MS SQL Server

TUTORIAL

ishwar.academy

Website

IA

1 MS SQL Server Installation

Click here to watch Video

2 SSMS Installation

3 Create Database

IA

MS SQL Server

TUTORIAL

4 **SQL Data types**

DATA TYPES

- **Numeric**
  - INTEGER
  - DECIMAL
  - NUMERIC
  - SMALLINT
  - BIGINT
  - TINYINT
  - REAL
  - FLOAT
  - MONEY
  - SMALLMONEY
- **Character**
  - CHAR
  - VARCHAR
  - NCHAR
  - NVARCHAR
- **Temporal**
  - DATE
  - TIME
  - DATETIME
  - SMALLDATETIME
  - DATETIME2
  - DATETIMEOFFSET
- **Miscellaneous**
  - BIT
  - SQL_VARIANT
  - UNIQUEIDENTIFIER
  - XML
  - TABLE
  - Binary Data Types
  - Large Object Data Types

IA

MS SQL Server

TUTORIAL

5 **Integrity Constraints**

INTEGRITY CONSTRAINTS

- ✓ A set of rules
- ✓ used to maintain the quality of information.
- ✓ ensure that the data insertion, updating, and other processes have to be performed in such a way that data integrity is not affected.

UNIQUE	**NOT NULL**
PRIMARY KEY	**FOREIGN KEY**
DEFAULT	**CHECK**

IA

MS SQL Server

TUTORIAL

6 **SQL STATEMENTS : INTRO**

SQL STATEMENTS

- ✓ **DDL STATEMENTS**

used to create, modify the object's structure (database, tables, views, triggers)

CREATE	**ALTER**
DROP	**TRUNCATE**
RENAME	

IA

SQL STATEMENTS

- ✓ **DML STATEMENTS**

used to insert data, modify the existing data, removing and retrieving data from the tables

INSERT	**UPDATE**
DELETE	**SELECT**

IA

SQL STATEMENTS

- ✓ **TCL STATEMENTS**

used to save data, undo/redo transactions performed on the database.

COMMIT	ROLLBACK	SAVE TRANSACTION

SQL STATEMENTS

- ✓ **DCL STATEMENTS**

  used to create roles, permissions, referential integrity and as well it is used to control access to database by securing it.

GRANT	REVOKE

MS SQL Server

TUTORIAL

7

CREATE STATEMENT

DDL STATEMENTS

- ✓ **CREATE**

used to create object's structure (database, tables, views, triggers)

```
CREATE DATABASE database_name;

CREATE DATABASE employeeDB;
```

IA

DDL STATEMENTS

```
CREATE TABLE table_name
(column_1      datatype(size)     constraints,
 column_2      datatype(size)     constraints
 .       .     .       .      .       .       .
column_n       datatype(size)     constraints);
```

```
CREATE TABLE employee_info (
 empId          INTEGER            PRIMARY KEY,
 empName        VARCHAR(20)        NOT NULL,
 empSalary      DECIMAL(10,2)      NOT NULL,
 job            VARCHAR(20),
 phone          INTEGER            UNIQUE,
 deptID         INTEGER            NOT NULL
);
```

IA

MS SQL Server

TUTORIAL

8 **INSERT STATEMENT**

DDL STATEMENTS

- ✓ **INSERT**

used to insert data or information into the table.

```
INSERT INTO table_name VALUES(col1_value, col2_value, …….);

INSERT INTO employee_info
VALUES (01, ‘Adam’, 20000, ‘Developer’, 9879879879, 10);
```

```
INSERT INTO table_name (col1, col2)
VALUES (col1_value, col2_value);

INSERT INTO employee_info (empId, empName, empSalary,
deptId) VALUES (02, ‘Smith’, 35000, 10);
```

IA

MS SQL Server

TUTORIAL

9 **SELECT STATEMENT**

DML STATEMENTS

- ✓ **SELECT**

used to retrieve data or information from the table.

```
SELECT */ [column_name] [function_name] FROM table_name
[WHERE condition]
[GROUP BY condition]
[HAVING condition]
[ORDER BY column_name];
```

```
SELECT * FROM employee_info;

SELECT empName, empSalary FROM employee_info;
```

IA

MS SQL Server

TUTORIAL

10 **UPDATE STATEMENT**

DML STATEMENTS

- ✓ **UPDATE**

used to update/modify existing table data/information .

```
UPDATE table_name
SET column_name = column_value, . . .
[WHERE condition];
```

```
UPDATE employee_info SET empSalary = empSalary + 1000;

UPDATE employee_info SET job = 'Tester'
WHERE empName = 'Smith';
```

IA

MS SQL Server

TUTORIAL

11

DELETE STATEMENT

DML STATEMENTS

- ✓ **DELETE**

used to delete one or more records from table.

```
DELETE FROM table_name [WHERE condition];
```

```
DELETE FROM employee_info;

DELETE FROM employee_info WHERE deptId = 10;
```

IA

MS SQL Server

TUTORIAL

12 **ORDER BY CLAUSE**

ishwar.academy

Website

IA

ORDER BY CLAUSE

- ✓ used to sort / arrange records in either ascending or descending order.
- ✓ always used with SELECT statement.

```
SELECT column_name(s) FROM table_name
ORDER BY column_name [DESC];
```

```
SELECT * FROM employee_info ORDER BY empSalary;

SELECT * FROM employee_info
ORDER BY empSalary DESC;
```

IA

MS SQL Server

TUTORIAL

13

WHERE CLAUSE

WHERE CLAUSE

- ✓ Using WHERE clause, you are able to restrict the query to rows that meet a condition.
- ✓ You can use any operators in WHERE clause.

```
SELECT * FROM table_name WHERE condition;

UPDATE table_name SET column_name = value WHERE condition;

DELETE FROM table_name WHERE condition;
```

IA

14 AGGREGATE FUNCTION

15 NUMERIC FUNCTION

16 STRING FUNCTION

17 Boolean Operators

18 DATE & TIME FUNCTION

IA

MS SQL Server

TUTORIAL

19 **GROUP BY CLAUSE**

GROUP BY CLAUSE

- ✓ defines one or more columns as a group such that all rows within any group have the same values for those columns.
- ✓ always used with SELECT statement.

```
SELECT column_name(s), aggregate_function( )
FROM table_name GROUP BY column_name;
```

```
SELECT deptId FROM employee_info GROUP BY deptID;

SELECT deptId, sum(empSalary) FROM employee_info
GROUP BY deptId;
```

IA

MS SQL Server

TUTORIAL

20

HAVING CLAUSE

HAVING CLAUSE

- ✓ The HAVING clause defines the condition that is then applied to groups of rows.
- ✓ always used with SELECT statement inside GROUP BY clause.

```
SELECT column_name(s), aggregate_function( )
FROM table_name
GROUP BY column_name HAVING condition;
```

```
SELECT deptId, sum(empSalary)
FROM employee_info
GROUP BY deptId HAVING deptId = 20;
```

IA

MS SQL Server

TUTORIAL

21

TOP( ) CLAUSE

TOP( ) CLAUSE

- ✓ The TOP clause specifies the ***first n rows*** of the query result that are to be retrieved.
- ✓ This clause should always be used with the ORDER BY clause

```
SELECT TOP(n) column_name FROM table_name
ORDER BY column_name [DESC];
```

```
SELECT TOP(3) empSalary FROM employeeInfo
ORDER BY empSalary DESC;
```

IA

MS SQL Server

TUTORIAL

22

Create a copy of the table from different database

✓ To create a copy of the table from different database

```
SELECT column_name / * INTO table_name
FROM database_name.table_name;
```

```
SELECT * INTO employee_details
FROM employee_db.employee_info
```

IA

MS SQL Server

TUTORIAL

23

ALTER STATEMENT

Adding Columns & Constraints

DDL STATEMENTS

- ✓ **ALTER TABLE**
  - ❖ Modifies a table definition by **adding**, **altering**, or **dropping columns and constraints**.
  - ❖ It also reassigns and rebuilds partitions, or disables and enables constraints and triggers.

