**SQL**

**Rows**: records

**Columns**: fields

**Datatypes**

* bigint: used to store large integer values
* int: used to store small integer values ranges from -2147483648 to 2147483648
* smallint: used to store small integer ranges from -32768 to 32767
* tinyint: ranges from 0 to 255
* decimal(s,d): used to store fractional values
* char: used to store string ranges till 255 characters
* varchar: used to store variable character
* ranges till 255 characters
* text: used to store string ranges till 65535
* date: YYYY-MM-DD
* time: HH:MM:SS
* Year: YYYY

**Constraints**

* Not Null: ensures that every column or field contains values
* Default: used to assign default constraint
* to each columns
* Unique: used to ensure all the values in the columns are different from each other
* Primary key: combination of not null and unique
* **Creating table:**

CREATE TABLE table\_name(

column1 datatype,

column2 datatype,

column3 datatype,

..

columnN datatype,

PRIMARY KEY(columns\_x) );

A table can have only one primary key

* **Inserting into table:**

INSERT INTO table\_name

VALUES(value1, value2, value3, … valueN);

* **Selecting data from the table:**

SELECT columns1, column2,…, columnN

FROM table\_name;

* **Selecting entire table:**

SELECT \* from employee;

* **Selecting only distinct values from table:**

SELECT DISTINCT column1, column2

FROM table\_name;

* **Where clause:** used to extract only those records that fulfill a specified condition

SELECT column1, column2, ….,columnN

FROM table\_name WHERE[condition];

* **Selecting data using logical operators**

1. **AND:**

SELECT column1, column2,…, columnN

FROM table\_name WHERE[condition1]

AND [condition2]…AND[conditionN];

1. **OR:**

SELECT column1, column2,…, columnN

FROM table\_name WHERE[condition1]

OR[condition2]…. OR[conditionN];

1. **NOT:** selects data where the condition is not true

SELECT column1, column2,…, columnN

FROM table\_name WHERE NOT[condition1];

**Functions**

* MIN(): gives smallest value in the column

SELECT MIN(column)FROM table\_name;

* MAX(): gives maximum value in the column

SELECT MAX(column) from table\_name;

* COUNT(): returns the number of rows that match a specific criteria

SELECT COUNT(\*) FROM table\_name

WHERE condition;

* SUM(): returns the total sum of numeric column

SELECT SUM(col\_name) FROM table\_name;

* AVG(): gives the average values of a numeric column

SELECT AVG(col\_name) FROM table\_name;

* LTRIM(): removed blanks on the left side of string

SELECT LTRIM(‘ string’) from table\_name;

* LOWER(): converts all characters to lower case letters

SELECT LOWER(‘STRING’) from table\_name;

* UPPER(): converts all characters to upper case letters

SELECT UPPER(‘string’) from table\_name;

* REVERSE(): reverses all the characters in the string

SELECT REVERSE(‘string’) from table\_name;

* ORDER BY(): used to sort data in ascending or descending order

SELECT column FROM table\_name

ORDER BY column1, column2,…., columnN ASC | DESC

* TOP(): used to fetch the top N records

SELECT TOP c column\_list FROM table\_name;

* GROUP BY(): groups rows that have the same values into summary rows, used to get aggregate result with respect to the group

SELECT column FROM table\_name WHERE[condition] GROUP BY column

ORDER BY column

* UPDATE(): used to modify the existing records in a table

UPDATE table\_name

SET col1=val1, col2=val2,… colN=valN

WHERE[condition];

* DELETE(): used to delete existing records in the table

DELETE FROM table\_name

WHERE[condition];

* TRUNCATE(): deletes all the data from the table

TRUNCATE TABLE table\_name;

* INNER JOIN(): used to return records that have matching values in both tables

SELECT columns

FROM table1

INNER JOIN table2

ON table1.column\_x=table2.column\_y;

* LEFT JOIN(): returns all the record from the left table and the matched records from the right table

