

1. Introduction to Problem Solving (C#

Context)

1.1 What is Problem Solving in Programming?

Problem solving in programming is the process of:

Understanding a real-world problem,

Designing an efficient logical approach (algorithm),

Translating it into a working program (code),

Testing and validating the solution.

1.2 Importance of Algorithms

An algorithm is a step-by-step method for solving a problem.

Why are algorithms important?

Efficiency: A bad algorithm can make a program slow, even if the code is bug-

free.

Reusability: Good algorithms can be used across different programs.

Scalability: Helps your program handle large inputs gracefully.

1.3 Characteristics of a Good Algorithm

A good algorithm typically has:

Correctness: It should solve the problem properly.

Efficiency: Minimal time and space usage.

Finiteness: It should complete in a finite number of steps.

Clarity: Easy to understand and implement.

Generality: Works for all valid inputs, not just a few cases

1.5 Example: Solving a Simple Problem (Step-by-Step)

Problem:

Write a program to calculate the sum of two numbers entered by the user.

Step 1: Understand Inputs/Outputs

Input: Two integers

Output: Sum of the two integers

Step 2: Write Algorithm (in English)

1. Ask the user for the first number.

2. Ask the user for the second number.

3. Add both numbers.

4. Display the result.

Step 3: Write Pseudocode

START

Input number1

Input number2

sum = number1 + number2

Display sum

END

Step 4: Implement in C#

using System;

class Program

{

static void Main()

{

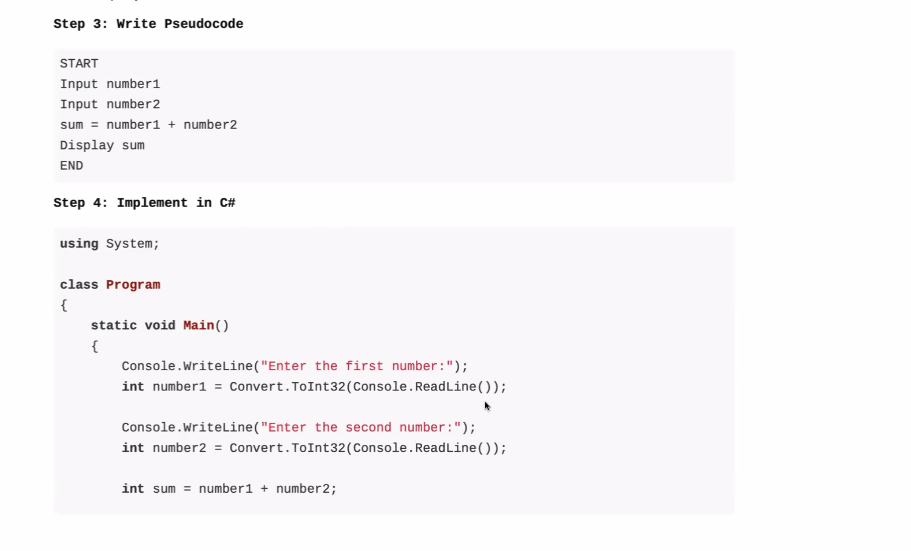
Console.WriteLine("Enter the first number:");

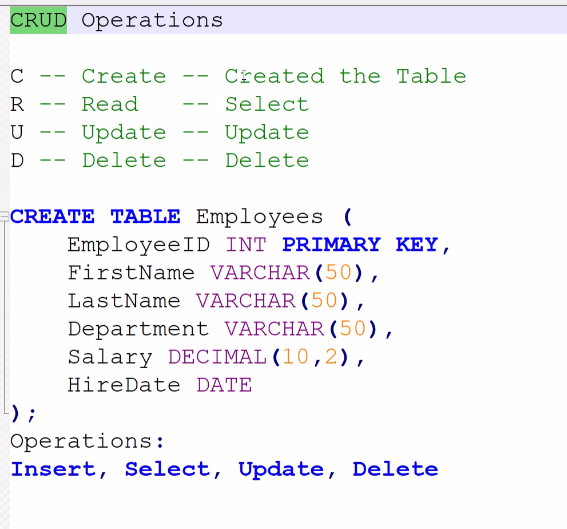
int number1 = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Enter the second number:");

int number2 = Convert.ToInt32(Console.ReadLine());

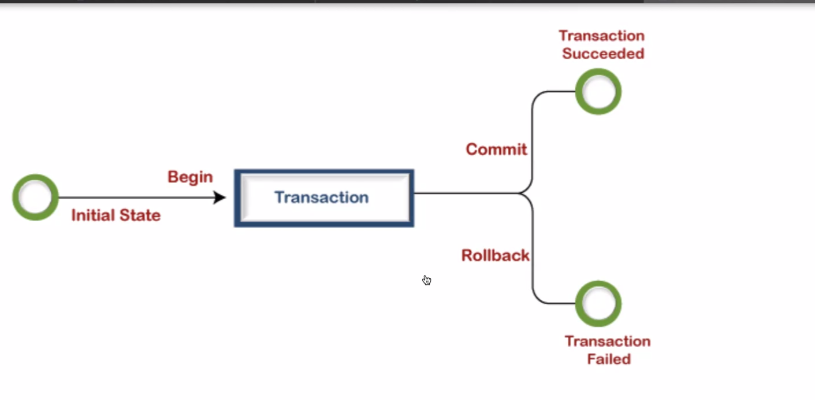
int sum = number1 + number2;