IA

ALTER TABLE Statement

- ✓ **Adding a new column**
  - adds a column without constraint that allows null values.

```
ALTER TABLE table_name ADD column_name datatype(size) [null];
```

```
ALTER TABLE emp_info ADD salary decimal;

ALTER TABLE emp_info ADD phone varchar(10) null;
```

IA

ALTER TABLE Statement

- ✓ **Adding a column with a constraint**
  - adds a new column with constraint (UNIQUE, DEFAULT etc.).

```
ALTER TABLE table_name ADD column_name datatype(size) constraint [null];

ALTER TABLE table_name ADD column_name datatype(size) null
        CONSTRAINT constraint_reference_name constraint;
```

```
ALTER TABLE emp_info ADD salary decimal not null;

ALTER TABLE emp_info ADD projectID integer null
        CONSTRAINT pID_unique_key UNIQUE;
```

IA

ALTER TABLE Statement

- ✓ **Adding several columns with constraints**
  - adds more than one column with constraints defined with the new column.

```
ALTER TABLE table_name
        ADD column_name datatype(size) constraint,
                                . . .       . . .       . . .                ;
```

```
ALTER TABLE emp_info
        ADD salary decimal DEFAULT 15000,
              projectID integer null CONSTRAINT pID_unique_key UNIQUE;
```

IA

MS SQL Server

TUTORIAL

24

ALTER STATEMENT

Dropping Columns & Constraints

ALTER TABLE Statement

- ✓ **Dropping a column or columns**
  - remove a column or multiple columns.

```
ALTER TABLE table_name DROP COLUMN column_name(s);
```

```
ALTER TABLE emp_info DROP COLUMN salary;

ALTER TABLE emp_info DROP COLUMN salary, age;
```

IA

ALTER TABLE Statement

- ✓ **Dropping constraints and columns**
  - removes a constraint
  - removes constraints and columns.

```
ALTER TABLE table_name DROP CONSTRAINT constraint_name;

ALTER TABLE table_name DROP CONSTRAINT constraint_name,
                                COLUMN column_name;
```

```
ALTER TABLE emp_info DROP CONSTRAINT pID_unique_key;

ALTER TABLE emp_info DROP CONSTRAINT pID_unique_key,
                                COLUMN salary;
```

IA

MS SQL Server

TUTORIAL

25

ALTER STATEMENT

Altering Column Definitions

ALTER TABLE Statement

- ✓ **Changing the data type of a column**
  - changes a column of a table from one data type to another.

```
ALTER TABLE table_name ALTER COLUMN column_name datatype(size);
```

```
ALTER TABLE emp_info ALTER COLUMN salary decimal(8,2);

ALTER TABLE emp_info ALTER COLUMN emp_name varchar(50);
```

IA

ALTER TABLE Statement

- ✓ **Changing the size of a column**
  - change (increase or decrease) the size of a column.

```
ALTER TABLE table_name ALTER COLUMN column_name datatype(size);
```

```
ALTER TABLE emp_info ALTER COLUMN salary decimal(8,2);

ALTER TABLE emp_info ALTER COLUMN emp_name varchar(50);
```

Note: If the columns contain data, the column size can only be increased.

IA

ALTER TABLE Statement

- ✓ **Changing the data type and size of a column**
  - change the data type as well as the size of a column at same time.

```
ALTER TABLE table_name ALTER COLUMN column_name datatype(size);
```

```
ALTER TABLE emp_info ALTER COLUMN salary decimal(8,2);

ALTER TABLE emp_info ALTER COLUMN emp_name varchar(50);
```

Note: If the columns contain data, the column size can only be increased.

IA

26

Aliases

Aliasing

- can be used to create a temporary name for columns or tables.

Types

1. **Column Aliases** are used to make column headings in query output easier to read.
   (Specially with Functions and Column Concatenation)

2. **Table Aliases** are used to shorten your SQL to make it easier to read.
   (Specially in Join, and Subquery)

Column Alias

Syntax

```
SELECT column_name [function( )] AS alias_name FROM table_name;

SELECT column_name [function( )] alias_name FROM table_name;
```

Example

```
SELECT sal AS Salary FROM employee;

SELECT fname || lname 'Full Name' FROM employee;
```

27

Joins

ishwar.academy

Website

Joins

- used to retrieve data from multiple tables.

Types

1. **Inner Join**
2. **Outer Join**
    i. Left Outer Join
    ii. Right Outer Join
    iii. Full Outer Join
3. **Cross Join**

EMPLOYEE

EMP_ID	EMP_NAME	EMP_SALARY	EMP_DEPTID
1111	STEVE	35000	D1
1112	ADAM	28000	D2
1113	JOHN	50000	D3
1114	MIKE	45000	D2
1115	PETER	60000	D1

DEPARTMENT

DEPT_ID	DEPT_NAME	DEPT_LOCATION
D1	DEVELOPMENT	CALIFORNIA
D2	MARKETING	MUMBAI
D3	ACCOUNTS	NEW YORK
D4	MANAGEMENT	SYDNEY

EMPLOYEE_RECORD

EMP_ID	EMP_NAME	EMP_SALARY	DEPT_NAME	DEPT_LOCATION
1111	STEVE	35000	DEVELOPMENT	CALIFORNIA
1112	ADAM	28000	MARKETING	MUMBAI
1113	JOHN	50000	ACCOUNTS	NEW YORK
1114	MIKE	45000	MARKETING	MUMBAI
1115	PETER	60000	DEVELOPMENT	CALIFORNIA

28

Inner Join

ishwar.academy

Website

Inner Join

- The inner join is one of the most commonly used joins in SQL Server.
- It return all rows from multiple tables where the join condition is satisfied.

Syntax

```
SELECT column_name(s) FROM table1_name INNER JOIN table2_name
ON table1_name.column_name = table2_name.column_name;
```

EMPLOYEE

EMP_ID	EMP_NAME	EMP_SALARY	EMP_DEPTID
1111	STEVE	35000	D1
1112	ADAM	28000	D2
1113	JOHN	50000	D3
1114	MIKE	45000	D2
1115	PETER	60000	D5

DEPARTMENT

DEPT_ID	DEPT_NAME	DEPT_LOCATION
D1	DEVELOPMENT	CALIFORNIA
D2	MARKETING	MUMBAI
D3	ACCOUNTS	NEW YORK
D4	MANAGEMENT	SYDNEY

EMPLOYEE_RECORD

EMP_ID	EMP_NAME	EMP_SALARY	DEPT_NAME	DEPT_LOCATION
1111	STEVE	35000	DEVELOPMENT	CALIFORNIA
1112	ADAM	28000	MARKETING	MUMBAI
1113	JOHN	50000	ACCOUNTS	NEW YORK
1114	MIKE	45000	MARKETING	MUMBAI

Example

```
SELECT emp_id, emp_name, emp_salary, dept_name, dept_location
FROM employee INNER JOIN department
ON employee.emp_deptid = department.dept_id;
```

```
SELECT e.emp_id, e.emp_name, e.emp_salary, d.dept_name,
d.dept_location FROM employee e INNER JOIN department d
ON e.emp_deptid = d.dept_id;
```

29

Left Outer Join

Left Outer Join

- return all rows from the left-hand table and records in the right-hand table with matching values.

Syntax

```
SELECT column_name(s)
FROM table1_name LEFT OUTER JOIN table2_name
ON table1_name.column_name = table2_name.column_name;
```

Note: The records without matching values are replaced with NULLs in the respective columns.

