About Technoglobe

Technoglobe is Leading IT Training Company of India working for IT Training, Skilling & Placement of Students since year 2001. Technoglobe has trained & placed a huge number of students in various sectors like Digital Marketing, Graphic Designing, Accounting, Video Editing, Web Development with Java Python &
PHP, Data Analytics, Data Sciences, Adv Excel, Networking, Cyber Security, Devops, Generative AI & many more technologies.
It has been awarded more than 30 times for its Quality Education & Placements at National & International platforms. It is one of the very few IT Training Companies in India that are awarded at Oxford University UK. Technoglobe has more than 125+ centers in India, UAE, UK, Canada & Singapore.
As part of its Strong Placement Support Technoglobe has done 500+ tie ups with various IT & Non IT companies & adding more companies to it.
If you are not willing to learn, no one can help you. If you are determined to learn, no one can stop you.

Message from Team Technoglobe

Dear Students,
IT skilling is crucial for India as it significantly contributes to the nation's economic growth by powering the rapidly expanding IT sector, generating substantial employment opportunities, driving innovation, and enabling India to compete effectively in the global market, making it one of the key. pillars of the Indian economy
Skilled IT professionals are essential for driving innovation in various sectors, including IT, healthcare, finance, Banking and manufacturing through technology adoption.

We at Technoglobe bridge the gap between the requirement of companies & skills of the students.
Our job oriented Training programs makes the students employable & industry ready.

About the Book

This book is a comprehensive self-learning guide designed to equip aspiring and professional data analysts with the essential skills, tools, and analytical mindset required in today's data-driven world. Whether you are a beginner stepping into the field of data analytics or a working professional aiming to strengthen your technical and analytical expertise, this guide offers structured, step-by-step learning supported by practical examples and real-world datasets.

The book covers the complete data analytics lifecycle — from understanding data fundamentals, statistics, and data cleaning to advanced topics such as data visualization, exploratory data analysis, business intelligence, and data-driven decision making. It focuses on industry-relevant tools and technologies including Excel, SQL, Python, Power BI, Tableau, and an introduction to AI-driven analytics and automation in data workflows.

Each chapter is thoughtfully designed to gradually build proficiency through hands-on exercises, project-based learning, case studies, and real business scenarios. Readers will gain practical experience in areas such as data manipulation, statistical analysis, dashboard creation, reporting, trend analysis, and storytelling with data. Special emphasis is placed on real-world applications across industries like finance, marketing, operations, and product analytics.

The content is aligned with current industry trends in analytics, business intelligence, AI integration, and digital transformation, ensuring learners are job-ready and confident to handle real-world data challenges. Whether your goal is to become a data analyst, business analyst, BI professional, freelancer, or corporate data professional, this book provides both foundational knowledge and advanced insights required for long-term success.

This guide is developed by Technoglobe, a leading IT and multimedia training institute recognized for Quality Education and Strong Placement Support, with over 125+ centers and a proven legacy of training thousands of students since 2001.

Index

45. Data Selection, Filtering, and Sorting
46. Data Cleaning – Missing Values, Duplicates, and Type Conversion
47. String Operations and Column Creation
48. Merging, Joining, and Concatenating Data
49. Exploratory Data Analysis (EDA)
50. Visualization with Matplotlib and Seaborn
51. Power BI Interface and Ecosystem
52. Connecting to Data Sources in Power BI
53. Power Query Editor Basics
54. Creating Visualizations – Bar, Pie, and Column Charts
55. Using Filters and Slicers
56. Data Modeling and Relationships
57. Star vs Snowflake Schema
58. Calculated Columns vs Measures
59. DAX Basics – SUM, AVERAGE, COUNT, DISTINCTCOUNT
60. Logical and Date Functions in DAX
61. Time Intelligence – YTD, MTD, QTD
62. Advanced Data Cleaning in Power Query
63. Column Splits, Merges, and Custom Columns
64. Report Design – Layouts, Bookmarks, Buttons
65. Drillthrough and Conditional Formatting
66. Sharing, Refresh, and Gateways in Power BI
67. Mobile App & Dashboards
68. Excel, SharePoint, and Power Automate Integration
69. AI Visuals and Power BI APIs
70. Introduction to Tableau and Workspace Overview
71. Connecting to Data Sources in Tableau
72. Dimensions vs Measures
73. Basic Charts in Tableau
74. Sorting, Filtering, and Grouping in Tableau
75. Publishing Dashboards in Tableau
76. Data Interpreter, Pivoting, Joins & Unions in Tableau
77. Groups, Sets, Hierarchies, and Calculated Fields
78. Advanced Charts in Tableau
79. Parameters and Dual-Axis Charts
80. Dashboard Design and Interactivity
81. Story Points and Mobile View Design
82. Forecasting, Trend Lines, and Reference Lines
83. Level of Detail (LOD) Calculations
84. Geo Maps in Tableau
85. Cluster Analysis in Tableau
86. Tableau Prep for Data Cleaning
87. Exporting Reports and Publishing in Tableau
88. Final Tableau Project

Chapter 1: Excel Basics & Interface Overview

Introduction: This chapter introduces Microsoft Excel in a simple and easy-to-follow manner. It explains what Excel is, how to open it, and how to use its basic features. The goal is to help new users become comfortable with the Excel interface and begin using it with confidence, even if they have never used it before.

1.1 Understanding Microsoft Excel

Microsoft Excel is a spreadsheet program created by Microsoft. It allows users to enter, organize, calculate, and analyze various types of information.
It is widely used in different fields such as:

- **Education (for student marks and attendance)**
- **Business (for accounts, sales reports, and inventory)**
- **Household management (for monthly expenses or personal budgeting)**

Key features of Excel include:

- **Grid Layout: Excel displays data in a table format using rows and columns.**
- **Data Entry: You can enter names, numbers, dates, or formulas into the cells.**
- **Automatic Calculations: Excel can quickly add, subtract, multiply, or divide numbers using built-in formulas.**
- **Data Analysis: You can sort, filter, and search data easily.**
- **Charts and Graphs: Excel helps create visual representations of data for better understanding.**
- **Formatting Tools: You can format tables with colors, borders, fonts, and cell styles to make the data easier to read.**

Common uses of Excel:

- **Creating a monthly budget with income and expenses**
- **Recording and calculating student marks**
- **Making a to-do list or timetable and Keeping a sales report or stock inventory**
- **Summarizing survey results or data from forms**

1.2 Opening Excel

To begin working with Excel, the program must be opened.
Steps to open Excel:

- **Click on the Start button (usually found in the bottom-left corner of the computer screen).**
- **Type "Excel" in the search bar.**
- **Click on the Microsoft Excel icon that appears in the list.**

Once Excel opens, a welcome screen appears. This screen contains options such as:

- **Blank Workbook: Opens a new, empty worksheet.**
- **Templates: These are ready-made designs such as calendars, invoices, attendance sheets, etc.**
- **Recent Files : Shows a list of Excel files that were recently opened or worked on.**

To begin learning, it is best to select Blank Workbook.

1.3 Overview of the Excel Window

When a new workbook is opened, Excel displays a large grid of boxes. These are called cells. The window has many useful parts, each serving a specific purpose.
Workbook : A workbook is the entire Excel file. Just like a physical notebook can have many pages, an Excel workbook can contain multiple worksheets.
Worksheet: A worksheet is a single sheet or page in Excel where you enter data. Each worksheet has rows (horizontal) and columns (vertical). The default name of a worksheet is Sheet1.
You can rename a worksheet by right-clicking on the sheet tab and selecting Rename.
Cells : Each small box where data is entered is called a cell. Every cell has an address made up of a column letter and a row number.

