**Day 3 Notes**

**Name : Podutur Lahari - DE126**

**Date : 06-11-24**

**SQL Server Built-in Functions:**

* SQL Server includes several functions to transform and manage data in queries. These are grouped by purpose, allowing users to manipulate text, dates, numbers, and more within result sets.

**Types of Functions:**

* String Functions: Modify and analyze text data, such as converting case, finding length, and replacing characters. Examples include:
  + ASCII – Returns ASCII code of a character.
  + CHAR – Converts ASCII code to character.
  + LEN – Counts characters in a string.
  + UPPER/LOWER – Converts text to uppercase or lowercase.
* Usage Example: SELECT UPPER('example') converts text to uppercase.

**Date and Time Functions:**

* Handle dates and times, useful for extracting or manipulating specific parts of a date:
  + GETDATE() – Retrieves the current system date.
  + DATEADD() – Adds a specified time interval to a date.
  + DATEDIFF() – Calculates the difference between two dates.
  + YEAR, MONTH, DAY – Extract individual components from a date.
* Usage Example: SELECT YEAR('2024-01-15') returns the year 2024.

**Mathematical Functions:**

* Used for numeric data processing in calculations:
  + ABS – Returns absolute value.
  + CEILING – Rounds up to the nearest integer.
  + FLOOR – Rounds down to the nearest integer.
  + LOG, EXP, POWER – Perform advanced calculations like logarithms and exponentials.
* Usage Example: SELECT FLOOR(15.75) returns 15.

**Ranking Functions:**

* Useful for assigning row numbers or ranking data based on certain conditions:
  + ROW\_NUMBER() – Assigns sequential numbers to rows.
  + RANK() – Assigns rank with possible gaps for duplicates.
  + DENSE\_RANK() – Similar to RANK(), but with no gaps.
  + NTILE(n) – Divides rows into n equal groups.
* Usage Example: SELECT ROW\_NUMBER() OVER (ORDER BY column\_name) ranks rows based on the specified order.

**System Functions:**

* Retrieve system and user-specific information, often for troubleshooting or setup purposes:
  + HOST\_NAME() – Returns the name of the computer.
  + USER\_NAME() – Retrieves current user name.
  + DB\_NAME() – Returns the database name.
* Usage Example: SELECT HOST\_NAME() returns the current system name.

**Aggregate Functions:**

* Summarize columns, calculating values like averages or totals across rows:
  + AVG() – Computes the average value.
  + COUNT() – Counts rows or specific values.
  + MIN(), MAX() – Finds smallest or largest values in a column.
  + SUM() – Adds up values in a column.
* Usage Example: SELECT AVG(salary) FROM employees computes average salary.

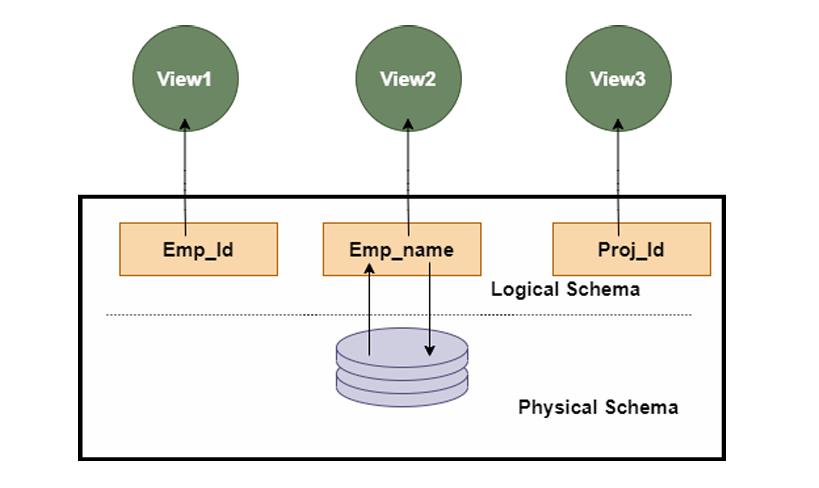
**Grouping and Summarizing Data:**

* Organize data by specified columns and conditions using:
  + GROUP BY – Groups rows that share values in specified columns.
  + COMPUTE and COMPUTE BY – Used for additional data aggregation in older SQL Server versions, where summaries are generated within result sets.
* Usage Example: SELECT department, COUNT(\*) FROM employees GROUP BY department counts employees per department.

**Database Basics:**

* A database is a structured place to store data, ranging from simple lists to complex records.
* Database schema: Logical structure that organizes data, defining tables, attributes, and relationships without holding data itself.