EMPLOYEE

EMP_ID	EMP_NAME	EMP_SALARY	EMP_DEPTID
1111	STEVE	35000	D1
1112	ADAM	28000	D2
1113	JOHN	50000	D3
1114	MIKE	45000	D2
1115	PETER	60000	D5

DEPARTMENT

DEPT_ID	DEPT_NAME	DEPT_LOCATION
D1	DEVELOPMENT	CALIFORNIA
D2	MARKETING	MUMBAI
D3	ACCOUNTS	NEW YORK
D4	MANAGEMENT	SYDNEY

EMPLOYEE_RECORD

EMP_ID	EMP_NAME	EMP_SALARY	DEPT_NAME	DEPT_LOCATION
1111	STEVE	35000	DEVELOPMENT	CALIFORNIA
1112	ADAM	28000	MARKETING	MUMBAI
1113	JOHN	50000	ACCOUNTS	NEW YORK
1114	MIKE	45000	MARKETING	MUMBAI
1115	PETER	60000	NULL	NULL

Example

```
SELECT emp_id, emp_name, emp_salary, dept_name, dept_location
FROM employee LEFT OUTER JOIN department
ON employee.emp_deptid = department.dept_id;
```

```
SELECT e.emp_id, e.emp_name, e.emp_salary, d.dept_name,
d.dept_location FROM employee e LEFT OUTER JOIN department d
ON e.emp_deptid = d.dept_id;
```

30

Right Outer Join

ishwar.academy

Website

Right Outer Join

- return all rows from the right-hand table and records in the left-hand table with matching values.

Syntax

```
SELECT column_name(s)
FROM table1_name RIGHT OUTER JOIN table2_name
ON table1_name.column_name = table2_name.column_name;
```

Note: The records without matching values are replaced with NULLs in the respective columns.

EMPLOYEE

EMP_ID	EMP_NAME	EMP_SALARY	EMP_DEPTID
1111	STEVE	35000	D1
1112	ADAM	28000	D2
1113	JOHN	50000	D3
1114	MIKE	45000	D2
1115	PETER	60000	D5

DEPARTMENT

DEPT_ID	DEPT_NAME	DEPT_LOCATION
D1	DEVELOPMENT	CALIFORNIA
D2	MARKETING	MUMBAI
D3	ACCOUNTS	NEW YORK
D4	MANAGEMENT	SYDNEY

EMPLOYEE_RECORD

EMP_ID	EMP_NAME	EMP_SALARY	DEPT_NAME	DEPT_LOCATION
1111	STEVE	35000	DEVELOPMENT	CALIFORNIA
1112	ADAM	28000	MARKETING	MUMBAI
1114	MIKE	45000	MARKETING	MUMBAI
1113	JOHN	50000	ACCOUNTS	NEW YORK
NULL	NULL	NULL	MANAGEMENT	SYDNEY

Example

```
SELECT emp_id, emp_name, emp_salary, dept_name, dept_location
FROM employee RIGHT OUTER JOIN department
ON employee.emp_deptid = department.dept_id;
```

```
SELECT e.emp_id, e.emp_name, e.emp_salary, d.dept_name,
d.dept_location FROM employee e RIGHT OUTER JOIN department d
ON e.emp_deptid = d.dept_id;
```

31

Full Outer Join

ishwar.academy

Website

Full Outer Join

- return all rows from both left-hand and right-hand table with matching values.

Syntax

```
SELECT column_name(s)
FROM table1_name FULL OUTER JOIN table2_name
ON table1_name.column_name = table2_name.column_name;
```

Note: The records without matching values are replaced with NULLs in the respective columns.

EMPLOYEE

EMP_ID	EMP_NAME	EMP_SALARY	EMP_DEPTID
1111	STEVE	35000	D1
1112	ADAM	28000	D2
1113	JOHN	50000	D3
1114	MIKE	45000	D2
1115	PETER	60000	D5

DEPARTMENT

DEPT_ID	DEPT_NAME	DEPT_LOCATION
D1	DEVELOPMENT	CALIFORNIA
D2	MARKETING	MUMBAI
D3	ACCOUNTS	NEW YORK
D4	MANAGEMENT	SYDNEY

EMPLOYEE_RECORD

EMP_ID	EMP_NAME	EMP_SALARY	DEPT_NAME	DEPT_LOCATION
1111	STEVE	35000	DEVELOPMENT	CALIFORNIA
1112	ADAM	28000	MARKETING	MUMBAI
1113	JOHN	50000	ACCOUNTS	NEW YORK
1114	MIKE	45000	MARKETING	MUMBAI
1115	PETER	60000	NULL	NULL
NULL	NULL	NULL	MANAGEMENT	SYDNEY

Example

```
SELECT emp_id, emp_name, emp_salary, dept_name, dept_location
FROM employee FULL OUTER JOIN department
ON employee.emp_deptid = department.dept_id;
```

```
SELECT e.emp_id, e.emp_name, e.emp_salary, d.dept_name,
d.dept_location FROM employee e FULL OUTER JOIN department d
ON e.emp_deptid = d.dept_id;
```

32

Subquery

ishwar.academy

Website

EMPLOYEE

EMP_ID	EMP_NAME	EMP_SALARY	EMP_DEPTID
1111	STEVE	35000	D1
1112	ADAM	28000	D2
1113	JOHN	50000	D3
1114	MIKE	45000	D2
1115	PETER	60000	D5

DEPARTMENT

DEPT_ID	DEPT_NAME	DEPT_LOCATION
D1	DEVELOPMENT	CALIFORNIA
D2	MARKETING	MUMBAI
D3	ACCOUNTS	NEW YORK
D4	MANAGEMENT	SYDNEY

- ? **Display name, salary of the employees whose salary is greater than Mike's salary.**
- ? **Display name, salary of the employees whose salary is greater than Adam's salary and deptno same as Adam's deptno.**
- ? **Display the employee information whose department is located at New York.**

Subquery

- A query within another SQL query and embedded within the WHERE clause.
- Subquery must be enclosed within parenthesis ( ).
- Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the comparison operators.
- A subquery can have only one column in the SELECT statement.

Syntax

```
SELECT column_name(s) FROM table_name

WHERE column_name OPERATOR

        (SELECT column_name FROM table_name [WHERE condition]);
```

Example

```
SELECT emp_name, emp_salary FROM employee
WHERE emp_salary =
        (SELECT emp_salary FROM employee WHERE emp_name = 'MIKE');
```

```
SELECT emp_name, emp_salary FROM employee
WHERE emp_salary IN
        (SELECT emp_salary FROM employee WHERE emp_name = 'MIKE');
```

33

Transact-SQL (T-SQL)

Fundamentals

ishwar.academy

Website

Transact-SQL

- Transact-SQL is better known as **T-SQL**.
- The purpose of T-SQL is used **to provide a set of tools for the development of a transactional database**.

Why T-SQL?

Standard SQL

- DML Statements
- Operators
- Built-in Functions
- Single-line Query

T-SQL

- Standard SQL
- Working with Variables
- Batch or Script
- User-defined Functions
- Triggers
- Stored Procedures

and many more…

34

Working with Variables

T-SQL

Working with Variables

- In every programming language, variables are generally used to temporarily store values in memory.
- T-SQL variables are created with **DECLARE** command followed by **variable name** preceded with **@** symbol and **data type**.
- By default, the value of declared variable is **NULL**.

