Key-Note OF MS SQL

SQL>>> It's stands for

Structured Query Language (which is a standard language for accessing & manipulating data)

SQL Commands divided into 4 categories.

1. Data Query Language (Select)
2. Data Definition language (Create table, Alter Table, Drop Table)
3. Data Manipulation language (Insert, Update, Delete)
4. Date Control language (Grant, Revoke)

How to Set MS SQL>>

1. How to Install Microsoft SQL Server 2022 & SSMS - Complete guide | Microsoft SQL Server 2022 - YouTube
2. How to Download and Install SQL Server for Windows
3. SQL Server installation guide - SQL Server | Microsoft Learn

SQL Terminologies>>

SQL Table>>

A Table is a database object which comprises of rows and columns. (Rows know as a Records and Columns knows as a Field)

Syntaxes>>

Command	Query	Note
Create	CREATE DATABASE database_name;	
Use	USE database_name;	
Drop	DROP DATABASE database_name;	
Create Table	CREATE TABLE table_name (column1 datatype, column2 datatype…. columnN datatype, PRIMARY KEY(column_x));	
Insert	INSERT INTO table_name VALUES (value1, value2…. valueN);	
Select	SELECT column1, column2…. columnN FROM table_name;	

	(For retrieve all the data from the table use “*” operator,) SELECT * FROM table_name;	
Select distinct	SELECT DISTINCT column1, column2…. columnN FROM table_name;	It’s used to select distinct values from our Columns.
Where	SELECT column1, column2…. column FROM table_name WHERE [condition];	It’s used to extract records which satisfy a condition.
UPDATE	UPDATE table_name SET col1 = val1, col2 = val2… [WHERE condition];	It’s used to modify the existing records in table.
DELETE	DELETE FROM table_name [WHERE condition];	It’s used to delete existing records in the table.

TRUNCATE	TRUNCATE TABLE table_name;	It's used to delete all the data inside the table.
VIEW	CREATE VIEW view_name AS SELECT column1, column2,… FROM table_name WHERE condition; (CREATE VIEW female_employees AS SELECT * FROM employee WHERE e_gender = "Female" ;)	It's a virtual table based on the result of an SQL statement.
DROP	DROP VIEW view_name;	
Alter Table>>		It's used to add, delete or modify columns in a table.
Alter Add	ALTER TABLE table_name ADD column_name datatype; (ALTER TABLE employee ADD e_dob DATE;)	
Alter Drop	ALTER TABLE table_name DROP COLUMN column_name; (ALTER TABLE employee DROP COLUMN e_dob;)	

MERGE>>	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]; (MERGE employee_target AS T USING employee_source AS S ON T.e_id = S.e_id WHEN MATCHED THEN UPDATE SET T.e_salary = S.e_salary, T.e_age = S.e_age WHEN NOT MATCHED BY TARGET THEN INSERT (e_id, e_name, e_salary, e_gender, e_dept) VALUES (S.e_id, S.e_name, S.e_salary, S.e_age, S.e_gender, S.e_dept) WHEN NOT MATCHED BY SOURCE THEN DELETE;)	It's the combination of INSERT, DELETE and UPDATE statements. For this we required two tables Source table and Target table.
Type of user defined Functions>>		
Scalar valued	CREATE FUNCTION function_name (@param1 data_type, @param2 data_type…) RETURNS return_datatype AS BEGIN ---Function body	it's always return a scalar value. For parameters use '@'.

	RETURN value END (CREATE FUNCTION add_five (@num as int) RETURNS int AS BEGIN RETURN (@num+5) END Call the Function: SELECT dbo.add_five(10) OUTPUT: 15)	
Table valued	CREATE FUNCTION function_name (@param1 data_type, @param2 data_type…) RETURNS TABLE AS RETURN (SELECT column_list FROM table_name WHERE [condition]) (CREATE FUNCTION select_gender(@gender AS VARCHAR(20)) RETURNS TABLE AS RETURN (SELECT * FROM employee WHERE e_gender = @gender)	It's returns a table

	Call the Function: SELECT * FROM dbo.select_gender('female'); SELECT * FROM dbo.select_gender('male');)	
Case statement	CASE WHEN condition1 THEN result1 WHEN condition2 THEN result2 WHEN conditionN THEN result ELSE result END; (SELECT CASE WHEN 10>20 THEN '10 is greater than 20' WHEN 10<20 THEN '10 IS less than 20' ELSE '10 is equal to 20' END SELECT *, grade= CASE WHEN e_salary<=90000 THEN 'C' WHEN e_salary<=120000 THEN 'B' ELSE 'A' END FROM employee GO ;)	It's helps in multi way decision making.
IIF() function	IIF(boolean_expression, true_value, false_value); (SELECT IIF(10>20, '10 is greater than 20', '10 is less than 20'); On employee table:	It's function an alternative for the case statement.