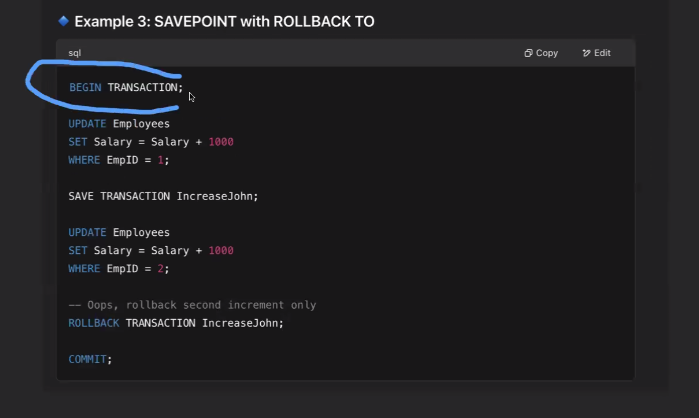


**Transaction-**

1. **Transaction States**:
   * **Begin**: Starting point of a transaction
   * **Initial State**: Before any operations are performed
   * **Succeeded**: All operations completed successfully
   * **Failed**: One or more operations encountered errors
2. **Transaction Control Commands**:
   * **COMMIT**: Permanently saves all changes made during the transaction
   * **ROLLBACK**: Reverts all changes made during the transaction



Savepoint Key Features:



1. Transaction Control

- `BEGIN TRANSACTION` starts the transaction block

- `COMMIT` finalizes all changes not rolled back

2.Savepoint Usage:

- `SAVE TRANSACTION IncreaseJohn` creates a restore point

- `ROLLBACK TRANSACTION IncreaseJohn` undoes only changes after the savepoint

3. Effect:

- EmpID 1 will get +1000 salary (kept)

- EmpID 2's salary increase will be rolled back

- Other employees remain unchanged

4. Use Case:

- Useful when you need to undo only part of a transaction

- Helps in complex transactions with multiple steps where some may need reversal

In standard SQL this is called SAVEPOINT, but SQL Server uses `SAVE TRANSACTION` syntax.

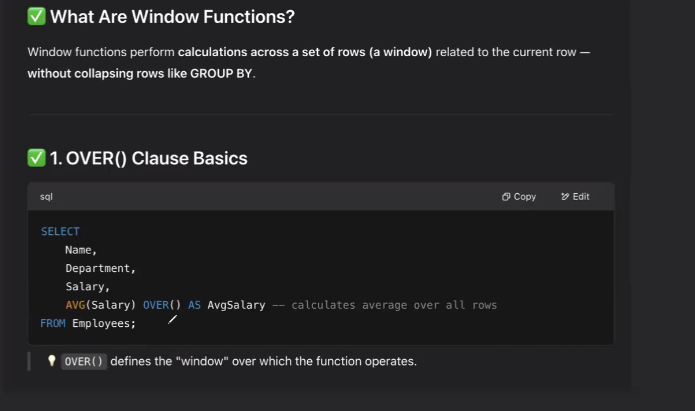
# Window Functions Overview

## What Are Window Functions?

Window functions perform calculations across a set of rows (a window) related to the current row while maintaining individual rows (unlike GROUP BY which collapses rows).

**OVER() Clause Basics**

The OVER() clause defines the "window" of rows that the function operates on. For example:



**Key characteristics:**

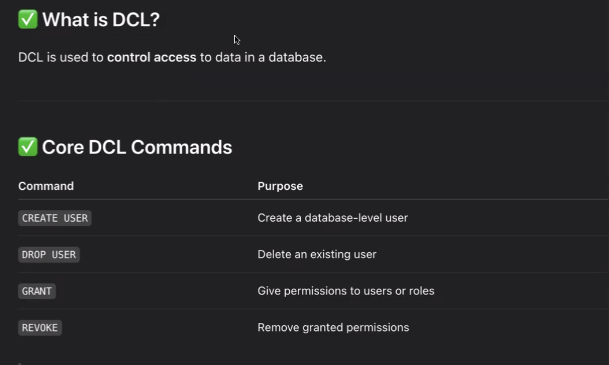
- Maintains original row structure while adding computed columns

- OVER() without partitioning operates on the entire result set

- Different from aggregate functions with GROUP BY which reduce rows

- Enables calculations like running totals, rankings, and moving averages

\*\*similar to group by



Delegates