DECLARE @variable_name datatype(size)**;**

DECLARE @name VARCHAR(50)**;**

DECLARE @name VARCHAR(50), @age INT**;**

Assign a Value to a Variable

- Both the **SET** and the **SELECT** command can assign the value to a variable.
- **SET** can only set the value of **one variable** at a time;
- **SELECT** command retrieve data from tables and assign **multiple variables** values with a single statement.

```
SET @variable_name = value;

SELECT  @variable_name = value, @variable_name = value;
```

```
SET @salary = 30000;

SELECT @name = 'ishwar', @age = 25;
```

Incrementing Variable

- With the increment variable feature, we can perform **mathematical operations (like addition, subtraction, and multiplication)** on the variable.

```
SET @number += 10;
```

```
SET @number = @number + 10;
```

```
SET @number *= 10;
```

```
SET @number = @number * 10;
```

```
SET @number -= 10;
```

```
SET @number = @number - 10;
```

Example

```
DECLARE @number INT = 100;

SET @number += 50;

SELECT @number;


SET @number -= 50;

SELECT @number;


SET @number *= 50;

SELECT @number;
```

35

Global Variables

T-SQL

ishwar.academy

Website

Global Variables

- Global variables represent a special type of variable.
- The server always maintain the values of these variables.
- All the global variables represent information specific to the server or a current user session.
- **Global variable names begin with a @@ prefix.**
- They are system-defined functions and you cannot declare them.

@@CONNECTION	@@ERROR
@@MAX_CONNECTION	@@ROWCOUNT
@@SERVERNAME	@@IDENTITY
@@SPID	@@TIMETICKS
@@CPU_BUSY	@@VERSION
@@IDLE	@@LANGUAGE

etc….

35

Script

T-SQL

ishwar.academy

Website

36

Batch

T-SQL

ishwar.academy

Website

Batch

- A batch of SQL statements is **a group of two or more SQL statements** or **a single SQL statement**.
- A batch of SQL statement can have :
  - ✓ Data Definition Language (DDL) Statements
  - ✓ Data Manipulation Language (DML) Statements
  - ✓ Data Control Language (DCL) Statements

Standard Types of Batches

1. **Explicit Batch**

   An ***explicit batch*** is two or more SQL statements separated by semicolons (;).

   For example,

   ```
   insert into employee (emp_name, emp_salary) values('Brad',45000);
   insert into employee (emp_name, emp_salary) values('Joe',36000);
   ```

2. **Procedure**

   If a ***procedure*** contains more than one SQL statement, it is considered to be a batch.

37

Go Command

T-SQL

GO Command

- GO is not a T-SQL statement; it is a command recognized by SQL Server utilities.
- GO can be executed by any user. It requires no permissions.
- It signals the end of a batch to the SQL Server utilities.

Syntax GO [count]

where, count is a positive integer. The batch will execute the specified number of times.

Example

The following example creates two batches.

```
USE COMPANY_DB;
GO


DECLARE @Name VARCHAR(50);
SELECT @Name = ‘Microsoft’;
GO
```

- The first batch contains only a **USE COMPANY_DB** statement to set the database.

- The remaining statements use a local variable. Therefore, all local variable declarations must be grouped in a single batch. This is done by not having a GO command until after the last statement that references the variable.

38

Control of Flow

T-SQL

Control of Flow

- **Transact-SQL (T-SQL) statements are executed in sequential order** (suppose, we have created three statements then, **first statement will run, followed by second, followed by third**.).
- However, in many cases, we will want to interrupt this normal flow.
- **T-SQL has keywords to control the order of execution.**
- In T-SQL, these keywords are known as **a control-of-flow**.

Control-of-flow Keywords

BEGIN…END	IF…ELSE	WHILE
BREAK	CONTINUE	GOTO
RETURN	TRY…CATCH	THROW
WAITFOR		

39

BEGIN...END

T-SQL

ishwar.academy

Website

BEGIN…END

- The **BEGIN…END** keywords are used to group multiple lines into one Statement Block.
- In addition, **BEGIN…END can be nested**. It simply means that we can place a **BEGIN…END** statement within another **BEGIN… END** statement.

Syntax

```
BEGIN
    { SQL Statements or Statement Block }
END
```

Example

```
BEGIN
        DECLARE @name VARCHAR(50), @salary INTEGER,
                @DeptID VARCHAR(10) = 'D3';
        SELECT @name = emp_name, @salary = emp_salary FROM employee
        WHERE emp_deptid = @DeptID;
        SELECT @name 'Name', @salary 'Salary';
END
```

Example of Nesting BEGIN...END

```
BEGIN
        DECLARE @name VARCHAR(50), @salary INTEGER,
                        @DeptID VARCHAR(10) = 'D3';
        SELECT @name = emp_name, @salary = emp_salary FROM employee
        WHERE emp_deptid = @DeptID;
        SELECT @name 'Name', @salary 'Salary';
        BEGIN
                PRINT 'Department ID : ' + @DeptID;
        END
END
```

40

IF...ELSE

T-SQL

IF...ELSE

- The **IF...ELSE** statement is a control-flow statement that allows you to execute or skip a statement block based on a specified condition.

Syntax : IF statement

```
IF condition
BEGIN
    { SQL Statements or Statement Block }
END
```

NOTE: If the condition contains a SELECT statement, then it must be enclosed in parentheses.

Example of IF

```
BEGIN
    DECLARE @salary DECIMAL;
    SELECT @salary = AVG(emp_salary) FROM employee;
    SELECT @salary AS 'Avg. Salary';
    IF @salary > 25000
    BEGIN
        PRINT 'Average salary is greater than 25000';
    END
END
```

Syntax : IF…ELSE statement

```
IF condition
        BEGIN
                { SQL Statements or Statement Block }
        END
ELSE
        BEGIN
                { SQL Statements or Statement Block }
        END
```

Example of IF…ELSE

```
BEGIN
    DECLARE @salary DECIMAL;
    SELECT @salary = AVG(emp_salary) FROM employee;
    SELECT @salary AS 'Avg. Salary';
    IF @salary > 25000
        BEGIN
            PRINT 'Avg. salary is greater than 25000';
        END
    ELSE
        BEGIN
            PRINT 'Avg. salary is less than 25000';
        END
END
```

41

WHILE

T-SQL

ishwar.academy

Website

WHILE

- The **WHILE** loop statement is a control-flow statement that allows you to execute a statement block repeatedly as long as a specified condition is **TRUE**.
- The execution of statements in the **WHILE** loop can be controlled from inside the loop with the **BREAK** and **CONTINUE** keywords.

Syntax

```
WHILE condition
BEGIN
    { SQL Statements or Statement Block }
END
```

NOTE: If the condition contains a SELECT statement, then it must be enclosed in parentheses.

Example

```
BEGIN
    WHILE ( SELECT MIN(emp_salary) FROM employee ) < 80000
    BEGIN
            UPDATE employee SET emp_salary = emp_salary + 10000;
            PRINT 'Salary updated';
            IF ( SELECT MIN(emp_salary) FROM employee) >= 80000
                    PRINT ‘Min. Salary is greater or equal to 80000.';
                    BREAK;
    END
END
```

42

TRY...CATCH

T-SQL

TRY…CATCH

- **TRY…CATCH** implements **error handling** for T-SQL.
- It is similar to the exception handling in the object-oriented programming languages such as C++, Java, JavaScript, etc.
- A group of T-SQL statements can be enclosed in a **TRY** block.
- If the statements between the **TRY** block **complete without an error**, the statements between the **CATCH block will not execute**. However, if any statement inside the **TRY** block **causes an exception**, the **control transfers to the statements in the CATCH** block.

Syntax

```
BEGIN TRY
    { SQL Statement or Statement Block }
END TRY
BEGIN CATCH
    [ { SQL Statement or Statement Block } ]
END CATCH
```

NOTE: Any group of Transact-SQL statements in a batch or enclosed in a BEGIN...END block.