	SELECT e_id,e_name,e_age, IIF(e_age>30, 'Old employee', 'young employee') AS employee_generation FROM employee;)	
Join>>		
Inner Join	SELECT columns FROM table1 INNER JOIN table2 ON table1.column_x =table2.column_y; (SELECT employee.e_name, employee.e_dept, department.d_name, department.d_location FROM employee INNER JOIN department ON employee.e_dept = department.d_name;	It's returns records that have matching values in both the tables. It's also known as simple join.
Left Join	SELECT columns FROM table1 LEFT JOIN table2 ON table1.column_x = table2.column_y; (select employee.e_name, employee.e_dept, department.d_name, department.d_location FROM employee LEFT JOIN department NO employee.e_dept = department.d_name;	It's returns all the records from the left table, and the matched records from the right table.
Right Join	SELECT columns FROM table1 RIGHT JOIN table2 ON table1.column_x = table2.column_y;	It's returns all the records from the right table and the

	(SELECT employee.e_name, employee.e_dept, department.d_name, department.d_location FROM employee RIGHT JOIN department ON employee.e_dept = department.d_name;)	matched records from the left table.
Full Join	SELECT columns FROM table1, FULL JOIN table2 ON table1.column_x = table2.column_y; (SELECT employee.e_name, employee.e_dept, department.d_name, department.d_location FROM employee FULL JOIN department ON employee.e_dept = department.d_name;)	it's returns all rows from the LEFT table and The RIGHT table with NULL values in place where the join condition is not met.
Update Using with Join	UPDATE employee set e_age = e_age + 10 FROM employee JOIN department ON employee.e_dept = department.d_name WHERE d_location = "New York";	
Delete Using with Join	DELETE employee FROM employee JOIN department ON employee.e_dept = department.d_name WHERE d_location = "New York";	
Operator>>		
AND Operator	SELECT column1, column2…. column FROM table_name WHERE [condition1] AND	It's displays records if all the conditions separated by

	[condition2] …. AND [conditionN];	AND are TRUE.
OR Operator	SELECT column1, column2…. column FROM table_name WHERE [condition1] OR [condition2] …. OR [conditionN];	It's displays records if any of the conditions separated by OR is TRUE.
NOT Operator	SELECT column1, column2…. column FROM table_name WHERE NOT [condition];	It's displaying a record if the condition is 'NOT TRUE'.
Like	SELECT col_list FROM table_name WHERE column_N LIKE '_xxxx%'; (SELECT * FROM employee WHERE e_name LIKE 'J%'; OR SELECT * FROM employee WHERE e_Age LIKE '3_';)	It's used to extract records where a particular pattern is present. It's used is congestion.
Between	SELECT col_list FROM table_name WHERE column_N BETWEEN val1 AND val2;	It's used to select values within a given range.

	(SELECT * FROM employee WHERE e_salary BETWEEN 90000 AND 120000;)	
UNION	SELECT column_list FROM table1 UNION SELECT column_list FROM table2; (SELECT * FROM student_details1 UNION SELECT * FROM student_details2;)	It's used to combine the result-set of two or more SELECT statements.
UNION ALL	SELECT column_list FROM table1 UNION ALL SELECT column_list FROM table2; (SELECT * FROM student_details1 UNION ALL SELECT * FROM student_details2;)	It's used to gives all the rows from both the tables including the duplicates.
EXCEPT	SELECT column_list FROM table1 EXCEPT SELECT column_list FROM table2; (SELECT * FROM student_details1 EXCEPT SELECT * FROM student_details2;)	It's combines two select statements and returns unique records from the left query which are not part of the right query.
INTERSECT	SELECT column_list FROM table1 INTERSECT SELECT column_list FROM table2;	It's helps to combine two select

	(SELECT * FROM student_details1 INTERSECT SELECT * FROM student_details2;)	statements and return the records which are common to both the select statements.
Wild Card Characters		
"%"	Percentage symbol	Represent zero, one or multiple characters.
"_"	Underscore symbol	Represents a single character.
Functions		
Aggregate functions >>		
MIN()	SELECT MIN(col_name) FROM table_name;	It's gives as the smallest value.
MAX()	SELECT MAX(col_name) FROM table_name;	It's gives as the largest value.
COUNT()	SELECT COUNT(*) FROM table_name WHERE condition;	It's returning the number of rows that

		match's a specific criteria.
SUM()	SELECT SUM(col_name) FROM table_name;	It's gives the total sum of a numeric column.
AVG()	SELECT AVG(col_name) FROM table_name;	It's gives the average value of a numeric column.
Clauses >>		
ORDER BY	SELECT colomn_list FROM table_name ORDER BY col1, col2,….ASD/DESC; (SELECT * FROM employee ORDER BY e_salary; Or SELECT * FROM employee ORDER BY e_salary DESC;)	it's used to sort the data in ascending or descending order. by default it's ASD to change that DESC used.
TOP	SELECT TOP x column_list FROM table_name; (SELECT TOP 3 * FROM employee;	It's used to fetch the TOP-N records.