For example:

- **The top-left cell is A1**
- **The cell in column B, row 3 is B3**

Each cell can contain:

- **Text (e.g., "Name", "Total")**
- **Numbers (e.g., 50, 99.99)**
- **Dates (e.g., 01/01/2025)**
- **Formulas (e.g., =A1+B1)**

Columns and Rows

- **Columns are vertical and are labeled with letters like A, B, C& and so on.**
- **Rows are horizontal and are labeled with numbers like 1, 2, 3&**
- **The combination of a column and row creates a cell address, such as D5.**

Ribbon : The Ribbon is the main toolbar at the top of the window. It contains tabs like Home, Insert, Page Layout, Formulas, Data, and more.

Each tab has groups of tools:

- **The Home tab includes buttons for changing font, size, alignment, number format, and applying styles.**
- **The Insert tab lets you add tables, charts, pictures, and more.**
- **The Formulas tab gives access to Excel9s built-in functions like SUM, AVERAGE, IF, etc.**

Formula Bar : The Formula Bar is found just below the Ribbon. It displays the contents of the currently selected cell. You can also use it to type or edit data and formulas.

Name Box : Located to the left of the Formula Bar, the Name Box shows the cell address of the active (selected) cell. For example, if cell D4 is selected, the Name Box will show "D4".

Sheet Tabs : At the bottom of the screen, you will find Sheet Tabs such as Sheet1, Sheet2, etc. These help organize your data across multiple pages in one file.

You can:

- **Click the + sign to add a new sheet**
- **Right-click to rename, delete, or move a sheet**

Status Bar : At the very bottom of the Excel window is the Status Bar. It provides useful information like:

- **The sum or average of selected numbers**
- **Zoom level control (to zoom in or out of your sheet)**
- **Cell mode (Ready, Edit, Enter)**

1.4 Moving Around in Excel

It is easy to move between different cells in Excel.

Using the Mouse:

- **Click on a cell to select it.**
- **Double-click to start editing the contents directly in the cell.**

Using the Keyboard:

- **Arrow keys move the selection up, down, left, or right.**
- **Enter moves to the cell below.**
- **Tab moves to the cell on the right.**
- **Ctrl + Arrow Keys jump to the edges of a data range (e.g., the last cell with data in a row or column).**
- **Ctrl + Home takes you to the top-left cell (A1).**

These shortcuts help work faster, especially in large sheets.

1.5 Typing and Editing Data

Excel allows different types of data in each cell.

Common data types:

- **Text: Labels or words (e.g., "Name", "Subject", "Total")**
- **Numbers: Quantities or values (e.g., 200, 3.75)**
- **Dates: Calendar dates (e.g., 10/12/2025)**
- **Formulas: Calculations (e.g., =A2+B2, =SUM(A1:A5))**

To enter data:

- **Click on a cell.**
- **Type the text, number, or formula.**
- **Press Enter to confirm and move down to the next cell.**

To edit data:

- **Double-click on the cell, or**
- **Click the cell once and edit the content in the Formula Bar.**

Data in Excel is dynamic, meaning it can change automatically if formulas are used. For example, changing a number in cell A1 will update all formulas connected to A1.

1.6 Saving a File

It is important to save the Excel file to avoid losing work.

Steps to save:

- **Click on the File tab in the top-left corner.**
- **Select Save As.**
- **Choose the location (such as Desktop or Documents).**
- **Type a file name.**
- **Click Save.**

To save quickly, press Ctrl + S on the keyboard.

Common file types:

- **.xlsx: Default Excel format (recommended)**
- **.xls: Older format, useful for compatibility with old versions**
- **.csv: Simple format used for data export, without formatting**

Files can also be saved in PDF format for printing or sharing.

1.7 Closing Excel

After completing the work, Excel can be closed.

To close Excel:

- **Click the X button in the top-right corner of the window, or**
- **Press Alt + F4 on the keyboard**

If the file has not been saved, Excel will ask whether to save it before closing. Always choose save if you don't want to lose your work.

Summary

This chapter introduced the fundamental concepts of Microsoft Excel. The following topics were explained in details:

- **What Excel is used for and its common features.**
- **How to open Excel and start a new blank workbook.**
- **The parts of the Excel window, including workbook, worksheet, cells, rows, columns, ribbon and more.**
- **Methods of moving around in Excel using mouse and keyboard.**
- **How to enter and edit different types of data such as text, numbers, dates and formulas.**
- **How to save and close an Excel file safely.**

These basic are the foundation for learning Excel in the next chapter the focus will be on formatting worksheet which includes changing font style, colors, borders, alignment and making the data look clean and professional.

Chapter 2: Columns and Sheet Tabs

Introduction: Excel is made up of a big grid of small boxes called cells. These cells are arranged in rows and columns, and the entire grid is organized into different sheets inside a file. Understanding how rows, columns, and sheets work is the key to using Excel well.

2.1. Understanding Rows in Detail

What is a Row?

- **A row is a horizontal line of cells.**
- **Rows go side to side (left to right).**
- **Rows are numbered starting from 1 at the top then 2,3,4 and so on all the way down to over 1 million rows (1,048,576 exactly).**
- **The numbers are shown on the left side of the Excel window.**

Why are Rows Important?

Rows help you keep related data together on the same horizontal line. Usually, each row represents a single record or entry.

Example:

Imagine you have a list of books. Each book has information like title, author, and year published. You can use each row to store information about one book.

	A	B	C
	Title	Author	Year
1.	The Hobbit	J.R.R. Tolkien	1937
2.	1984	George Orwell	1949

- **Row 1 is often used as a header row, describing what the columns mean.**
- **Row 2 stores data for the first book, Row 3 for the second book, and so on.**

How to Select and Work with Rows

- **To select a row, move your mouse pointer to the number on the left of the row.**
- **Click once to highlight the entire row across all columns.**
- **You can also select multiple rows by clicking and dragging over multiple row numbers.**

After selecting, you can:

- **Enter or delete data. o Format the whole row (change color, font, etc.). o Insert a new row (to add data).**
- **Delete a row (to remove data).**

Inserting and Deleting Rows

To insert a new row:

- **Right-click on a row number.**
- **Choose Insert. A new blank row will be added above the row you selected.**

To delete a row:

- **Right-click on the row number. o Choose Delete. The entire row will be removed and rows below will move up.**

2.2. Understanding Columns in Detail

What is a Column?

- **A column is a vertical line of cells.**
- **Columns go up and down.**
- **Columns are labeled with letters starting at A, B, C ... up to Z.**
- **After Z, columns continue as AA, AB, AC ... up to XFD (which is the 16,384th column).**
- **The letters are shown at the top of the Excel window.**

Why are Columns Important?

Columns help you organize types of data. Each column usually stores one kind of information.

Example:

Continuing with the books list:

	A	**B**	**C**
	Title	Author	Year
1	The Hobbit	J.R.R. Tolkien	1937
2	1984	George Orwell	1949

- **Column A is for the book title.**
- **Column B is for the author9s name.**
- **Column C is for the year published.**

How to Select and Work with Columns

- **To select a column, click the letter at the top.**
- **The whole column from top to bottom will be highlighted.**
- **You can select multiple columns by clicking and dragging over the letters.**

After selecting, you can:

- **Change the width (make the column wider or narrower). o Format the whole column.**
- **Insert or delete columns.**

Inserting and Deleting

Columns

To insert a new column:

- **Right-click on the column letter.**
- **Choose Insert. A new blank column will be added to the left.**

To delete a column:

- **Right-click on the column letter.**
- **Choose Delete.**

2.3. What Are Cells? 4 The Small Boxes in Excel

Cell Address (Reference)

- **Each cell has a unique name called a cell reference.**
- **The reference is made by combining the column letter and row number.**
- **For example:**
  - **Cell at column A and row 1 is A1. O**
  - **Cell at column C and row 5 is C5.**

This helps you locate cells quickly.