To retrieve the information about the error :

- **ERROR_MESSAGE()** returns the complete text of the generated error message.
- **ERROR_NUMBER()** returns the number of the error.
- **ERROR_LINE()** returns the line number inside the routine that caused the error.
- **ERROR_PROCEDURE()** returns the name of the stored procedure or trigger where the error occurred.
- **ERROR_STATE()** returns the error state number.
- **ERROR_SEVERITY()** returns the severity**.**

NOTE: These functions return NULL if they are called outside the scope of the CATCH block.

Example

```
BEGIN TRY
    SELECT 100/5 AS 'Division';
END TRY
BEGIN CATCH
   SELECT ERROR_MESSAGE() AS 'Error Message';
END CATCH;
```

Example

```
BEGIN TRY
    SELECT 10/0 AS 'Division';
END TRY
BEGIN CATCH
   SELECT ERROR_MESSAGE() AS 'Error Message';
END CATCH;
```

43

WAITFOR

T-SQL

WAITFOR

- **WAITFOR** blocks the execution of a batch, stored procedure, or transaction until either a specified time or time interval elapses, or a specified statement modifies or returns at least one row.
- **WAITFOR** has two arguments :

TIME	the period of time to wait. time_to_pass
DELAY	the time (up to a maximum of 24 hours) at which the WAITFOR statement finishes.

Syntax

```
BEGIN
    WAITFOR TIME 'time_to_execute'
    { SQL Statement or Statement Block }
END
```

where, **time_to_execute** can be specified either in a **datetime** data format, or as a **local variable**.

Syntax

```
BEGIN

    WAITFOR DELAY 'time_to_pass'

    { SQL Statement or Statement Block }

END
```

where, **time_to_pass** can be specified either in a **datetime** data format, or as a **local variable**.

Example

```
BEGIN
    WAITFOR TIME '18:00:00'
    SELECT * FROM employee;
END
```

```
BEGIN
    WAITFOR DELAY '00:00:10'
    SELECT * FROM employee;
END
```

T-SQL

44

STORED PROCEDURE

ishwar.academy

Website

STORED PROCEDURE

- A **stored procedure** is a group of one or more T-SQL statements.
- It can be stored in the database.
- **Stored procedures**
  - ✓ accept input parameters and return multiple values
  - ✓ contain programming statements that perform operations in the database.
  - ✓ return a status value to a calling program to indicate success or failure

BENEFITS

- ✓ Reuse of code
- ✓ Easy to maintain
- ✓ Improve performance
- ✓ Strong security
- ✓ Reduce server/client network traffic

TYPES

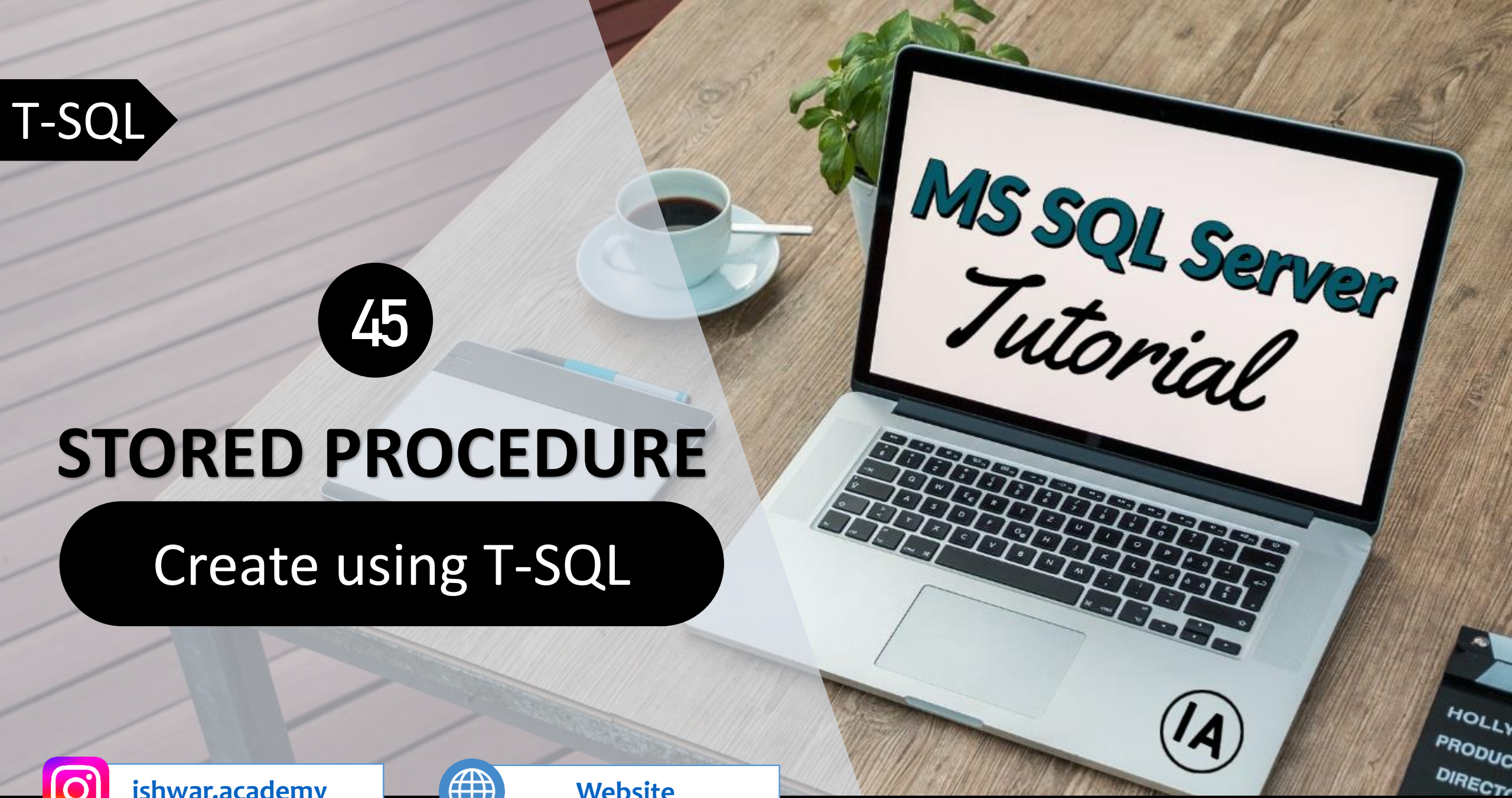
- ✓ **System**
  - physically stored in the internal **Resource** database
- ✓ **User-defined**
  - It can created in a user-defined database or in all system databases except the **Resource** database.
- ✓ **Temporary**
  - a form of user-defined procedures. The temporary procedures are like a permanent procedure, except temporary procedures are stored in **tempdb.**

T-SQL

45

STORED PROCEDURE

Create using T-SQL



ishwar.academy

Website

STORED PROCEDURE

- Two ways to create (or define) a stored procedure
    1. **Stored Procedure without Parameter (Simple Stored Procedure)**
    2. **Stored Procedure with Parameter**

Syntax (Simple Procedure)

```
CREATE PROCEDURE procedure_name

AS

BEGIN

    { SQL Statement or Statement Block }

END
```

Example: Create a stored procedure that returns all employees.

```
SELECT * FROM employee;
```

```
CREATE PROCEDURE proc_allEmployeesDetails
AS
BEGIN
    SELECT * FROM employee;
END
```

T-SQL

46

STORED PROCEDURE

Parameterized

Syntax (Stored Procedure with parameter)

```
CREATE PROCEDURE procedure_name(parameter list)

AS

BEGIN

    { SQL Statement or Statement Block }

END
```

Example: Create a stored procedure that returns all employees whose department location is Mumbai.

```
SELECT * FROM employee e
inner join department d
ON e.emp_deptid = d.deptid
WHERE dept_location = 'mumbai';
```

```
CREATE PROCEDURE
proc_allEmployeesDetails(@location AS VARCHAR(100))
AS
BEGIN
    SELECT * FROM employee e inner join department d
    ON e.emp_deptid = d.deptid
    WHERE dept_location = @location;
END
```

