SQL

- SQL, which stands for Structured Query Language, is a powerful programming language used for managing and manipulating relational databases.
- It provides a standardized way to communicate with databases and perform various operations, such as creating tables, inserting data, updating records, and retrieving information.
- SQL follows a declarative approach, allowing users to specify what they want to achieve rather than how to achieve it, making it easy to learn and use.
- SQL is widely used in the industry and is supported by most database management systems, including MySQL, Oracle, SQL Server, and PostgreSQL.
- With SQL, you can perform complex queries by combining conditions, joining tables, aggregating data, and sorting results.
- SQL offers a wide range of operators, functions, and clauses, such as SELECT, FROM, WHERE, JOIN, GROUP BY, HAVING, and ORDER BY, to help you manipulate and retrieve data effectively.
- It supports data integrity through constraints like primary keys, foreign keys, unique constraints, and check constraints, ensuring the accuracy and consistency of the database.
- SQL provides transaction management capabilities, allowing users to perform atomic and consistent operations on the database by using commands like COMMIT and ROLLBACK.
- SQL also offers data manipulation capabilities, enabling you to insert, update, and delete records in a database, ensuring efficient data management.
- SQL is a fundamental skill for anyone working with data and databases, as it empowers developers, data analysts, and administrators to interact with and extract valuable insights from vast amounts of structured data.
-

Data modelling

Data modelling is the process of creating a conceptual representation of data structures and relationships within a domain. It involves defining the logical structure, organization, and attributes of data to facilitate efficient data storage, retrieval, and manipulation.

Data modelling typically begins with understanding the requirements and objectives of the system or application being developed. This includes identifying entities (objects, concepts, or things) relevant to the domain and determining their attributes and relationships.

There are different types of data models, including conceptual, logical, and physical models.

- Conceptual models focus on high-level concepts and relationships, providing an abstract view of the data without specifying implementation details. They help stakeholders understand the domain and communicate requirements.

- Logical models describe the structure and relationships of the data in a more detailed manner. They serve as a blueprint for database design, defining entities, attributes, relationships, and constraints.
- Physical models represent how the data is physically stored in a database or data management system. They consider implementation-specific details such as data types, indexes, and table structures.

Data modelling employs various techniques and notations, such as entity-relationship diagrams (ERDs), UML diagrams, and data flow diagrams. These visual representations aid in documenting, analysing, and communicating complex data structures and processes.

Effective data modelling improves data quality, enhances system performance, and ensures data integrity. It helps in designing efficient databases, optimizing queries, and supporting data-driven decision-making processes.

- ***Importance of Data Modelling***

Data modelling is a critical step in the database design process as it establishes the blueprint for how data will be organized, stored, and accessed within a database system. Here are some key reasons why data modelling is important:

- ❖ Data Organization: Data modelling helps structure data by identifying entities, attributes, and relationships. Entities represent real-world objects, such as customers, products, or employees, while attributes define the properties or characteristics of these entities. Relationships establish connections between entities, determining how they interact with each other. By organizing data in this manner, data modelling enables efficient storage and retrieval of information.
- ❖ Data Integrity: Data modelling ensures data integrity by enforcing constraints and rules. Constraints define the boundaries and rules for data values, such as data types, ranges, uniqueness, and relationships. By defining these constraints, data modelling helps maintain consistency, accuracy, and reliability in the stored data. It prevents data anomalies, such as duplication, inconsistencies, and conflicts, that can lead to data corruption or erroneous results.
- ❖ System Efficiency: Proper data modelling contributes to system efficiency by optimizing data retrieval and query performance. By understanding the relationships between entities and their attributes, data modelling enables the creation of appropriate data structures, indexes, and access paths. These optimizations enhance query execution and improve overall system performance. Efficient data modelling also facilitates data updates, inserts, and deletions by minimizing the impact on the database.
- ❖ Application Development: Data modelling serves as the foundation for developing data-driven applications. By creating a well-defined data model, developers can understand the data requirements of the application and map them to the corresponding functionality. This alignment between data and application ensures

consistency, maintainability, and scalability in the development process. Additionally, data modelling helps in the integration of multiple applications by providing a common understanding of data structures and relationships.