SELECT columns

FROM table

LEFT JOIN table2

ON table1.column\_x= table2.column\_y;

* RIGHT JOIN(): returns all the records from the right table and the matched records from the left table

SELECT columns

FROM table1

RIGHT JOIN table2

ON table1.column\_x = table2.column\_y;

* FULL JOIN(): returns all the rows from the Left table and right table with Null values in place where the condition is not met

SELECT columns

FROM table1

FULL JOIN table2

ON table1.column\_x = table2.column\_y

* UNION Operator: used to combine the result set of two or more SEECT statements

SELECT column\_list FROM table1

UNION

SELECT column\_list FROM table2

* UNION ALL operator: gives all the rows from both the tables including the duplicates
* EXCEPT operator: combines two select statements and returns unique records from the left query which are not part of the right query

SELECT column\_list FROM table1

EXCEPT

SELECT column\_list FROM table2

* INTERSECT OPERATOR: combines two select statements and returns the records which are common to both the select statements

SELECT column\_list FROM table1

INTERSECT

SELECT column\_list FROM table2

**VIEWS**: It is a virtual table based on the result of an sql statement

CREATE VIEW view\_name AS

SELCT column1, column2

FROM table\_name

WHERE condition;

**ALTER TABLE** : statement used to add, delete or modify columns in a table

**ADD column**:

ALTER table

ADD column\_name datatype;

**DROP column**:

ALTER TABLE table\_name

DROP column\_name;

**MERGE TABLE**: joins two target tables to the source table using common column in both tables

MERGE [TARGET] AS T

USING [SOURCE] AS S

ON [JOIN CONDITION]

WHEN MATCHED

THEN [Update statement]

WHEN NOT MATCHED BY TARGET

THEN [Insert statement]

WHEN NOT MATCHED BY SOURCE

THEN [Delete statement];

**TYPES OF USER DEFINED FUNCTION:**

1. **SCALAR VALUED:** Always return a scalar value

CREATE FUNCTION function\_name(@param1 data\_type, @param2 data\_type…)

RETURNS return\_datatype

AS

BEGIN

——function body

RETURN value

END

1. **TABLE VALUED**: returns a table instead of scalar

CREATE FUNCTION function\_name(@param1 data\_type, @param data\_type…)

RETURNS table

AS

RETURN(SELECT column\_list FROM table\_name WHERE[condition])

**Temporary tables**: it helps to store and process intermediate results and are created in the tempDBM and automatically deleted when no longer in use, it could be very useful in cases where ever we need to store temporary data

CREATE TABLE #table\_name();

**COMMON TABLE EXPRESSION:** A Common Table Expression (CTE) is a temporary named result set that you can reference within a SELECT, INSERT, UPDATE, or DELETE statement. It provides a way to define a temporary result set that can be used and referred to multiple times within a single query. CTEs are defined within the scope of a single statement and are not visible outside of that statement. They can be used to simplify complex queries, break down a large query into smaller, more manageable parts, or to create recursive queries. CTEs are defined using the WITH keyword followed by the CTE name, column names (optional), and the query that defines the CTE. The CTE can then be referenced in the main query using its defined name.

WITH CTE\_Name (Column1, Column2, Column3) AS (

SELECT Column1, Column2, Column3

FROM TableName

WHERE Condition

)

SELECT \*

FROM CTE\_Name;

**DATE RANGES:** In the context of the previous conversation, date\_ranges is a Common Table Expression (CTE) that is used to generate a sequence of date ranges. It starts with an initial date range (start\_date and end\_date) and then recursively generates subsequent date ranges by adding a specific interval (in this case, one month) to the previous start\_date and end\_date. The date\_ranges CTE is used to define a sequence of date ranges that are used in the main query to retrieve data based on those date ranges. It helps in organizing the query and generating the necessary date ranges for further processing or analysis.

**CASE STATEMENT:** works so the case statement goes through conditions and returns a value when the first condition is met

CASE

WHEN condition1 THEN result1

WHEN condition1 THEN result2

..