T-SQL

47

STORED PROCEDURE

Modify / Alter

ishwar.academy

Website

Modify (Alter) a STORED PROCEDURE

- We can modify a previously created stored procedure.
- To perform modification, we need to use **ALTER** command**.**

Syntax (Alter Procedure)

```
ALTER PROCEDURE procedure_name

AS

BEGIN

    { SQL Statement or Statement Block }

END
```

Example: Modify an existing simple stored procedure proc_allEmployeesDetails.

```
CREATE PROCEDURE
proc_allEmployeesDetails
AS
BEGIN
    SELECT * FROM employee;
END
```

```
ALTER PROCEDURE proc_allEmployeesDetails
AS
BEGIN
    SELECT e.emp_name, e.emp_salary, d.dept_location
    FROM employee e
    inner join department d
    ON e.emp_deptid = d.dept_id;
END
```

Example: Modify an existing parameterized stored procedure proc_employeeDetailsLocationWise.

```
CREATE PROCEDURE
proc_employeeDetailsLocationWise(@location AS VARCHAR(100))
AS
BEGIN
        SELECT * FROM employee e
        inner join department d
        ON e.emp_deptid = d.dept_id
        WHERE d.dept_location = @location;
END;
```

```
ALTER PROCEDURE
proc_employeeDetailsLocationWise(@location AS VARCHAR(100))
AS
BEGIN
        SELECT e.emp_name, e.emp_salary, d.dept_location
        FROM employee e
        inner join department d
        ON e.emp_deptid = d.dept_id
        WHERE d.dept_location = @location;
END;
```

T-SQL

48

STORED PROCEDURE

Rename

Rename a Stored Procedure

- To rename the existing stored procedure, we need to use **system** procedure

sp_rename

Syntax (Rename Procedure)

```
EXEC sp_rename 'COMPANY_DB.proc_allEmployeeDetails', 'proc_displayEmployeeDetails';
```

Drawbacks (or Limitations)

- Renaming a stored procedure does not change the name of the corresponding object name in the definition column of the **sys.sql_modules** catalog view. To do that, you must drop and re-create the stored procedure with its new name.
- Changing the name or definition of a procedure can cause dependent objects to fail when the objects are not updated to reflect the changes that have been made to the procedure.

T-SQL

49

User-Defined Functions

Introduction

User-Defined Functions (UDFs)

- **User-defined functions (UDFs)** are routines that accept parameters, perform an action (complex calculation), and return the result of that action as a value. The return value can either be a single scalar value or a result set.

Why UDFs

- ✓ Modular Programming
- ✓ Faster execution
- ✓ Reduce network traffic

Why UDFs

- ✓ **Modular Programming**
  - create the function once,
  - store it in the database, and
  - call it any number of times in your program.

Why UDFs

- ✓ **Faster execution**
  - reduce the compilation cost of Transact-SQL code

    (UDFs does not need to be reparsed and reoptimized with each use resulting in much faster execution times).

Why UDFs

- ✓ **Reduce network traffic**
  - function can be invoked in the WHERE clause to reduce the number of rows sent to the client.

TYPES

- ✓ **System**
  - SQL Server provides many system functions that you can use to perform a variety of operations. They cannot be modified
- ✓ **Scalar**
  - Scalar functions return a single data value of the type defined in the **RETURNS** clause.
- ✓ **Table-Valued**
  - table-valued functions return a **table** data type.

T-SQL

50

User-Defined Functions (Scalar)

Create

ishwar.academy

Website

Before creating a function, things to know…

- User-defined function always returns a value.
- User-defined functions have only input parameters for it.
- User-defined functions can not return multiple result sets.
- Error handling is restricted in a user-defined function. A UDF does not support **TRY…CATCH**, **@ERROR** or **RAISERROR**.
- SET statements are not allowed in a user-defined function.
- User-defined functions cannot call a stored procedure.
- User-defined functions can be nested. User-defined functions can be nested up to 32 levels.

Syntax (Scalar Function)

```
CREATE FUNCTION function_name(parameter datatype, . . . )

RETURNS return_datatype

[ WITH <function_option> [ ,...n ] ]

[ AS ]

    BEGIN

            function_body

            RETURN scalar_expression

    END;
```

Example: Create a function to get employee salary by passing employee name.

```
CREATE FUNCTION salary(@name as varchar(50))
RETURNS decimal
BEGIN
    DECLARE @sal decimal;
    SELECT @sal = emp_salary FROM employee
    WHERE emp_name = @name;

    RETURN @sal;
END
```

T-SQL

51

User-Defined Functions (Table-valued)

Create

Table-valued Functions

- User-defined table-valued functions return a **table** data type.
- A table-valued function accepts zero or more parameters.

Types

1. **Inline Table-valued Function**
   - There is no function body (i.e. there is no need for a BEGIN-END block in an Inline function)
   - The table is the result set of a single SELECT statement.

Syntax (Inline Table-valued Function)

```
CREATE FUNCTION function_name(parameter datatype, . . . )

RETURNS return_datatype

AS

RETURN statement
```

Example: Create a function to get employee information by passing employee salary.

```
CREATE FUNCTION getAllEmployees(@salary decimal)
RETURNS TABLE
AS
RETURN
    SELECT * FROM employee WHERE emp_salary = @salary;
```

To execute the function,

```
SELECT * FROM getAllEmployees(20000);
```

Types

2. Multi-statement Table-valued Function

- It contains multiple SQL statements enclosed in **BEGIN-END** blocks.
- The return value is declared as a **table variable**. The **RETURN** statement is without a value and the declared table variable is returned.

Syntax (Multi-Statement Table-valued Function)

```
CREATE FUNCTION function_name(parameter datatype, . . . )

RETURNS @table_variable TABLE

(column_1 datatype, column_2 datatype, . . .)

AS

    BEGIN

        SQL-statement(s)

        RETURN

    END;
```

Example: Create a function to get list of employees by passing department id.

```
CREATE FUNCTION getAllEmployees(@id varchar(50))
RETURNS @Table TABLE
(ID int, NAME varchar(50), SALARY decimal, DEPTID varchar(50))
AS
BEGIN
    INSERT INTO @Table
    SELECT * FROM employee WHERE emp_deptid = @id;
    RETURN
END;
```

T-SQL

52

User-Defined Functions

Modify, Rename, Delete, View

ishwar.academy

Website

T-SQL

53

Trigger - Introduction

ishwar.academy

Website

Trigger

- A trigger is a type of stored procedure, that automatically runs when an event occurs in the database server.
- Here, events are DML operations (INSERT, DELETE, UPDATE).

Types

1. **DDL Trigger**
   - DDL trigger is fired when DDL statements like Drop Table, Create Table or Alter Table occurs. DDL Triggers are always After Triggers.
2. **DML Trigger**
   - We can create triggers on DML statements (like INSERT, UPDATE and DELETE) and Stored Procedures. DML Triggers are of **three** types.

1. **AFTER triggers**
   - These triggers executes after the action of the INSERT, UPDATE, MERGE, or DELETE statement is performed.
2. **INSTEAD OF triggers**
   - These triggers overrides the standard actions of the triggering statement.
   - It can be used to perform error or value checking on one or more columns
   - These triggers perform additional actions before insert, updating or deleting the row or rows.

You will learn ...

- Create Trigger
- Alter Trigger
- Drop Trigger
- Disable Trigger

T-SQL

54

DML Triggers

Part-I

ishwar.academy

Website

DML Trigger

- A trigger is a type of stored procedure, that automatically runs when an event occurs in the database server.
- Here, events are DML operations (INSERT, DELETE, UPDATE).