What Can You Put in a Cell?

- **Text (words or sentences).**
- **Numbers (like 25, 100, 3.14).**
- **Dates (like 31/05/2025).**

- **Formulas (special instructions to calculate values).**

2.4. Sheet Tabs 4 Multiple Pages in One Workbook

What is a Sheet?

A sheet is like a single page inside your Excel file.

- **You can have many sheets in one file.**
- **Each sheet has its own rows, columns, and cells.**
- **Sheets are useful to organize different data types or projects in the same file.**

Sheet Tabs Location

- **Sheet tabs are located at the bottom of the Excel window.**
- **You can see default names like Sheet1, Sheet2, Sheet3.**
- **You can switch sheets by clicking on their tabs.**

How to Rename Sheet Tabs Right-click the sheet tab.

- **Click Rename.**
- **Type a new name (for example, Sales 2025 or Inventory).**
- **Press Enter.**

Adding New Sheets

- **Click the small + button next to the last sheet tab.**
- **A new blank sheet will appear.**

Deleting Sheets

- **Right-click the sheet tab you want to remove.**
- **Click Delete.**
- **Excel will warn you before deleting permanently.**

2.5. More Tips on Rows, Columns, and Sheets

Adjusting Column Width and Row Height

- **Sometimes text doesn9t fit inside a cell.**
- **To fix this, you can change the size:**
  - **Move your mouse to the line between two column letters.**
  - **When the pointer changes to a double-headed arrow, click and drag to make it wider or narrower.**
- **For rows, do the same on the line between row numbers on the left.**
- **Double-click the boundary to automatically resize based on content size (called AutoFit).**

Moving Around Cells Quickly

- Use arrow keys on your keyboard to move one cell at a time.
- Press Tab to move right.
- Press Enter to move down.
- Use Page Up and Page Down to scroll up and down faster.

Freezing Rows or Columns

- When working with large sheets, freezing lets you keep headers or important columns visible.
- Go to the View tab → Click Freeze Panes.
- You can freeze:
    - The top row.
    - The first column.
    - Or a custom number of rows and columns.

2.6. Practice Exercise: Try These

- Open Excel.
- Click on cell A1 and type Name.
- Click on cell B1 and type Age. 4. Click on cell C1 and type City.

Enter data below these headers in the rows (example: Alice, 25, London).

- Click the number 2 on the left to select row
- Click the letter B at the top to select column B.
- Right-click the sheet tab Sheet1, rename it to People.
- Add a new sheet and rename it Summary.
- Try changing column widths and row heights to fit the data.
- Freeze the top row so you can always see the headers when you scroll.

Summary

Term	What It Means	How to Identify	Why It Matters
Row	Horizontal line of cells	Number on the left side	Holds one full record or entry
Column	Vertical line of cells	Letter at the top	Organizes types of data
Cell	Single box where a row and column meet	Named by column + row (like A1)	Where you type data
Sheet Tab	One page inside the Excel workbook	Tabs at bottom	Organizes different tables/projects

Chapter 3: Basic Excel Projects

Introduction: You9ve already learned what rows, columns, cells, and sheets are. Now let9s use those skills to build real Excel files step by step. Each project here is something you could use at work, in school, or at home.

3.1. Project 1: Employee Salary Sheet

Objective:

To calculate each employee9s final (net) salary based on:

- **Basic Pay**
- **HRA (House Rent Allowance)**
- **DA (Dearness Allowance)**
- **Deductions (like tax or PF)**

Structure of the Table:

Column	Purpose
Name	Employee's name
Basic Pay	Base monthly salary
HRA	Money given for rent expenses
DA	Extra allowance (inflation-based)
Deductions	Reductions like PF, tax, etc.
Net Salary	Final salary = Total – Deductions

Step-by-Step in Excel:

Step 1: Set Up Headings

In row 1, type the headings:

- **A1 = Name**
- **B1 = Basic Pay**
- **C1 = HRA**
- **D1 = DA**
- **E1 = Deductions**
- **F1 = Net Salary**

Step 2: Enter Employee Data

	A	B	C	D	E	F
	Ramesh	30000	5000	3000	2000	=B2+C2+D2–E2
	Suresh	28000	4500	2800	1500	=B3+C3+D3–E3
	Priya	32000	6000	3200	2200	=B4+C4+D4–E4

Excel will calculate Net Salary for each employee using the formula.

Excel Tip:

- **Use the AutoFill feature: Type formula in F2, then drag the small square (bottom right corner of cell) down to copy the formula to other rows.**
- **Use Currency Format: Select salary cells → Home → Format → Currency.**

3.2. Project 2: Student Marksheet

Objective:

- **To calculate:**
- **Total Marks**
- **Percentage**
- **Result: Pass or Fail (using IF function)**

Table Structure:

Column	What It Shows
Name	Student name
Math	Marks in subject 1
Science	Marks in subject 2
English	Marks in subject 3
Total	Sum of all subjects
%	Average marks (Total ÷ Subjects)
Result	IF % >= 40 → Pass, else → Fail

Step-by-Step:

Step 1: Create Headings

A1: Name, B1: Math, C1: Science, D1: English, E1: Total, F1: %, G1: Result

Step 2: Enter Data + Formulas

A	B	C	D	E	F	G
Amit	78	82	75	=SUM(B2:D2)	=E2/3	=IF(F2>=40,"Pass","Fail")
Neha	56	62	71	=SUM(B3:D3)	=E3/3	=IF(F3>=40,"Pass","Fail")
Ravi	35	30	40	=SUM(B4:D4)	=E4/3	=IF(F4>=40,"Pass","Fail")

Bonus: Make It Visually Clear

- **Use conditional formatting:**
- **Highlight "Fail" in red**
- **Highlight marks < 35 in orange**

Extra Challenge:

- **Add a column for Grade (A/B/C/D/F)**
- **Use more IF or IFS functions to assign grades:**

excel

CopyEdit

=IFS(F2>=90,"A", F2>=75,"B", F2>=60,"C", F2>=40,"D", TRUE,"F")

3.3. Project 3: Monthly Budget Tracker

Objective: Compare Planned vs. Actual expenses. Know where you overspent or saved.

Table Structure:

A	B	C	D
Category	Planned	Actual	Difference

Step-by-Step:

Step 1: Enter Categories

A	B	C	D
Income	40000	40000	=B2–C2
Rent	10000	10000	=B3–C3
Groceries	5000	5500	=B4–C4
Transport	2000	1800	=B5–C5

Savings	8000	7000	=B6–C6
Entertainment	3000	4000	=B7–C7
Totals	=SUM(B2:B7)	=SUM(C2:C7)	=B8–C8

Add a Chart (Optional)

- Select A2:C7
- Go to Insert → Column Chart
- This shows visual comparison of your planned vs. actual spending

Analysis Tip:

- Positive values in Difference = You saved
- Negative values = You overspent
- Highlight negative numbers in red using conditional formatting

3.4. Practice Time 3 Do It Yourself!

Here are some tasks you can try on your own:

Task #	Practice Activity
1	Create a salary sheet for 5 employees
2	Add bonus and overtime columns
3	Create a marksheet with 4 subjects and 5 students
4	Add a grade system using IF or IFS function
5	Create a 6-category budget sheet
6	Use charts to compare planned vs actual spending
7	Format headers, use currency/percentage formats

3.5. Summary Table

Feature/Project	Key Function/Formulas Used	What It Teaches You
Salary Sheet	=B2+C2+D2–E2	Basic formulas and calculations
Marksheet	SUM(), /, IF()	Working with logic and averages
Budget Tracker	–, SUM(), conditional formatting	Expense planning and visual data comparison

Chapter 4: Basic Excel Functions

Introduction: Excel is powerful because of its built-in functions that help you calculate, analyze, and manage data quickly. In this chapter, you will learn all the essential basic functions used in everyday Excel work — with simple explanations and real-life examples.