WHEN conditionN THEN resultN

ELSE result

END;

**IIF function**: is an alternate of case expression, it peeks in a boolean expression as the first parameter and returns the true value if the boolean expression is evaluated to true or returns the false value if the boolean expression is evaluated to false

**STORED PROCEDURE**: is a prepapred sql code which can be saved and reused

CREATE PROCEDURE procedure\_name

AS

sql\_statement

GO;

EXEC procedure name

**TRIGGERS:** it is a special type of stored procedure that occurs automatically when an event occurs in the database server

Types of triggers:

1. Data manipulation trigger
2. Data termination trigger
3. Logon trigger

CREATE TRIGGER trigger\_name

(Before | After)

[Insert | Update | Delete]

On [table \_name]

[for each row | for each column]

[Trigger\_body]

**NESTED TRIGGERS:** they are multiple actions that are automatically executed when a certain database operation is performed

**EXCEPTION HANDLING:** an error condition during a program execution is called as an exception and the mechanism for resolving such an exception is known as exception handling. SQL provides try/catch blocks for exception handling

Try statement allows us to test a block of statement while being executed and the catch block allows us to handle any exception which occurs in the try

BEGIN TRY

SQL statement

END TRY

BEGIN CATCH

-Print Error OR

-Roll back Transaction

END CATCH

**TRANSACTION:** It is a group of commands that change data stored in a database and a transaction is treated as a single unit, it ensured that either all of the commands are executed successfully or none of them

**SUBQUERIES:** they are queries within another SQL that is embedded within WHERE clause From clause or HAVING clause

**DATABASE ADMINISTRATOR**: a database administrator is someone who directs or performs all activities related to maintaining a successful database environment

**Roles of database administrator**:

1. Software Installation and Maintenance

2. Taking care of Database backup and recovery

3. Maintaining security of database

4. Take care of access control for different users

5. Monitor databases for performance issues

**Types of database administrator:**

1. Production DBA

2. Application DBA

3. Development DBA

4. UAT DBA

5. Data warehouse DBA

**MASTER DATABASE**: the master database records all the system level information for a SQL Server System. This includes instance wide metadata such as logon accounts, endpoints, linked servers and system configuration settings

**MODEL DATABASE**: the model database is used as a template whenever a new user database is created. You can change most database properties, create users, stored procedures, tables, views, etc whatever you do will be applied to any new databases

**MSDB DATABASE**: the msdb database is used by SQL server agent for schedulling alerts and jobs and by other features such as SQL Server Management Studio, Service Broker and Database Mail

**TEMPDB DATABASE**: tempdb is the workspace for holding temporary objects or intermediate results of query processing and sorting, it is recreated everytime the SQL server service is started and any object that we have created in tempdb will be gone once the server restarts

**EDITIONS OF SQL SERVER**

1. Enterprise Edition

2. Standard Edition

3. Web Edition

**BACKUP AND RESTORE**: one of the most important tasks of database administrator is to keep backup and restore the databases to make sure there is no data loss. The system databased that you must always back up includes msdb, master and model. It minimizes the risk of catastrophic data loss. We need to backup our databases to preserve modifications to a date on a regular basis and this is where a well planned backup and restore strategy helps in protecting the database against dataloss caused by a variety of failures

**RECOVERY MODELS**: a recovery model is a database property that controls how transactions are logged, whether the transaction log required backing up, and what kinds of restore operations are available and all of the backup restore and recovery options are based on of these three available recovery models that are simple, full and bulk load recovery

**1.Simple recovery model**: it maintains only minimum amount of information in the SQL Server transaction log file. With simple recovery model the data is recoverable only to the most recent full database of differential backups. Here the transaction log truncation happens after a checkpoint or as soon as you can change the recovery model of your database to simple recovery model.