**Types of Database Schema:**

* Physical Schema: Defines data storage on physical devices (files, indices).
* Logical Schema: Defines tables, constraints, and views at the logical level, focusing on relationships and integrity.
* View Schema: Focuses on end-user interactions, showing how users view or interact with the data.

**Key Schema Elements:**

* Primary Key: Uniquely identifies records.
* Foreign Key: Links data between tables, establishing relationships.
* ER Modelling: Visual representation showing relationships among entities (tables).

**Schema vs. Instance:**

* Schema is the unchanging structure of a database.
* Instance is a snapshot of data at a specific point, which can change over time.

**Schema Creation in DBMS:**

* MySQL: CREATE SCHEMA is synonymous with CREATE DATABASE.
* Oracle: CREATE USER command initiates schema creation.
* SQL Server: CREATE SCHEMA creates a new schema by name.

**Popular Schema Models:**

* Flat Model: Simple 2D structure, best for small, non-relational datasets.
* Hierarchical Model: Tree-like structure with a one-to-many relationship (e.g., XML, JSON).
* Network Model: Allows many-to-many relationships, suitable for workflows.
* Relational Model: Most common; stores data in relational tables.
* Star Schema: Optimized for large datasets; splits data into “facts” and “dimensions.”
* Snowflake Schema: Similar to star schema but with additional levels in dimension tables.

**Stored procedures:**

* Definition: A stored procedure is reusable SQL code saved in a database to run specific queries or actions.
* Benefits:
  + Efficiency: Reduces repeated SQL code writing.
  + Parameterization: Accepts parameters, making queries flexible.
  + Security: Controls data access through defined procedures.
* Syntax for Creation:
  + Use CREATE PROCEDURE followed by the procedure name and SQL statement(s).
* Execution:
  + Use EXEC followed by the procedure name to run the stored procedure.
* Example:
  + Procedure to fetch all customer records:
    - CREATE PROCEDURE SelectAllCustomers AS SELECT \* FROM Customers;
    - Run with EXEC SelectAllCustomers;
* Parameterized Procedures:
  + Single Parameter: Allows filtering, e.g., by City.
    - Example: CREATE PROCEDURE SelectAllCustomers @City nvarchar(30) AS SELECT \* FROM Customers WHERE City = @City;
    - Execute with EXEC SelectAllCustomers @City = 'London';
  + Multiple Parameters: Filters by multiple fields, like City and Postal Code.
    - Example: CREATE PROCEDURE SelectAllCustomers @City nvarchar(30), @PostalCode nvarchar(10) AS SELECT \* FROM Customers WHERE City = @City AND PostalCode = @PostalCode;
    - Execute with EXEC SelectAllCustomers @City = 'London', @PostalCode = 'WA1 1DP';

**Calculating subtotals in SQL queries:**

* Definition: Subtotals provide the sum of specific grouped data without displaying a grand total. They are often used in reports, receipts, and financial documents.
* SQL Extensions for Subtotals:
  + **ROLLUP**: Adds hierarchical subtotal rows and a grand total row based on specified columns.
    - Example: GROUP BY ROLLUP (Year, Quarter) provides subtotals by year and quarter.
  + **GROUPING**: Identifies aggregated columns in a group, useful for labeling subtotal and grand total rows.
* GROUPING SETS:
  + An alternative to ROLLUP, enabling more flexible grouping structures and multi-level subtotals within one query.
* Practical Uses:
  + Monthly and Quarterly Summaries: By using extensions, we can summarize data by specific timeframes, e.g., by month or quarter, within a single dataset.
  + Filtering: By combining GROUPING with HAVING, we can exclude unwanted rows from the results.
* Applications: These SQL techniques help generate dynamic, organized financial or sales reports by calculating subtotals across varying categories and hierarchies.

**SQL order of execution:**

* SQL order of execution refers to the sequence in which the database processes different parts of a query to retrieve accurate results.
* Importance:
  + Ensures accuracy of query results.
  + Helps in query optimization for faster performance.
  + Reduces resource usage on the database server.
  + Simplifies troubleshooting and modifying complex queries.
* **Order of Execution:**
  + FROM: Identifies tables and accesses necessary data.
  + WHERE: Filters data based on specified conditions.
  + GROUP BY: Groups data and applies aggregation functions.
  + HAVING: Filters grouped data based on conditions.
  + SELECT: Chooses columns for the final result.
  + ORDER BY: Sorts the result set.
  + LIMIT/OFFSET: Limits the number of rows in the output.
* **Application:**
  + Understanding execution order aids in building efficient queries and enhances readability, especially in complex queries with multiple clauses.