4.1. What Are Functions in Excel?

A function is a pre-defined formula that performs a specific task. Functions always start with an equal sign = followed by the function name and brackets with the input (called arguments).
Example: excel
CopyEdit
=SUM(A1:A3)
This adds the values in cells A1, A2, and A3.

4.2. Common Categories of Basic Functions

We9ll divide the functions into four main types:

- **Math & Calculation Functions**
- **Counting Functions**
- **Date & Time Functions**
- **Text Functions**

4.3. Math & Calculation Functions

These are used for adding, averaging, rounding, and more.

Function	What It Does	Example	Result
SUM()	Adds values	=SUM(A1:A5)	Total value
AVERAGE()	Finds average	=AVERAGE(B1:B3)	Mean value
MAX()	Finds the highest value	=MAX(C1:C5)	Largest number
MIN()	Finds the lowest value	=MIN(D1:D5)	Smallest number
ROUND()	Rounds to specific decimal places	=ROUND(3.14159, 2)	3.14
INT()	Rounds down to nearest whole number	=INT(9.8)	9

When to use: Totals, marksheets, budgets, bills, salary sheets, etc.

4.4. Counting Functions

These help you count different types of cells.

Function	What It Does	Example	Result
COUNT()	Counts only numbers	=COUNT(A1:A5)	e.g. 3
COUNTA()	Counts all non-empty cells	=COUNTA(A1:A5)	e.g. 5
COUNTBLANK()	Counts how many empty cells	=COUNTBLANK(A1:A5)	e.g. 2

When to use: Attendance sheets, survey forms, data quality checks.

4.5. Date and Time Functions

These functions work with today's date and time.

Function	What It Does	Example	Output
TODAY()	Shows the current date	=TODAY()	31/05/2025
NOW()	Shows current date and time	=NOW()	31/05/2025 10:42 AM

When to use: Invoices, reports, daily logs, tracking deadlines.

4.6. Text Functions

These are useful for joining or formatting text.

Function	What It Does	Example	Result
CONCAT()	Joins multiple cells/text into one	=CONCAT(A1, " ", B1)	John Smith
TEXT()	Formats a number or date as text	=TEXT(TODAY(), "dd-mmm-yyyy")	31-May-2025

When to use: Full names, receipts, date formatting, report headers.

4.7. Real-Life Example: Marksheet

A	B	C	D	E
Name	Math	Science	English	Total Marks
Rahul	80	90	85	=SUM(B2:D2)
Priya	75	88	92	=SUM(B3:D3)
Amit	60	65	70	=SUM(B4:D4)

You can also add:

- **Average: =AVERAGE(B2:D2)**
- **Highest mark: =MAX(B2:D2)**
- **Grade comment using CONCAT: =CONCAT(A2, " - Passed")**

4.8. Practice Exercise

Try these steps in Excel:

- **Type numbers in A1 to A5.**
- **Use =SUM(A1:A5) in A6 to add them.**
- **In B1 to B5, type any words. Use =COUNTA(B1:B5) to count them.**
- **Leave some cells empty and try =COUNTBLANK() on them.**
- **Try =TODAY() and =NOW() in different cells.**
- **Type your first and last name in two cells and join with =CONCAT(A1, " ", B1).**

4.9. Summary Table

Function	Purpose	Example	Result
SUM()	Add numbers	=SUM(A1:A5)	Total
AVERAGE()	Average of values	=AVERAGE(A1:A5)	Mean
MAX()	Largest number	=MAX(A1:A5)	Highest
MIN()	Smallest number	=MIN(A1:A5)	Lowest
ROUND()	Round number to decimal places	=ROUND(3.14, 1)	3.1
INT()	Remove decimal part	=INT(4.9)	4
COUNT()	Count numbers	=COUNT(A1:A5)	Count of numbers
COUNTA()	Count non-empty cells	=COUNTA(A1:A5)	Count of filled cells
COUNTBLANK()	Count empty cells	=COUNTBLANK(A1:A5)	Count of blanks
TODAY()	Show current date	=TODAY()	Today's date
NOW()	Show date and time	=NOW()	Date + time
CONCAT()	Join text	=CONCAT(A1, " ", B1)	Combined string
TEXT()	Format value as text	=TEXT(TODAY(), "dd-mmyyyy")	31-05-2025

Chapter 5: Cell Formatting and Layouts

Introduction: Formatting is how you make your Excel sheet look clean, organized, and easy to understand. Good formatting helps highlight important information and makes your data readable. In this chapter, you will learn how to change the appearance of cells, adjust layout, and present your data professionally.

5.1. What Is Cell Formatting?

Cell formatting refers to how the content inside a cell looks. This includes:

- **Font style and size**
- **Colors (text and background)**
- **Borders**
- **Number formats (e.g., currency, date, percentage)**
- **Alignment (center, left, right)**

Example:

You can make the header row bold with a yellow background to stand out:
markdown
CopyEdit
Name | Age | City
Rahul | 25 | Delhi

5.2. Basic Text Formatting

You can format text like you would in Word. Select the cell(s) → Use the Home tab.

Option	What It Does	How to Use
Bold (B)	Makes text bold	Click B or press Ctrl + B
Italic (I)	Slants text	Click I or press Ctrl + I
Underline (U)	Adds underline	Click U or press Ctrl + U
Font Size	Changes text size	Select size from the dropdown (e.g., 12, 14)
Font Color	Changes text color	Click the A icon with color under it
Fill Color	Adds background color to cell	Click the paint bucket icon

Pro Tip: Use bold for headers, and light background shades to group data visually.

5.3. Adjusting Cell Size

Change Column Width:

- **Hover on the line between column letters (e.g., A and B).**

- Drag to adjust.
- Double-click to AutoFit (fit content automatically).

Change Row Height:

- Do the same on row numbers (e.g., between 1 and 2).

5.4. Cell Alignment and Wrapping

Option	What It Does	How to Use
Align Left/Center/Right	Changes horizontal alignment	Home tab → Alignment section
Top/Middle/Bottom Align	Changes vertical alignment	Useful for taller rows
Wrap Text	Shows all content in one cell (multiline)	Select cell → Click Wrap Text
Merge & Center	Combines multiple cells into one big cell	Select multiple cells → Click Merge & Center

Use Case: For creating centered titles across a table.

5.5. Number Formatting

Excel lets you format numbers for different uses:

Format Type	What It Does	Example Input	Example Display
General	Default format (no styling)	1234.5	1234.5
Number	Adds commas and decimals	1234.5	1,234.50
Currency	Adds ₹, $, etc.	5000	₹5,000.00
Percentage	Converts to %	0.25	25%
Date	Converts to date format	31/05/2025	31-May-2025
Time	Converts to time	10:15 AM	10:15 AM

How to Apply:

- Select the cells.
- Go to Home tab → Number group.
- Choose the format from the dropdown.

5.6. Adding Borders

Borders make your data look like a table.

To add borders:

- Select the range of cells.
- Click the Borders icon (square box).
- Choose from options like:
  - All Borders
  - Top Border
  - Bottom Border
  - Thick Outer Border

Use Case: Create visible rows/columns in reports or forms

5.7. Formatting Tables

To quickly make a nice-looking table:

- Select your data.
- Go to the Insert tab → Click Table.
- Choose a Table Style (with colors and filters).