Types

1. **AFTER Triggers**
   - These triggers are executed after the event of the INSERT, UPDATE, MERGE, or DELETE statement is performed.
2. **INSTEAD OF Triggers**
   - These triggers override the standard events of the triggering statement.
   - It can be used to perform error / value checking on one or more columns.
   - Perform additional actions before insert, updating or deleting the row or rows.
3. **CLR Triggers**
   - It can be either an **AFTER or INSTEAD OF** trigger.
   - It can also be a **DDL** trigger.

Syntax

```
CREATE TRIGGER trigger_name

ON { table | view }

[ WITH DML_trigger_option ]

{ FOR | AFTER | INSTEAD OF }

{ [ INSERT ] , [ UPDATE ] , [ DELETE ] }

AS

BEGIN

    { SQL Statement or Statement Block }

END;
```

Before creating a trigger, things to know...

Magic Tables

SQL Server automatically creates and manages magic tables. DML trigger statements use two magic tables.

1. **Inserted Table**
   - This table stores copies of the affected rows during INSERT and UPDATE statements. During these transactions, **new rows are added to both the inserted table and the trigger table**.
2. **Deleted Table**
   - This table stores copies of the affected rows during DELETE and UPDATE statements. During the execution of these statements, **rows are deleted from the trigger table and stored into the deleted table**.

Example: Create a trigger when new employee added to employee table.

```
CREATE TRIGGER tr_message
ON employee
AFTER INSERT
AS
BEGIN
    PRINT 'New employee added to the Employee Table';
END;
```

T-SQL

55

DML Triggers

Part-II

ishwar.academy

Website

T-SQL

56

DML Triggers

Part-III

ishwar.academy

Website

T-SQL

57

DDL Triggers

Part-I

DDL Trigger

- DDL triggers fire in response to a different **DDL events** correspond to SQL statements such as **CREATE**, **ALTER**, **DROP**, **GRANT**, **REVOKE** etc.
- Also, some system stored procedures that perform **DDL-like operations** (for example, **sp_rename**) can also fire DDL triggers.

- **DDL events Reference: https://bit.ly/3I1EHay**

Why DDL Trigger?

- Prevent certain changes to your database schema.
- Have something occur in the database in response to a change in your database schema.
- Record changes or events in the database schema.

Syntax

```
CREATE TRIGGER trigger_name

ON { ALL SERVER | DATABASE }

[ WITH DDL_trigger_option ]

{ FOR | AFTER } { event_type1, event_type2, . . . }

AS

BEGIN

    { SQL Statement or Statement Block }

END;
```

Example: Create a trigger when new employee added to employee table.

```
CREATE TRIGGER tr_onTableCreate
ON DATABASE
FOR CREATE_TABLE
AS
BEGIN
    PRINT 'New table is created successfully';
END;
```

T-SQL

58

DDL Triggers

Part-II

DDL Trigger

- DDL triggers fire in response to a different **DDL events** correspond to SQL statements such as **CREATE**, **ALTER**, **DROP**, **GRANT**, **REVOKE** etc.
- Also, some system stored procedures that perform **DDL-like operations** (for example, **sp_rename**) can also fire DDL triggers.

- **DDL events Reference: https://bit.ly/3I1EHay**

EVENTDATA( )

- It is a built-in function.
- It returns the information about the events executed the DDL trigger.
- The information is in XML format.

- **EventType** (Create Table, Alter Table, Drop Table, etc…)
- **PostTime** (Event trigger time)
- **SPID** (SQL Server session ID)
- **ServerName** (SQL Server instance name)
- **LoginName** (SQL Server Login name)
- **UserName** (username for login, by default dbo schema as username)
- **DatabaseName** (name of database where DDL Trigger was executed)
- **SchemaName** (schema name of the table)
- **ObjectName** (Name of the table)
- **ObjectType** (Object types. such as Table, view, procedure, etc…)
- **TSQLCommand** (Schema deployment Query which is executed by user)
- **SetOptions** (SET Option which are applied while Creating table or Modify it)
- **CommandText** (Create, Alter or Drop object command)

Syntax

```
CREATE TRIGGER trigger_name

ON { ALL SERVER | DATABASE }

[ WITH DDL_trigger_option ]

{ FOR | AFTER } { event_type1, event_type2, . . . }

AS

BEGIN

    { SQL Statement or Statement Block }

END;
```

T-SQL

59

Merge

ishwar.academy

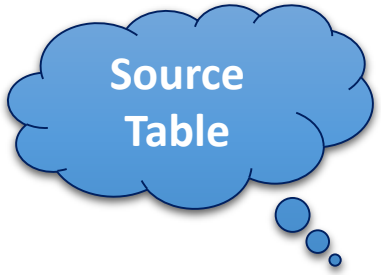
Website

Merge

- **Merge** is a logical combination of an insert and an update.
- It combines the sequence of conditional **INSERT**, **UPDATE**, and **DELETE** statements in a single statement.
- Using Merge statement, you can sync two different tables so that the content of the **target table** is modified based on differences found in the **source table**.

Why Merge?

- It is specially used to maintain a history of data in data warehousing during the ETL (**E**xtract, **T**ransform, **L**oad) cycle.
- **Scenario**: Suppose, tables need to be refreshed periodically with new data arriving from online transaction processing (OLTP) systems. This new data may contain changes to existing rows in tables and/or new rows that need to be inserted.



CHECK-IN

FIRSTNAME	FLIGHTCODE	FLIGHTDATE	SEAT
STEVE	SQL2022	2022-03-27	7F
ADAM	SQL2022	2022-03-27	19A
JOHN	SQL2022	2022-03-27	4B
MIKE	SQL2022	2022-03-27	20A

FLIGHT-PASSENGER

FLIGHTID	FIRSTNAME	FLIGHTCODE	FLIGHTDATE	SEAT
1	STEVE	SQL2022	2022-03-27	7F
2	ADAM	SQL2022	2022-03-27	19A
3	JOHN	SQL2022	2022-03-27	20B
4	MIKE	SQL2022	2022-03-27	2A

Actions / Conditions

1. Rows in the **source table** are not found in the **target table**. Then, rows from the source will be added to the target table.
2. Rows in the **target table** are not found in the **source table**. Then, delete rows from the target table.
3. Rows in the **source and target table** have the same keys but, they have different values in the non-key columns. Then, update the rows in the target table with data from the source table.

Syntax

```
MERGE TargetTable

USING SourceTable

ON join-conditions

WHEN Matched

        THEN DML Statement

WHEN NOT MATCHED BY TARGET

        THEN DML Statement

WHEN NOT MATCHED BY SOURCE

        THEN DML Statement;
```

It must be terminated by a semicolon.

States

1. **MATCHED**: These are the rows that match the merge condition. For the matching rows, you need to update the rows columns in the target table with values from the source table.
2. **NOT MATCHED BY TARGET**: These are the rows from the source table that does not have any matching rows in the target table. In this case, you need to add the rows from the source table to the target table.
3. **NOT MATCHED BY SOURCE**: These are the rows in the target table that does not match any rows in the source table. If you want to synchronize the target table with the data from the source table, then you will need to use this match condition to delete rows from the target table.

Syntax

```
MERGE TargetTable

USING SourceTable

ON join-conditions

WHEN Matched

    THEN DML Statement

WHEN NOT MATCHED BY TARGET

    THEN DML Statement

WHEN NOT MATCHED BY SOURCE

    THEN DML Statement;
```

```
MERGE TargetTable

USING SourceTable

ON join-conditions

WHEN Matched

    THEN Update

WHEN NOT MATCHED BY TARGET

    THEN Insert

WHEN NOT MATCHED BY SOURCE

    THEN Delete;
```

MS SQL SERVER

60

INDEX

WELCOME TO

ISHWAR ACADEMY

ishwar.academy

Website

Index

- In general, **index** is used to measure the performance.
- Database systems uses indices to provide fast access to relational data.
- It is a special type of physical data structure used to access one or more data rows fast.
- Database index can change each time the corresponding data is changed.