**2.Full recovery model**: It logs every transaction and maintains it there until a transaction log backup is taken. With this model you can devise a disaster recovery plan that includes a combination of full backup and transaction log backups. You can recover to an arbitrary point in point from transaction log backups and hence no work is lost due to damaged data files

**3.Bulk logged recovery model**: with this model there are certain bulk operations such as BULK INSERT, CREATE INDEX, SELECT INTO, etc that are not fully logged in the transaction log and therefore do not take as much space in the transaction log. It can be used when you run batch jobs or during time when you perform routine maintenance. It reduces log space usage by using minimal logging for most bulk operations

**SIMPLE:**

USE[master]

GO

ALTER DATABASE

[database\_name]

SET RECOVERY SIMPLE WITH

NO\_WAIT

GO

**FULL:**

USE[master]

GO

ALTER DATABASE

[database\_name]

SET RECOVERY FULL WITH

NO\_WAIT GO

**BULK LOGGED:**

USE[master]

GO

ALTER DATABASE

[database\_name]

SET RECOVERY BULK\_LOGGED WITH

NO\_WAIT

GO

**PLANNING A BACKUP STRATEGY**: After you have selected a recovery model that meets your business requirements for a specific database, you have to plan and implement a corresponding backup strategy

**TYPES OF BACKUP MODELS**

1.Full backups: backs up the whole database

2.Differential backups: copies all the data that has changed since the last full backup

3.File backups: when a database size and performance requirements make a full database impractical, you can create a file backup instead

4.Filegroup backups: when a database size and performance requirements make a full database impractical, you can create a file group backup instead

5.Partial backups: resembles a full database backup, except that it does not contain all the filegroups, relevant to databased that contain read only files

6.Copy only backups: is an independent backup and doest not break the chain of database backups

7.Mirror backups: are actually identical copies of the same backup

8.Transaction log backups: allows you to backup the active part of the transaction

**POINT IN TIME RECOVERY**: it allows to restore a database into a state it was in any point of time. This type of recovery is applicable only to databases that run under the full of bulk logged recovery model

**DYNAMIC MANAGEMENT VIEWS**: they return server state information that can be used to monitor the health of a server instances, diagnose problems and tune performances. They return metadata of the system state. They are actually composed of both views and table valued function. All dynamic management views and functions exist in the sys schema and follow this naming convention dm\_\*

**ACTIVITY MONITOR**: it displays information about SQL server processes and how these processes affect the current instance of SQL server. Activity monitor is a tabbed document window with panes that are Overview, processes, resource waits, data file I/O, recent expensive queries and active expensive queries

**PERFORMANCE MONITOR**: sometimes there are issues outside of SQL server that may be causing performance issues and this is where we can use performance monitor. With the help of this we can view SQL server objects, performance counters and the behaviour of other objects such as processors, memory, cache and threads

**TRACING SQL SERVER ACTIVITY**: when working with long and complex SQL queries, it becomes extremely important to monitor them and do a thorough and accurate analysis of all the task, so that it becomes easy to troubleshoot the data. This is where we can trace all the ongoing tasks and understand about the exceptions properly

**SQL SERVER PROFILER**: it is a tool for tracing, recreating and troubleshooting problems in MS SQL server. The profiler lets developers and database administrator create and handle traces and replay and analyze trace results. It shows the health pf an instance

**DATA INTEGRITY**: it is used to maintain accuracy and consistency of data in a table

Categories of data integrity:

**1.Entity integrity**: ensures that each row in a table is uniquely identifiable entity

**2.Referential integrity**: ensures the relationship between two tables is proper and we can apply this using a foreign key constraint

**3.Domain integrity**: ensures data values in a database followed defined rules for values range and format and we can enforce these rules using check and default constraint

**4.User defined data integrity**: it can be enforced through triggers

**INDEX**: it is an on disk structure associated with a table or view that speeds retrieval of rows from the table or view. An index contains keys built from one or more columns in the table or view

**FRAGMENTATION**: as data is modified in a database, the database and its indexes become fragmented. As indexes become fragmented, ordered data retrieval becomes less efficient and reduced database performance