Benefits:

- Built-in filter options
- Alternating row colors
- Easy sorting and filtering

5.8. Page Layout Formatting (for Printing)

Go to the Page Layout tab for print settings.

Feature	What It Does
Margins	Adjust space at the edges of the page
Orientation	Set to Portrait or Landscape
Size	Set paper size (A4, Letter, etc.)
Print Area	Choose which part of the sheet to print
Gridlines	Show/hide lines between cells when printing
Headers/Footers	Add title, page numbers, date to printed sheet

5.9. Practice Exercise

Try these steps:

- Create a table with Name, Age, Salary.
- Make the header row bold, centered, and add a background color.
- Format Salary column as currency.
- Adjust column width to fit content.
- Add borders to the table.

- **Merge and center the title above the table.**
- **Use Wrap Text on any long content.**
- **Go to Page Layout tab → Set orientation to Landscape.**
- **Try printing preview (Ctrl + P) and check how it looks.**

5.10. Summary Table

Task	Tool/Option	Location
Make text bold	Bold (B)	Home tab
Add cell color	Fill Color	Home tab
Align text	Align Left / Center	Home tab
Wrap long content	Wrap Text	Home tab
Format as currency	Currency from Number Format	Home tab
Add cell borders	Borders	Home tab
Insert a table	Insert Table	Insert tab
Adjust print layout	Page Layout options	Page Layout tab

Chapter 6: Pivot Tables, Pivot Charts, and Logical Functions (IF, AND, OR)

Introduction: Pivot Tables, Pivot Charts, and logical functions like IF, AND, and OR are powerful tools in Excel or Google Sheets used for data analysis and decision-making. A Pivot Table helps summarize large data sets by grouping, counting, or totaling values—for example, showing total sales by region or product. Pivot Charts are visual representations of Pivot Tables, allowing you to easily compare data trends using graphs like bar charts or pie charts. Logical functions make decisions based on conditions: the IF function returns one value if a condition is true and another if false; AND checks if all conditions are true; OR checks if any one condition is true. Together, these tools help users quickly understand data, automate tasks, and make informed decisions.

6.1. What is Data Analysis in Excel?

Before diving into tools like Pivot Tables and Logical Functions, it9s important to understand why we use them. Imagine you9re a teacher with a class of 100 students or a sales manager tracking thousands of orders. Manually reviewing every record would be impossible. That9s where tools like Pivot Tables and logical functions come in handy — they help Excel summarize, analyze, and make decisions automatically.

6.2. Pivot Tables 4 Summarize Big Data Instantly

What is a Pivot Table?

A Pivot Table is a summary tool that rearranges (or <pivots=) your data to help you analyze it without changing the original dataset. It can calculate totals, averages, counts, and more.

Why Use Pivot Tables?

- **To analyze large data sets in seconds**
- **To answer specific questions (e.g., How many products sold in each region?)**
- **To automatically group and summarize data**

Real-Life Example:

You have the following sales data:

Salesperson	Region	Product	Sales
Ankit	North	Pen	500
Rina	South	Notebook	700
Ankit	North	Pencil	300
Rina	South	Pen	200
Amit	East	Notebook	600

You want to know total sales by Region and by Salesperson.

How to Create a Pivot Table

- Select your data (including headers).
- Go to the Insert tab → Click PivotTable.
- Choose where to place the Pivot Table: New Worksheet or Existing Worksheet.
- In the PivotTable Fields pane:
  - Drag Region to the Rows box.
  - Drag Sales to the Values box.
  -

Excel will automatically add up the sales and show totals for each region.

Things You Can Do With Pivot Tables

- Group dates into months/quarters
- Filter specific values (e.g., show only <North= region)
- Sort values (e.g., largest to smallest)
- Add multiple value types: Sum, Average, Count, Max, Min

6.3. Pivot Charts 4 Visualize Pivot Table Data

A Pivot Chart is linked to your Pivot Table and updates automatically when you change the Pivot Table.

How to Insert a Pivot Chart

- Click anywhere inside your Pivot Table.
- Go to Insert → Choose a chart type (e.g., Column, Pie, Bar).
- A chart will appear — it updates whenever you change the Pivot Table.

Example:

If your Pivot Table shows sales by region, the Pivot Chart can visually compare regions.

Region	Total Sales
North	800
South	900
East	600

A bar chart will show which region is performing best.

6.4. Logical Functions: IF, AND, OR 4 Make Smart Decisions

Logical functions help Excel answer YES/NO questions and act accordingly. These are especially useful for evaluations, grading, bonus decisions, etc.

6.5. IF Function 4 Choose Between Two Outcomes

Syntax:

excel

```
CopyEdit
=IF(condition, value_if_true, value_if_false)
```

Example 1: Pass or Fail

Name	Marks	Result Formula
Neha	75	=IF(B2>=40, "Pass", "Fail")
Raj	32	=IF(B3>=40, "Pass", "Fail")

If marks are 40 or more → "Pass"
Else → "Fail"

6.6. AND Function 4 All Conditions Must Be True

Syntax:

excel

```
CopyEdit
=AND(condition1, condition2, ...)
```

Example 2: Student must pass both subjects

Name	Math	Science	Passed All? Formula
Anu	50	45	=AND(B2>=40, C2>=40) → TRUE
Ravi	35	55	=AND(B3>=40, C3>=40) → FALSE

Result is TRUE only if both are 40 or more.

6.7. OR Function 4 At Least One Condition Should Be True

Syntax:

excel

```
CopyEdit
=OR(condition1, condition2, ...)
```

Example 3: Passed Any One Subject

Name	Math	Science	Passed Any? Formula

Meena	38	42	=OR(B2>=40, C2>=40) → TRUE
Arya	35	30	=OR(B3>=40, C3>=40) → FALSE

Result is TRUE if at least one subject is passed.

6.8 . Combining IF with AND / OR

You can nest these functions for more powerful conditions.

Example 4: Bonus Eligibility (Sales > 10000 and Attendance > 90%)

Name	Sales	Attendance	Bonus Formula
Aman	12000	95%	=IF(AND(B2>10000, C2>90%), "Yes", "No")
Seema	9500	91%	=IF(AND(B3>10000, C3>90%), "Yes", "No")

Example 5: Extra Class If Student Scores < 40 in Any Subject

Name	Math	English	Help Needed Formula
Ravi	39	42	=IF(OR(B2<40, C2<40), "Yes", "No") → Yes
Nita	45	44	=IF(OR(B3<40, C3<40), "Yes", "No") → No

6.9. Practice Activity 3 Try This on Your Own

Create this table:

Name	English	Math	Science	Result
Aman	45	38	49	?
Tara	41	44	30	?

Tasks:

- **Mark "Pass" if all subjects >= 40 using:**

excel

CopyEdit

=IF(AND(B2>=40, C2>=40, D2>=40), "Pass", "Fail")

- **If any subject < 35, write "Needs Attention":**

excel

CopyEdit

=IF(OR(B2<35, C2<35, D2<35), "Yes", "No")

6.10. Summary Table

Function	Description	When to Use
IF	Makes decisions between two outcomes	"Pass" or "Fail", "Yes" or "No"
AND	All conditions must be true	Check if student passed all subjects
OR	At least one condition is true	Check if student passed any subject
IF + AND	Decision based on multiple criteria	Bonus if sales and attendance are high
IF + OR	Action if any of multiple conditions	Help if any subject is below pass mark

Chapter 7: Table Formatting and Advanced Tools

Introduction: Table formatting and advanced tools in Excel or Google Sheets enhance the clarity, organization, and functionality of data. Table formatting allows users to apply consistent styles, add filters, and enable automatic updates when data changes, making spreadsheets easier to read and work with. Advanced tools like data validation, conditional formatting, slicers, flash fill, and goal seek help users analyze data more efficiently, reduce errors, and automate tasks. These features are especially useful for managing large data.