Differences of Index

1. **Book Index**
   - ✓ A book reader can decide whether or not to use the book's index.
   - ✓ A particular book's index is edited together with the book and does not change at all. This means that you can find a topic exactly on the page where it is determined in the index.
2. **Database Index**
   - ✓ The system component called the query optimizer decides whether or not to use an existing index.
   - ✓ A database index can change each time the corresponding data is changed.

Example Find employee names whose salary is greater than 50000.

ID	NAME	SALARY
1001	SMITH	65000
1002	JOHN	30000
1003	MIKE	48000
1004	JACK	52000

SALARY	ROW ADDRESS
30000	ROW ADDRESS
48000	ROW ADDRESS
52000	ROW ADDRESS
65000	ROW ADDRESS

```
SELECT * FROM EMPLOYEE
WHERE SALARY > 50000;
```

Syntax

```
CREATE INDEX index_name ON table_name;
```

Example Find employee names whose salary is greater than 50000.

ID	NAME	SALARY
1001	SMITH	65000
1002	JOHN	30000
1003	MIKE	48000
1004	JACK	52000

SALARY	ROW ADDRESS
30000	ROW ADDRESS
48000	ROW ADDRESS
52000	ROW ADDRESS
65000	ROW ADDRESS

```
SELECT * FROM EMPLOYEE
WHERE SALARY > 50000;
```

```
CREATE INDEX IDX_EMPLOYEE_SALARY
ON EMPLOYEE(SALARY ASC);
```

WELCOME TO

ISHWAR ACADEMY

MS SQL SERVER

61

CLUSTERED INDEX

ishwar.academy

Website

Clustered Index

- A **clustered index** determines the physical order of the data in a table. Hence, a table can have only one clustered index.
- When a clustered index is created, the database engine sorts the data in the table based on the defined index key(s) and stores the table in that order.
- **Example**, a telephone book. A telephone book is always in sorted order, based on the last name of the individual followed by the first name. The sorted order makes it easy to find the phone number of the person you are looking for.

Things to remember...

- ✓ A **Primary key** constraint creates clustered index automatically if there is no clustered index exists on the table.
- ✓ An index can contain multiple columns, known as **composite index** (we will see this in later video).

Syntax

```
CREATE CLUSTERED INDEX index_name
ON table_name(column_name <ASC | DESC> );
```

```
CREATE CLUSTERED INDEX idx_employee_name
ON employee(name ASC);
```

WELCOME TO

ISHWAR ACADEMY

MS SQL SERVER

62

NONCLUSTERED INDEX

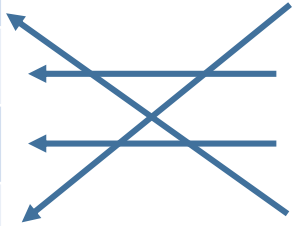
ishwar.academy

Website

Nonclustered Index

- A **nonclustered index** does not change the physical order of the rows in the table.
- In other (simple) words, A nonclustered index is similar to an index in a textbook. The data is stored in one place, the index in another place. The index will have pointer to the storage location of the data.
- Since, the index is stored separately from the actual data, a table can have more than one nonclustered index.

ID	SALARY	LOCATION	NAME
1003	65000	USA	SMITH
1002	30000	INDIA	JOHN
1001	48000	INDIA	MIKE
1004	52000	USA	JACK



NAME	ROW ADDRESS
JACK	ROW ADDRESS
JOHN	ROW ADDRESS
MIKE	ROW ADDRESS
SMITH	ROW ADDRESS

Syntax

```
CREATE NONCLUSTERED INDEX index_name
ON table_name(column_name <ASC | DESC> );
```

```
CREATE NONCLUSTERED INDEX idx_employee_name
ON employee(name ASC);
```

IMPORTANT: It is faster in searching for the values that are not in the range.

WELCOME TO

ISHWAR ACADEMY

MS SQL SERVER

63

UNIQUE INDEX

ishwar.academy

Website

Unique Index

- A **unique index** ensures that the index key contains no duplicate values.
- There are no differences between creating a **UNIQUE constraint** and creating a **unique index**.
- Creating a UNIQUE constraint on the column makes the purpose of the index clear.

Implementation

1. **PRIMARY KEY**
   - When you create a **PRIMARY KEY constraint**, a **unique clustered index** on the column or columns is automatically.
2. **UNIQUE constraint**
   - When you create a **UNIQUE constraint**, a **unique nonclustered index** is created to enforce a UNIQUE constraint by default.
3. **Index independent of a constraint**
   - Multiple unique nonclustered indexes can be defined on a table.

Syntax

```
CREATE UNIQUE NONCLUSTERED INDEX index_name
ON table_name(column_name);
```

```
CREATE UNIQUE NONCLUSTERED INDEX
idx_employee_id ON employee(id);
```

IMPORTANT: It is faster in searching for the values that are not in the range.

MS SQL SERVER

64

VIEW

WELCOME TO

ISHWAR ACADEMY

ishwar.academy

Website

View

- A view is a **virtual table** whose contents are defined by a query.

Does this make sense ?

Let's make it Simple

View

- A **view** does not require any storage in a database.
- It is a **Saved Query.** It acts as a filter on the tables referenced in the view query.
- The main use case of a view to **maintain the security at row and column level**.

Syntax

```
CREATE VIEW view_name
AS [simple select query] | [ join query ];
```

```
CREATE VIEW v_EmployeeDetails
AS SELECT empName, empSal, loc FROM employee;
```

MS SQL SERVER

65

CTE

WELCOME TO

ISHWAR ACADEMY

ishwar.academy

Website

CTE (Common Table Expression)

- **CTE** is a temporary named result set.
- A CTE must be followed by a single SELECT, INSERT, UPDATE, or DELETE statement that references some or all the CTE columns. A CTE can also be specified in a CREATE VIEW statement as part of the defining SELECT statement of the view.
- Multiple CTE query definitions can be defined.

  Note: The definitions must be combined by one of these set operators: UNION ALL, UNION, INTERSECT, or EXCEPT.

Syntax

```
WITH cte_name AS ( cte_query ) followed_query
```

```
WITH cte_name(col1, col2, ....) AS ( cte_query )
followed_query
```

```
WITH cte_name AS ( cte_query ) followed_query
```

```
WITH cte_avgSalary AS
(
        SELECT avg(salary) as AvgSalary FROM employee
)
SELECT AvgSalary FROM cte_avgSalary;
```

```
WITH cte_name(col1, col2, ....) AS ( cte_query )
followed_query
```

```
WITH cte_empCount(deptid, employeeCount) AS
(
        SELECT deptid, count(*) AS employeeCount
        FROM employee group by deptid
)
SELECT dname, employeeCount FROM
department JOIN cte_empCount
ON department.did = cte_empCount.deptid;
```

MS SQL SERVER

66

WELCOME TO

ISHWAR ACADEMY

Transaction Control Language

ishwar.academy

Website

Transaction Control Language

- If a transaction is successful, all of the data modifications made during the transaction are committed and become a permanent part of the database. If a transaction encounters errors and must be canceled or rolled back, then all of the data modifications are erased.
- **COMMIT**
- **ROLLBACK**
- **SAVE**

Syntax

BEGIN TRANSACTION;

SQL statements

COMMIT | ROLLBACK | SAVE

MS SQL SERVER

67

WELCOME TO

ISHWAR ACADEMY

Backup & Restore Database

ishwar.academy

Website