Normalization

Normalization is a process in database design that helps eliminate data redundancy, improve data integrity, and ensure efficient data storage and manipulation. It involves organizing and structuring the database tables to minimize data duplication and maintain data consistency.

The goal of normalization is to break down a database into smaller, well-structured tables, each containing a specific set of related data. This is achieved by applying a set of rules known as normalization forms, the most commonly used being the first, second, and third normal forms (1NF, 2NF, and 3NF).

1. **First Normal Form (1NF)**: In 1NF, data is organized into tables with atomic values, meaning each attribute contains only a single value. This eliminates repeating groups or multiple values within a single attribute.
2. **Second Normal Form (2NF)**: 2NF builds upon 1NF by ensuring that each non-key attribute is functionally dependent on the entire primary key. It eliminates partial dependencies, where an attribute depends on only a part of the primary key.
3. **Third Normal Form (3NF)**: 3NF further refines the table structure by removing transitive dependencies. Transitive dependencies occur when an attribute depends on another non-key attribute rather than directly on the primary key.

Benefits of Normalization:

a. **Data Redundancy Reduction**: By eliminating redundant data, normalization reduces the storage space required for storing data. Reducing redundancy also leads to a decrease in update anomalies, where changes to data must be propagated consistently across all instances of redundant information. This ensures that data remains consistent and avoids conflicts.

b. **Data Integrity Improvement**: Normalization improves data integrity by minimizing the risk of inconsistent or contradictory information. By breaking down data into smaller tables and ensuring that each table represents a single entity or relationship, normalization prevents update anomalies. It ensures that modifications to data are made in a consistent and controlled manner, maintaining the integrity of the database.

c**. Query Performance Optimization**: Normalization can enhance query performance by reducing the number of joins required to retrieve data. By decomposing data into smaller, specialized tables, normalization enables efficient data retrieval based on specific queries. This can lead to faster query execution times and improved system performance.

d. **Flexibility and Scalability**: Normalized database designs offer more flexibility and scalability. As data is organized into smaller, atomic units, it becomes easier to modify or extend the database structure without impacting the entire system. This flexibility allows for easier maintenance, future modifications, and adaptation to changing business requirements.

Star schema

A star schema is a popular data modelling technique used in data warehousing to organize and represent data for efficient querying and analysis. It is characterized by a central fact table surrounded by multiple dimension tables, forming a shape resembling a star.

In a star schema, the fact table contains the primary measures or numerical data that are the focus of analysis, such as sales revenue, quantity sold, or website visits. Each fact table record is associated with keys that link to the related dimension tables.

Dimension tables provide descriptive attributes or dimensions that provide context to the measures in the fact table. For example, in a sales data warehouse, dimension tables may include products, customers, time, and locations. Each dimension table typically has a primary key that is referenced in the fact table as a foreign key.

The star schema design offers several benefits:

1. Simplicity and understandability: The star schema is straightforward and easy to comprehend, making it user-friendly for business analysts and data consumers.
2. Simplified queries and aggregations: The structure of the star schema allows for simple and efficient querying of data. Aggregations and roll-ups across dimensions can be easily performed using the fact table's measures and the dimension table's attributes.
3. Improved query performance: Star schemas are optimized for query performance, as the denormalized structure avoids complex joins between tables.
4. Scalability: The star schema can accommodate large volumes of data and handle complex analytical queries efficiently.
5. Flexibility: The star schema enables easy integration of new dimensions or measures into the data warehouse without affecting existing tables or queries, allowing for agile and iterative development.
6. Enhanced data consistency and integrity: By separating the fact and dimension tables, data redundancy and inconsistencies are minimized, ensuring data integrity throughout the schema.