7.1. What is a Table in Excel?

An Excel Table is a structured range that comes with built-in tools for easy formatting, sorting, filtering, and data analysis. When you turn your data into a table, Excel treats it as a smart object.

Key Features of Excel Tables:

- **Built-in sorting and filtering**
- **Auto-formatting with themes**
- **Automatically expands when you add rows/columns**
- **Easy-to-read headers and total row**
- **Supports structured references in formulas**

7.2. How to Create a Table

Steps:

- **Select a range of data, including headers.**
- **Go to the Insert tab → Click Table.**
- **Check the box for My table has headers.**
- **Click OK.**

Now your data is formatted as a Table.

7.3. Table Design Tab – Styling Your Table

Once a table is created, Excel shows a Table Design tab (or Design tab in older versions). It includes:

Feature	Description
Table Name	Rename your table (e.g., SalesTable)
Table Styles	Choose color themes (banded rows/columns)
Header Row	Show or hide column headers
Total Row	Adds a row for sum, average, count, etc.

First Column/Last Column	Bold or highlight first/last column
Banded Rows/Columns	Alternating shading to improve readability

7.4. Converting Table Back to Normal Range

If you no longer want a table:

Steps:

- Click inside the table.
- Go to Table Design tab. 3. Click Convert to Range.

Excel will ask: <Do you want to convert the table to a normal range?= → Click Yes.
Formatting remains, but it loses all smart table features.

7.5. Using Formulas in Tables

In tables, Excel uses column names instead of cell addresses. This is called structured referencing.

Example:

excel

CopyEdit

=[@Price] * [@Quantity]

This means: Multiply values in the <Price= and <Quantity= columns of the same row.

7.6. Sorting and Filtering Data in Tables

Each column in a table includes a dropdown arrow for sorting and filtering.

Sorting:

Alphabetical (A to Z / Z to A)

Numerical (smallest to largest / vice versa)

Custom Sort (e.g., by month name)

Filtering:

Check/uncheck values

- Filter by conditions (e.g., > 5000)
- Filter by color (if cells have colors) Example:

Filter a <Region= column to show only North.

7.7. Conditional Formatting 3 Highlighting Key Data

Helps you visualize patterns, trends, and outliers.

How to Apply:

- Select data.
- Go to Home → Conditional Formatting.
- Choose a rule (e.g., Highlight Cell Rules → Greater Than).

Options:

- Highlight Cells Rules (e.g., > 50)

- **Top/Bottom Rules (Top 10%, Bottom 5 items)**
- **Data Bars**
- **Color Scales**
- **Icon Sets**

Real Use Case:
Highlight marks below 40 with red fill (Fail).

7.8. Data Validation 3 Control What Can Be Entered

Ensures users enter valid data in a cell.
Steps:

- **Select a cell or range.**
- **Go to Data tab → Click Data Validation.**
- **Choose:**
- **Whole number / decimal / date / time**
- **List (drop-down menu) o Custom formula**

Example:
Create a drop-down of valid cities (Delhi, Mumbai, Kolkata).

7.9. Remove Duplicates 3 Clean Up Your Data

Removes repeated entries from your table.

Steps:

- **Click anywhere in the table.**
- **Go to Data tab → Click Remove Duplicates.**
- **Select columns to check for duplicates.**
- **Click OK.**

Useful when importing raw data or survey results.

7.10. Total Row 3 Add Automatic Summaries

Adds a row at the bottom to show totals, averages, counts, etc.
Steps:

- **Click inside the table.**
- **Go to Table Design tab.**
- **Check Total Row.**
- **Click any cell in the Total Row → Choose function (SUM, AVERAGE, etc.).**

7.11. Insert Slicers for Filtering

Slicers are visual filters with buttons.
How to Add:

- **Click inside the table.**
- **Go to Table Design → Click Insert Slicer.**

- Choose a field (e.g., <Region=).
- Use slicer buttons to filter data visually.

Useful for dashboards!

7.12. Advanced Table Features

Feature	Description
Calculated Columns	Automatically fill formula down entire column
Auto-Expand Table	Table grows automatically when you add data
Structured References	Clearer formulas using column names
Named Ranges	Tables get names for easier reference in formulas

7.13. Practice Activity 3 Step-by-Step

- Create this dataset:

Name	Department	Salary	Bonus	Total
Raj	Sales	25000	2000	
Priya	HR	30000	3000	
Mohan	IT	28000	2500	
Simran	Sales	27000	1800	

- Convert it into a Table.
- Rename table to EmployeeData.
- In the <Total= column, use formula:
- =[@Salary] + [@Bonus]
- Format <Salary=, <Bonus=, and <Total= as Currency.
- Apply Conditional Formatting:
- Green for Total > ₹30,000
- Insert a Slicer for Department.
- Use Total Row to calculate average salary.

7.14. Summary Table

Tool	What It Does	Why It9s Useful
Excel Table	Structured format with built-in features	Easier data analysis

Table Design Tab	Change name, style, add total row	Enhances presentation
Sorting & Filtering	Organize and search data	Focus on what matters
Conditional Formatting	Color-code cells based on values	Highlight trends or issues
Data Validation	Restrict input to valid options	Reduce errors
Remove Duplicates	Delete repeating data	Clean and accurate data
Slicers	Visual buttons to filter data	Fast filtering for reports
Structured References	Use column names in formulas	Easier to read and maintain formulas
Total Row	Adds automatic calculations at bottom	Summarize data instantly

7.15 Charts and Data Visualization

Charts make your data easier to understand by showing patterns, trends, and comparisons visually.

Why Use Charts?

- **To quickly summarize large amounts of data**
- **To compare values (e.g., sales of two products)**
- **To spot trends over time (e.g., profit growth)**
- **To communicate results clearly in reports or presentations**

7.15.1 Types of Charts in Excel

Chart Type	Best For	Example
Column Chart	Comparing values side by side	Sales of products by month
Bar Chart	Horizontal comparison	Population of countries
Line Chart	Showing trends over time	Monthly income growth
Pie Chart	Showing parts of a whole	Market share percentage
Area Chart	Cumulative trends	Website traffic over time
Combo Chart	Mixed chart types (e.g., bars + line)	Revenue + Target Line

7.15.2 How to Create a Chart

- Select your data (including headers).
- Go to the Insert tab.
- Choose the chart type from the Charts group.
- Excel will insert the chart directly onto the sheet.

7.15.3 Customizing a Chart

After inserting a chart, you can format it:

- Chart Title: Click to rename
- Legend: Show/hide or move to left, right, top, or bottom
- Data Labels: Show exact values on the chart
- Chart Style: Use pre-designed styles or change colors
- Axes: Format number style, font size, and labels
- Gridlines: Show or hide lines for better readability
- Chart Design and Format tabs appear when the chart is selected.

7.15.4 Changing Chart Types

To switch between chart types:

- Select the chart.
- Go to the Chart Design tab.
- Click Change Chart Type.
- Choose a new type and click OK.

7.15.5 Inserting Charts from Tables

Excel Tables work seamlessly with charts:

- If you add more data to the table, the chart updates automatically.
- Structured data leads to cleaner, more dynamic charts.

7.15.6 Recommended Charts

Excel can suggest the best chart for your data:

- Select the data.
- Go to Insert → Click Recommended Charts.
- Choose a suggested chart that suits your data pattern.

