Understanding the SQL Server TOP Clause: A Step-by-Step Tutorial

The SQL Server “TOP” clause is a powerful tool for limiting the number of rows returned by a query or affected by an “UPDATE” or “DELETE” operation. This guide will teach you how to use the “TOP” clause effectively, focusing on several practical scenarios. Let's dive into how it works by creating a sample database and experimenting with different use cases.

This document is an explanation of [Top\_Clause](https://github.com/selsoftdf/SQL-Server-Development/blob/main/2005/Top_Clause.sql) file which I prepared and locate in my GitHub repository. You can download and see all samples because you may not see some samples in this document.

## Setting Up the Environment

Before we begin, you will need to create a database and generate some sample data. Follow these steps to prepare the environment:

### Step 1: Create a Database

|  |
| --- |
| CREATE DATABASE Top\_Sample; |

### Step 2: Create a Table and Insert Sample Data

|  |
| --- |
| USE Top\_Sample; GO  CREATE TABLE Projects (  Project\_ID VARCHAR(20) NOT NULL,  Project\_Name VARCHAR(50),  Progress INT,  City VARCHAR(20) );  INSERT INTO Projects VALUES  ('XYZ-1', 'Project 1', 10, 'Riyadh'),  ('XYZ-2', 'Project 2', 11, 'Jeddah'),  ('XYZ-3', 'Project 3', 20, 'Riyadh'),  ('XYZ-4', 'Project 4', 20, 'Jeddah'),  ('XYZ-5', 'Project 5', 40, 'Riyadh'),  ('XYZ-6', 'Project 6', 50, 'Dammam'),  ('XYZ-7', 'Project 7', 55, 'Riyadh'),  ('XYZ-8', 'Project 8', 65, 'Riyadh'),  ('XYZ-9', 'Project 9', 70, 'Riyadh'),  ('XYZ-10', 'Project 10', 80, 'Riyadh');  ALTER TABLE Projects ADD CONSTRAINT PK\_Projects PRIMARY KEY(Project\_ID); |

## Using the TOP Clause in SQL Server

The “TOP” clause is supported in SQL Server starting from version 2005.

Now that we have our environment ready, let’s explore different use cases for the “TOP” clause in SQL Server.

### Sample 1: Retrieving the Top 1 Row

|  |
| --- |
| SELECT TOP 1 Project\_ID, Project\_Name, Progress, City FROM Projects; |

This query retrieves the first row based on the default order of the data.

### Sample 2: Retrieving the Top N Rows with Ordering

|  |
| --- |
| SELECT TOP 2 Project\_ID, Project\_Name, Progress, City FROM Projects ORDER BY Progress ASC; |

This query orders the rows by the “Progress” column in ascending order and returns the first two rows with the lowest progress values.

### Sample 3 & 4: Using Parentheses with “TOP”

|  |
| --- |
| DECLARE @n INT = 3;  SELECT TOP (@n) Project\_ID, Project\_Name, Progress, City FROM Projects ORDER BY Progress ASC; |

In this example, the number of rows returned is determined by the value of the variable “@n”. This allows for more flexibility when the number of rows is not fixed.

### Sample 5: Using “TOP” with Ties

|  |
| --- |
| SELECT TOP 3 WITH TIES Project\_ID, Project\_Name, Progress, City FROM Projects ORDER BY Progress ASC; |

This query will include all projects with the same “Progress” as the third one, even if there are more than three rows. This is helpful when you want to ensure that you do not exclude rows with the same ranking value.

### Sample 6 & 7: Using “TOP” with Percent and Ties

|  |
| --- |
| DECLARE @x INT = 40;  SELECT TOP (@x) PERCENT Project\_ID, Project\_Name, Progress, City FROM Projects ORDER BY Progress ASC; |

This query returns the top 40% of rows from the “Projects” table, ordered by “Progress”. You can combine “PERCENT” with “WITH TIES” to include additional rows that match the cutoff value.

## Updating and Deleting Rows Using “TOP”

### Sample 8: Updating the Top N Rows

|  |
| --- |
| UPDATE TOP(3) Projects SET Project\_Name = Project\_Name + '00'; |

This query updates the “Project\_Name” of the top 3 rows in the “Projects” table by appending '00' to the existing project names.

### Sample 9: Updating Rows in a Specific Order

|  |
| --- |
| UPDATE Projects SET Project\_Name = tbl.Project\_Name + '00' FROM (SELECT TOP 3 \* FROM Projects ORDER BY Progress DESC) AS tbl WHERE Projects.Project\_ID = tbl.Project\_ID AND Projects.City = tbl.City; |

In this example, the top 3 rows are selected based on the highest “Progress” values, and then updated accordingly. This ensures that the rows are updated in a meaningful order.

### Sample 10: Deleting the Top Row

|  |
| --- |
| DELETE TOP (1) FROM Projects WHERE Project\_Name LIKE '%00'; |

This query deletes the first row that has a “Project\_Name” ending with '00'.

### Sample 11: Deleting Multiple Rows in a Specific Order

|  |
| --- |
| DELETE FROM Projects WHERE Project\_ID IN (SELECT TOP (2) Project\_ID FROM Projects ORDER BY Project\_ID DESC); |

This query deletes the top 2 rows based on the descending order of “Project\_ID”.

### Sample 12: Deleting a Percentage of Rows

|  |
| --- |
| DELETE TOP(50) PERCENT FROM Projects; |

This query deletes 50% of the rows from the “Projects” table.

## Cleaning Up: Dropping the Database

Once you are done experimenting, you can drop the database to clean up your environment:

|  |
| --- |
| USE master; GO DROP DATABASE Top\_Sample; |

## Conclusion

The “TOP” clause in SQL Server is a versatile tool for controlling the number of rows affected by a query. By practicing these examples step by step, you should now have a solid understanding of how to use “TOP” effectively in various scenarios, whether for querying, updating, or deleting data.