Data modeling

Data modeling is like making a plan or a map for how information is organized in a computer system. Imagine you have a bunch of different pieces of information (like names, ages, and addresses of people), and you need to figure out how to store them in a computer so you can easily find and use them later.

So, data modeling helps you figure out:

1. What kinds of information you have (like people and their details).
2. How these different pieces of information are related to each other (for example, each person has a name, an age, and an address).
3. How you'll store this information in the computer (like using tables and columns in a database).
4. How to make sure the information is organized efficiently, so you can find and update it quickly and accurately.

Think of it like drawing a blueprint before building a house. You plan out where each room will be, how they'll connect, and what materials you'll use. Similarly, data modeling helps you plan out how your data will be organized and stored in a computer system before actually building it

Normalization:

Imagine you have a list of people and their addresses, but some people live together. Instead of writing down the address for each person every time, you could just write it down once and then say which people share that address.

Normalization in SQL is kind of like that. It's a way of organizing your data in a database to avoid repeating the same information over and over again. Instead, you break your data into smaller tables and link them together using relationships. This helps keep your data tidy and prevents mistakes or inconsistencies.

Star Schema

Imagine you have a big table of sales data. In this data, you have information about products, customers, sales dates, and sales amounts. Now, to understand this data better, you might want to organize it in a way that's easy to analyze.

So, instead of having all the information in one big table, you could split it into smaller tables. In a star schema, you have one big central table called the "fact table." This table holds the main data you're interested in, like sales amounts.

Then, you have smaller tables surrounding the fact table called "dimension tables." Each dimension table holds details about a specific aspect of the data, like products, customers, or dates. These dimension tables are connected to the fact table.

The fact table sits in the middle, like the center of a star, with the dimension tables branching out from it like the arms of the star. Hence, it's called a star schema.

This setup makes it easier to analyze the data because you can focus on the main facts (in the fact table) and then use the dimension tables to add more details as needed. It's like having a central point of reference (the fact table) with different dimensions (dimension tables) branching out from it to give you a clearer picture of your data.

ACID transactions:

Imagine you're transferring money from one bank account to another. Now, there are a few things you want to make sure happen correctly:

1. **Atomicity:** You want the transfer to either happen completely or not at all happen. You wouldn't want the money to be withdrawn from one account but not deposited into the other.
2. **Consistency:** You want both accounts to remain in a valid state throughout the transfer. For example, you wouldn't want one account to have less money while the other account doesn't receive the transferred amount.
3. **Isolation:** While the transfer is happening, you don't want any other transactions to interfere with it. You wouldn't want someone checking the balance at the same time as you're transferring money and getting incorrect information.
4. **Durability:** Once the transfer is complete, you want to ensure that its changes are permanently saved and cannot be lost, even in the face of problems like power outages or system crashes. You wouldn't want the transfer to disappear if something unexpected happens.

So, ACID transactions are like a set of rules that make sure transactions (like transferring money) happen correctly, are reliable, and leave your accounts in a good state, no matter what happens. They help maintain the integrity and reliability of your data in a database.

Select, insert, update & delete (DML and DQL):

These are essential commands in SQL for interacting with databases:

DQL - Data Query Language:

1. SELECT to retrieve data

DML - Data Manipulation Language:

1. INSERT to add new data
2. UPDATE to modify existing data
3. DELETE to remove unwanted data.

Join operations

Join operations are like combining information from different sources or tables in a database to get a more complete picture.

1. **Inner Join:** This type of join returns only the rows that have matching values in both tables.

2. **Left Join (or Left Outer Join):** This type of join returns all the rows from the left table and the matched rows from the right table. If there's no match, it returns NULL values for the columns from the right table.
3. **Right Join (or Right Outer Join):** This type of join returns all the rows from the right table and the matched rows from the left table. If there's no match, it returns NULL values for the columns from the left table.
4. **Full Join (or Full Outer Join):** This type of join returns all the rows from both tables, matching them where possible. If there's no match, it returns NULL values for the columns from the table with no match.

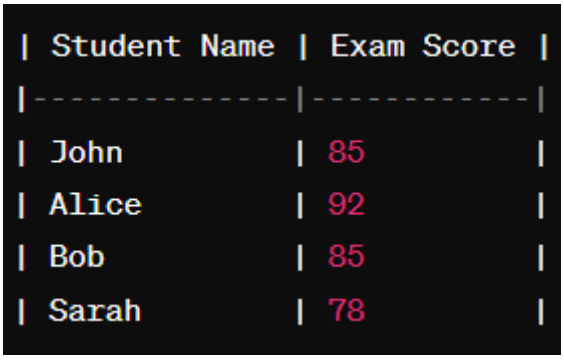
Window functions (rank, dense rank, row number

Window functions are like tools you can use to analyze data in specific ways without affecting the overall structure of your data. They operate within a "window" of rows from a query result, allowing you to perform calculations or assign rankings based on that window.

1. Rank:
    - Rank is like giving positions to rows based on certain criteria.
    - For example, if you're ranking students based on their exam scores, the student with the highest score gets rank 1, the next highest score gets rank 2, and so on. If two students have the same score, they would both get the same rank, and the next student would get the rank after theirs.
2. Dense Rank:
    - Dense rank is similar to rank, but it assigns consecutive ranks without any gaps.
    - For example, if two students have the same score and get rank 1, the next student would get rank 2, not rank 3. This means there are no gaps in the ranking sequence.
3. Row Number:
    - Row number is like giving each row a unique number within the window.
    - It simply assigns a sequential number to each row, starting from 1.
    - It's useful for identifying individual rows or for creating unique identifiers within a result set.

These functions are often used in SQL queries to perform tasks like ranking data, identifying top performers, or partitioning data into groups for analysis. They allow you to gain insights into your data and make comparisons or decisions based on specific criteria within a defined window of rows.

Example:



Student Name	Exam Score
John	85
Alice	92
Bob	85
Sarah	78

```
SELECT
   "Student Name", "Exam Score",
   RANK() OVER (ORDER BY "Exam Score" DESC) AS "Rank",
   DENSE_RANK() OVER (ORDER BY "Exam Score" DESC) AS "Dense Rank",
   ROW_NUMBER() OVER (ORDER BY "Exam Score" DESC) AS "Row Number"
FROM
   students;
```

Student Name	Exam Score	Rank	Dense Rank	Row Number
Alice	92	1	1	1
John	85	2	2	2
Bob	85	2	2	3
Sarah	78	4	3	4

Data Types, Variables and Constants

Window functions (rank, dense rank, row number

Conditional Structures (IF, CASE, GOTO and NULL)