7.15.7 Sparklines 3 Mini Charts Inside Cells

Sparklines are tiny charts that fit inside a cell to show trends.
To Insert:

- Select a cell next to your data.
- Go to Insert → Choose Line, Column, or Win/Loss Sparkline.
- Select the data range.

- **Click OK.**

Great for showing trends in:

- **Stock prices**
- **Sales patterns**
- **Attendance over weeks**

7.15.8 Practice Exercise 3 Create a Chart

Dataset:

Month	Sales
Jan	10000
Feb	15000
Mar	12000
Apr	18000
May	16000

Tasks:

- **Create a Line Chart to show sales trend.**
- **Add a Chart Title: "Monthly Sales Overview".**
- **Add Data Labels to show values.**
- **Apply a style and color theme.**
- **Insert a Sparkline next to each month.**

7.15.9 Summary 3 Charts in Excel

Feature	Use
Column/Bar Charts	Compare data across items or categories
Line/Area Charts	Show trends over time
Pie Charts	Show parts of a whole
Combo Charts	Combine different chart types
Sparklines	Display trends inside a single cell
Chart Customization	Improve readability and presentation
Table-based Charts	Auto-update when table grows

Chapter 8: Lookup Functions 3 VLOOKUP, HLOOKUP, LOOKUP, IFERROR

Introduction: Excel Lookup functions are powerful tools that help you search for data, retrieve matching values, and handle errors smartly. Whether you are working with a few rows or a large database, these functions make data retrieval faster and more dynamic.

8.1 Introduction to Lookup Functions

Lookup functions are used to find values in a table based on another value. They are widely used in reporting, data analysis, dashboards, and automation.

Why Lookup Functions Are Important

- **Automate the process of searching for related data.**
- **Save time when working with large tables.**
- **Make Excel work dynamically with changing data.**

8.2 VLOOKUP (Vertical Lookup)

What is VLOOKUP?

VLOOKUP searches for a value in the first column of a table and returns a value in the same row from another column to the right.

Syntax:

=VLOOKUP(lookup_value, table_array, col_index_num, [range_lookup])

Arguments:

- **lookup_value: The value you want to search for.**
- **table_array: The range that contains the data.**
- **col_index_num: Column number (starting from 1) from which the value should be returned.**
- **range_lookup:**
- **TRUE = Approximate match**
- **FALSE = Exact match (Recommended)**

Example:

A	B	C
ID	Name	Salary
101	Alice	50000
102	Bob	55000
103	Charlie	60000

To find salary of ID 102:
=VLOOKUP(102, A2:C4, 3, FALSE)
Result: 55000

Tips:

- **The lookup column must be the first column in the table.**
- **VLOOKUP cannot look to the left of the lookup column.**

8.3 HLOOKUP (Horizontal Lookup)

What is HLOOKUP?

HLOOKUP searches for a value in the first row of a table and returns a value in the same column from another row below.

Syntax:

=HLOOKUP(lookup_value, table_array, row_index_num, [range_lookup])

Example:

Source	A	B	C
Item	Pen	Pencil	Eraser
Price	10	5	8

To find price of "Pencil":

=HLOOKUP("Pencil", A1:C2, 2, FALSE) Result: 5

SUM Lookup & Picture VLOOKUP

- **SUM Lookup in Excel**

To sum values based on one or more conditions using Excel formulas.

Key Functions

Function	Description	Syntax Example
SUMIF	Sums values with a single condition	=SUMIF(range, criteria, sum_range)
SUMIFS	Sums values with multiple conditions	=SUMIFS(sum_range, range1, crit1, range2, crit2)

Example 1: Using SUMIF Goal: Sum total sales for "Apple".

Product	Sales
Apple	100

Banana	200
Apple	150
Orange	300
Banana	100

Formula:
=SUMIF(A2:A6, "Apple", B2:B6)
Explanation:

- **Checks column A for the word <Apple=**
- **Adds matching values from column B**

Result:
100 + 150 = 250
Example 2: Using SUMIFS
Goal: Sum sales for "Apple" where sales are greater than 120.
Data Table:

Product	Sales	Region
Apple	100	North
Banana	200	South
Apple	150	North
Orange	300	West
Apple	130	South

Formula:
=SUMIFS(B2:B6, A2:A6, "Apple", B2:B6, ">120")
Explanation:
Sums values in column B
Only includes rows where: o Product = "Apple" o Sales > 120
Result:
150 + 130 = 280

2. Picture VLOOKUP in Excel
To display an image dynamically based on a selected value (like VLOOKUP, but for pictures). Setup Steps

Step	Action
1	Create a table with Product Names and inserted images
2	Name each image using the Name Box (e.g., name it "Apple")
3	Add a dropdown list using Data → Data Validation

4	Use INDEX + MATCH to find the image cell based on dropdown value
5	Use Paste Special → Linked Picture to show the image dynamically

Example:

Data Table

Product	Picture
Apple	[Apple image here]
Banana	[Banana image]
Orange	[Orange image]

- Dropdown Cell: E2 Linked Image Formula:
- =INDEX(B2:B4, MATCH(E2, A2:A4, 0))

Steps to Link Picture:

- Copy a cell with an image (e.g., B2)
- Right-click another cell → Paste Special → Linked Picture
- With the linked picture selected, enter the formula in the formula bar

Result:

When you change the dropdown in E2, the image changes automatically.

Summary Table

Feature	Goal	Method / Formula
SUMIF	Sum values with 1 condition	=SUMIF(range, criteria, sum_range)
SUMIFS	Sum values with multiple filters	=SUMIFS(sum_range, cond1_range, cond1, ...)
Picture VLOOKUP	Show picture based on selection	Insert images + INDEX + MATCH + Linked Picture

8.4 LOOKUP (Legacy Function)

What is LOOKUP?

LOOKUP finds a value in a single row or column and returns a value from the same position in another row or column.V V Syntax:

=LOOKUP(lookup_value, lookup_vector, result_vector)

Important Notes:

- The lookup_vector must be sorted in ascending order.

- **Limited flexibility compared to VLOOKUP and HLOOKUP.**

Example:

A	B
Apple	40
Banana	35
Mango	60

=LOOKUP("Banana", A2:A4, B2:B4)
Result: 35

8.5 IFERROR (Handling Errors Gracefully)

What is IFERROR?
IFERROR is used to catch and handle errors like #N/A, #DIV/0!, #VALUE!, etc.
Syntax:
=IFERROR(value, value_if_error)
Example:
=IFERROR(VLOOKUP(105, A2:C4, 3, FALSE), "Not Found")
Result: If ID 105 is not found, it returns "Not Found" instead of #N/A.

8.6 Advanced Lookup Techniques

- **Nested IFERROR with Multiple Lookup Tables**

Search in multiple sheets:

- **=IFERROR(VLOOKUP(A2, Sheet1!A2:B100, 2, FALSE), VLOOKUP(A2, Sheet2!A2:B100, 2, FALSE))**

1. **Dynamic Column Index Using MATCH**

=VLOOKUP("Bob", A1:D10, MATCH("Salary", A1:D1, 0), FALSE)
This allows you to automatically locate the column based on the header name.

2. **2D Lookup using INDEX and MATCH**

=INDEX(B2:D4, MATCH("Bob", A2:A4, 0), MATCH("Salary", B1:D1, 0))
Powerful method for both row and column-based matching.

8.7. Common Errors in Lookup Functions

Error	Cause	Solution
#N/A	Value not found	Use IFERROR
#REF!	Invalid column index	Check col_index_num

#VALUE!	Wrong data type	Ensure correct input
#NAME?	Misspelled function name	Fix spelling

8.8. Practice Tasks

- Use VLOOKUP to find employee name based on ID.
- Use HLOOKUP to find the price of a product.
- Create a formula using LOOKUP for sorted data.
- Combine IFERROR with VLOOKUP for error-free output.
- Use MATCH inside VLOOKUP for dynamic column selection.