Benefits of Star Schema:

a. **Simplified Querying and Reporting**: The star schema simplifies querying and reporting processes by providing a denormalized structure. The central fact table contains measures or metrics (e.g., sales, revenue) and is surrounded by dimension tables that provide descriptive attributes (e.g., product, time, location). This structure allows for straightforward and efficient querying, as complex joins across multiple tables are minimized.

b. **Improved Query Performance**: The star schema's denormalized structure improves query performance. Since the dimension tables are separate from the fact table, aggregations, filtering, and grouping operations can be performed more efficiently. Queries involving slicing and dicing of data or aggregating across dimensions can be executed quickly, enabling faster decision-making and analysis.

c**. Enhanced Data Analysis**: The star schema design facilitates data analysis and exploration. With dimension tables representing different dimensions of interest (e.g., product, time, location), analysts can easily drill down into specific dimensions or perform multi-dimensional analysis. The simplicity and intuitiveness of the star schema enable users to gain insights from data quickly.

d. **Scalability and Maintainability**: Star schema designs are highly scalable and maintainable. New dimensions or measures can be added to the schema without disrupting existing queries or structures. This flexibility allows for the incorporation of additional data sources or business requirements, ensuring the data warehouse can adapt to evolving needs.

Acid transactions:

ACID is an acronym that stands for Atomicity, Consistency, Isolation, and Durability. ACID transactions are a set of properties that ensure the reliability and integrity of database transactions.

1. **Atomicity**: Atomicity guarantees that a transaction is treated as a single, indivisible unit of work. It ensures that all the changes within a transaction are either committed in their entirety or rolled back if any part of the transaction fails. In other words, a transaction is all or nothing.
2. **Consistency**: Consistency ensures that a transaction brings the database from one valid state to another. It means that the data must satisfy all the defined rules, constraints, and relationships. If a transaction violates any of these constraints, it is rolled back to maintain data integrity.
3. **Isolation**: Isolation ensures that concurrent transactions do not interfere with each other. Each transaction operates in isolation and appears to be executed serially, even if multiple transactions are executed concurrently. This prevents issues such as dirty reads, non-repeatable reads, and phantom reads.

4. **Durability**: Durability guarantees that once a transaction is committed, its changes are permanent and will survive any subsequent failures, such as power outages or system crashes. The committed data is stored in a way that can be recovered and restored in case of failures.

DML and DQL

DML and DQL are two categories of SQL (Structured Query Language) statements used to manipulate and query databases.

1. **DML (Data Manipulation Language)**: DML statements are used to modify and manipulate data within a database. The primary DML statements are:
    - **INSERT**: Adds new records or rows into a table.
    - **UPDATE**: Modifies existing records or rows in a table.
    - **DELETE**: Removes records or rows from a table.
    - **MERGE**: Performs both INSERT and UPDATE operations based on specified conditions.
    - **TRUNCATE**: Removes all records from a table, but the table structure remains intact.
2. **DQL (Data Query Language)**: DQL statements are used to retrieve or query data from a database. The most common DQL statement is:
    - **SELECT**: Retrieves specific columns or records from one or more tables based on specified conditions. SELECT statements can include various clauses such as WHERE, GROUP BY, HAVING, ORDER BY, and JOIN to filter, aggregate, and sort the data.

DML statements are focused on modifying data, while DQL statements are focused on retrieving data. Both DML and DQL statements are integral parts of SQL and are used extensively in database operations and data manipulation tasks.

Join operations:

Join operations in SQL are used to combine data from multiple tables based on related columns or fields. They allow you to retrieve and analyse data from multiple tables as if they were a single table. Join operations play a crucial role in querying and analysing relational databases.

When creating your document on join operations, consider including the following explanations:

1. **Inner Join**: The inner join returns only the matching records between two tables based on a specified join condition. It combines rows from both tables where the join condition is satisfied, discarding unmatched rows from either table.

