins for performance, particularly in OLTP environments
Storage	Works with data stored in HDFS, HBase, and other Hadoop-compatible storage	Works with structured data stored in relational databases
Concurrency	Less emphasis on concurrency; designed for	Strong support for concurrent transactions and

Feature	HiveQL	Traditional SQL
Control	batch jobs	ACID compliance

Advantages of HiveQL

- **Scalability**: HiveQL can handle petabytes of data, leveraging the distributed nature of Hadoop to process large datasets.
- **Flexibility**: The schema on read approach allows for storing unstructured and semi-structured data without needing to define a schema upfront.
- **Familiar Syntax**: HiveQL's SQL-like syntax makes it easier for analysts and data engineers familiar with SQL to transition to working with big data.

- **Use Cases for HiveQL**
- **Data Analysis**: Performing complex queries and aggregations on large datasets for reporting and analytics.
- **Data Transformation**: ETL processes to transform and load data into a structured format for downstream analysis.
- **Batch Processing**: Running large-scale batch jobs to process logs, clickstream data, or historical data analysis.