	Or SELECT TOP 3 * FROM employee ORDER BY e_Age DESC;)	
GROUP BY	SELECT column_list FROM table_name WHERE condition GROUP BY colname(s) ORDER BY colname(s); SELECT column_list FROM table_name GROUP BY colname(s); (SELECT AVG(e_salary), e_gender FROM employee GROUP BY e_gender; Or SELECT AVG(e_Age), e_dept FROM employee GROUP BY e_dept ORDER BY AVG(e_Age)DESC;)	It's used to get aggregate result with respect to group. Here careful about sequence.
HAVING	SELECT column_name(s) FROM table_name WHERE condition column_name(s) HAVING condition ORDER BY column_name(s); (SELECT e_dept, AVG(e_salary) AS avg_salary FROM employee GROUP BY e_dept HAVING avg(e_selary) > 100000;)	It's used in combination with Group By to impose conditions on groups.

String Functions >>		
LTRIM()	SELECT LTRIM('string');	Removes blanks on the left side of the character expression.
LOWER()	SELECT LOWER('string');	Converts all characters to lower case letters.
UPPER()	SELECT UPPER('string');	Converts all characters to upper case letters.
REVERSE()	SELECT REVERSE('string');	Reverses all the characters in the string.
SUBSTRING()	SELECT SUBSTRING('string',start index, end index);	Gives a substring from the original string.
Temporary Table	CREATE TABLE #table_name(); (CREATE TABLE #student(s_id int , s_name varchar(20));	They are created in tempDB and deleted as

	SELECT * FROM #student;)	soon as the session is terminated. Whenever we make or use temporary table use '#' to target or follow that.
Stored Procedure	CREATE PROCEDURE procedure_name AS sql_statement GO; EXEC procedure_name (CREATE PROCEDURE employee_age AS SELECT e_age FROM employee GO; EXEC employee_age	It's a prepared SQL code which can be saved and reused.
Stored Procedure with parameter syntax	CREATE PROCEDURE procedure_name @param1 data-type, @param2 data-type AS sql_statement GO; (CREATE PROCEDURE employee_gender @gender varchar(20) AS SELECT * FROM employee where e_gender = @gender GO; exec employee_gender @gender = 'Male';	

Try/Catch	BEGIN TRY SQL statements END TRY BEGIN CATCH print error OR rollback transaction END CATCH (DECLARE @val1 INT; DECLARE @val2 INT; BEGIN TRY SET @val1=8; SET @val2-@val1/0; END TRY BEGIN CATCH PRINT error_message() END CATCH Second Example>> BEGIN TRY SELECT e_salaryee_name FROM employee END TRY BEGIN CATCH	An error condition during a program execution is called an exception. The mechanism for resolving such an exception is exception handling SQL Provides the try/ catch blocks for exception handling.

Data type of SQL>>

(Data Types define what type of data a column can hold)

Data type	Range	Note
	Numerical Data Type	

Bigint	-9223372036854775808 ←→ +9223372036854775807	
Int	-2147483648 ←→ +2147483647	
Smallint	-32768 ←→ +32767	
Tinyint	0 ←→ 255	
Decimal(s,d)	-10^38+1 ←→ 10^38-1	
	Character Data Types	
Char(s)	255 characters	
Varchar(s)	255 characters	
Text	65535 characters	
	Date & Time Data Types	
Date	YYYY-MM-DD	
Time	HH:MM: SS	
Year	YYYY	

Constraints in SQL>>

(Constraints are used to specify rules for data in table)

- Not Null:

  It's used to ensures that a column cannot have a NULL value.

- Default:

It’s used to sets a default value for a column when no value is specified.

- Unique:

  It’s ensuring that all values in a column are different.

- Primary Key:

  It’s constraint uniquely identifies each record in a table. (Not Null + Unique)

Temporary Table>>

It’s created in tempDB and deleted as soon as the session is terminated.

Stored Procedure>>

It’s a prepared SQL code which can be saved and reused.

Exception handling>>

An error condition during a program execution is called an exception.

The mechanism for resolving such an exception is exception handling.

We can do this by using “try”, “catch”.

Useful Videos>>

- SQL Training | SQL Tutorial | Intellipaat - YouTube