SELECT Customers.customer_id, Customers.name, Orders.order_id, Orders.order_date

FROM Customers

INNER JOIN Orders ON Customers.customer_id = Orders.customer_id;

This query retrieves the customer ID, customer name, order ID, and order date for customers who have placed orders. It combines the matching records from both tables based on the customer ID.

2. **Left Join (or Left Outer Join)**: The left join returns all records from the left table and the matching records from the right table based on the join condition. If a record from the left table has no matching record in the right table, the result will include NULL values for the right table columns.

   SELECT Customers.customer_id, Customers.name, Orders.order_id, Orders.order_date

   FROM Customers

   LEFT JOIN Orders ON Customers.customer_id = Orders.customer_id;

   This query retrieves the customer ID, customer name, order ID, and order date for all customers, including those who haven't placed any orders. If a customer has no matching order, the order-related columns will contain NULL values.

3. **Right Join (or Right Outer Join)**: The right join returns all records from the right table and the matching records from the left table based on the join condition. If a record from the right table has no matching record in the left table, the result will include NULL values for the left table columns.

   SELECT Customers.customer_id, Customers.name, Orders.order_id, Orders.order_date

   FROM Customers

   RIGHT JOIN Orders ON Customers.customer_id = Orders.customer_id;

   This query retrieves the customer ID, customer name, order ID, and order date for all orders, including those without matching customers. If an order has no matching customer, the customer-related columns will contain NULL values.

4. **Full Join (or Full Outer Join)**: The full join returns all records from both the left and right tables. It includes all the rows from both tables, whether they have a match or not. Unmatched rows will have NULL values for the columns of the non-matching table.

   SELECT Customers.customer_id, Customers.name, Orders.order_id, Orders.order_date

   FROM Customers

FULL JOIN Orders ON Customers.customer_id = Orders.customer_id;

This query retrieves the customer ID, customer name, order ID, and order date for all customers and orders. It includes all rows from both tables, matching or non-matching. NULL values will appear for non-matching rows.

Window Functions:

Window functions, also known as analytic functions, are a powerful feature in SQL that allows you to perform calculations and analysis over a specific subset of rows within a result set, known as a window. These functions provide a way to aggregate and retrieve data while maintaining the individual row-level details.

Here are some key points to explain window functions:

1. **Syntax**: Window functions have a specific syntax that includes the function call followed by the OVER clause, which defines the window specification. The window specification can include the partitioning clause (PARTITION BY), ordering clause (ORDER BY), and frame clause (ROWS/RANGE).
2. **Partitioning**: The PARTITION BY clause divides the result set into partitions or groups based on specified columns. Window functions are then applied independently to each partition.
3. **Ordering**: The ORDER BY clause determines the order of rows within each partition. It defines the logical sequence for calculations performed by window functions.
4. **Frame**: The frame clause defines the range of rows included in the window. It specifies the relative position of rows based on the ordering clause and determines which rows are included in the calculation.
5. **Aggregation and Ranking**: Window functions can perform various calculations, including aggregation functions such as SUM, AVG, MIN, MAX, and COUNT. They can also generate rankings, such as ROW_NUMBER, RANK, and DENSE_RANK, which assign a unique number or rank to each row within the window.
6. **Moving Averages and Cumulative Totals**: Window functions can calculate moving averages, cumulative totals, and other rolling or running calculations by using the frame clause to specify a range of preceding or following rows.
7. Window Functions Examples: Here are a few examples of window functions:
    - **ROW_NUMBER ()**: Assigns a unique number to each row within the window.
    - **SUM ()**: Calculates the sum of a column within the window.
    - **RANK ()**: Assigns a rank to each row based on the column value.
    - **LAG () and LEAD ()**: Accesses the values of a column from the previous or next row within the window.

Window functions are beneficial for performing complex analytical operations and gaining insights from data. They eliminate the need for self-joins or subqueries, making queries more efficient and readable. Including examples and use cases in your document will help illustrate the versatility and practicality of window functions in SQL.