8.9. Summary Table

Function	Type	Works On	Returns From	Sorted Data Required	Use Case
VLOOKUP	Vertical	Column (1st)	Right-side	No (with FALSE)	Search for data in columns
HLOOKUP	Horizontal	Row (1st)	Below	No (with FALSE)	Search across rows
LOOKUP	Legacy	Row or Column	Same position	Yes	Quick lookup in sorted lists
IFERROR	Error Trap	Any formula	Custom value	No	Handle and hide formula errors

Chapter 9: Date & Time Functions 3 TODAY, NOW, DATEDIF, EDATE

Introduction: Excel's Date & Time functions help you work with dates, calculate time intervals, manage deadlines, and automate calendars. These functions are essential in reports, HR, accounting, project tracking, and more.

9.1 Understanding Excel Dates and Times

- **Excel stores dates as serial numbers.**
  - **Example: 01-Jan-1900 = 1, 02-Jan-1900 = 2, and so on.**
- **Times are stored as decimal fractions of 1 day.**
  - **Example: 12:00 PM = 0.5, 6:00 AM = 0.25, 6:00 PM = 0.75**
- **This system allows Excel to perform arithmetic (addition/subtraction) on dates and times.**

9.2 TODAY Function 3 Get the Current Date

What it Does:

Returns the current system date (without the time).

Syntax:

=TODAY()

Features:

- **Automatically updates when you reopen or refresh the sheet.**
- **No arguments needed.**

Example:

=TODAY()

Result: 02-Jun-2025 (based on system clock)

Use Cases:

- **Track due dates.**
- **Calculate days left before a deadline.**
- **Find age or service duration.**

9.3 NOW Function 3 Get Current Date and Time

What it Does:

Returns the current date and time based on the system clock.

Syntax:

=NOW()

Features:

- **Updates whenever Excel recalculates.**
- **Returns values like: 02-Jun-2025 10:35 AM**

Use Cases:

- **Time stamps for reports.**
- **Real-time updates.**
- **Calculate time differences.**

9.4 DATEDIF 3 Calculate the Difference Between Two Dates

What it Does:

Calculates the difference between two dates in years, months, or days.

Syntax:

=DATEDIF(start_date, end_date, unit)

Units and Meaning:

- **"Y" – Full years**
- **"M" – Full months**
- **"D" – Total days**
- **"MD" – Day difference ignoring months/years**
- **"YM" – Month difference ignoring years**
- **"YD" – Day difference ignoring years**

Example:

=DATEDIF("01/01/2000", "01/01/2025", "Y") → 25

Use Cases:

- **Employee service calculation**
- **Date gaps in projects**
- **Person's age**

9.5 EDATE 3 Add or Subtract Months from a Date

What it Does:

Returns a date a specific number of months before or after a given start date.

Syntax:

=EDATE(start_date, months)

Example:

=EDATE("01-Jan-2025", 6) → 01-Jul-2025

Use Cases:

- **Loan payment schedule**
- **Monthly reminders**
- **Project deadline shifts**

9.6 EOMONTH 3 End of the Month Calculation

What it Does:

Returns the last day of the month, before or after a given number of months.

Syntax:

=EOMONTH(start_date, months)

Example:

=EOMONTH("15-Feb-2025", 1) → 31-Mar-2025

9.7 DATE and DATEVALUE Functions

DATE Function

- Creates a date using year, month, and day values.

=DATE(2025, 6, 2) → 02-Jun-2025

DATEVALUE Function

- Converts a text date into an Excel serial date.

=DATEVALUE("2-Jun-2025") → 45152

Use this when dates are imported as text.

9.8 TIME and TIMEVALUE Functions

TIME Function

- Creates a time from hour, minute, and second.

=TIME(10, 30, 0) → 10:30 AM

TIMEVALUE Function

- Converts text time into a time serial number.

=TIMEVALUE("10:30 AM") → 0.4375

9.9 WEEKDAY and WEEKNUM

WEEKDAY

- Returns the day number of the week.

=WEEKDAY("02-Jun-2025") → 2 (Monday if week starts on Sunday)

WEEKNUM

Returns the week number of the year.

WEEKNUM("02-Jun-2025") → 23

9.10 NETWORKDAYS and WORKDAY

NETWORKDAYS

Returns the number of working days (excluding weekends and optionally holidays).

=NETWORKDAYS("01-Jun-2025", "15-Jun-2025")

WORKDAY

- **Returns a future or past working date by adding workdays to a start date.**

=WORKDAY("01-Jun-2025", 10) → 15-Jun-2025

9.11 TEXT Function for Custom Date Formatting

Example:

=TEXT(TODAY(), "dddd, mmmm dd, yyyy")

→ Monday, June 02, 2025

Can format dates as:

- **"dd-mm-yyyy"**
- **"mmm yy"**
- **"dddd"**

9.12. Common Errors in Date Functions

Error	Reason	Fix
#VALUE!	Non-date text used incorrectly	Use DATEVALUE() or check format
#NUM!	Invalid arguments in EDATE/EOMONTH	Ensure proper date and months
Wrong Result	DATEDIF returns 0 or wrong	Ensure start_date < end_date

9.13. Practical Tasks

- **Use TODAY() to calculate how many days are left in the current year.**
- **Use DATEDIF to calculate someone's exact age in years and months.**
- **Add 90 days to today's date using =TODAY()+90**
- **Use EDATE to find renewal dates 6 months from now.**
- **Extract weekday name using =TEXT(TODAY(),"dddd")**
- **Find number of business days between two dates.**

- **Use NETWORKDAYS with a holiday list.**
- **Use WORKDAY to find delivery dates after a set of business days.**

9.14 Summary Table

Function	Purpose	Sample Formula	Result Example
TODAY()	Get current date	=TODAY()	02-Jun-2025
NOW()	Get current date and time	=NOW()	02-Jun-2025 10:45 AM
DATEDIF	Date difference in Y, M, D	=DATEDIF(A1, B1, "Y")	5
EDATE	Add/subtract months	=EDATE(A1, 6)	01-Jan-2026
EOMONTH	Last date of the month	=EOMONTH(A1, 0)	30-Jun-2025
DATE	Create a date	=DATE(2025, 6, 2)	02-Jun-2025
TIME	Create a time	=TIME(10, 30, 0)	10:30 AM
NETWORKDAYS	Count working days	=NETWORKDAYS(A1, B1)	22
WORKDAY	Get a future/past workday	=WORKDAY(A1, 5)	07-Jun-2025
TEXT	Format dates	=TEXT(A1, "dddd, ddmmm")	Monday, 02-Jun

Chapter 10: Financial Functions 3 PMT, FV, NPV

Introduction: Excel's Financial Functions are powerful tools for calculating loan payments, future values, and investment performance. These are used in personal finance, accounting, budgeting, and business planning.

10.1 PMT Function 3 Calculate Loan Payment

What it Does:

Calculates the payment amount for a loan based on constant payments and a constant interest rate.

Syntax:

=PMT (rate, nper, pv, [fv], [type])

Arguments:

- **rate: Interest rate per period**
- **nper: Total number of payments**
- **pv: Present value or loan amount**
- **fv*(optional)*: Future value (default is 0)**
- **type*(optional)*: When payments are due: 0 = end of period, 1 = beginning**

Example:

=PMT (5%/12, 60, -20000)

Monthly payment for a 5-year loan of $20,000 at 5% annual interest.

Use Cases:

- **Car loans**
- **Home loans**
- **Business loans**

10.2 FV Function 3 Calculate Future Value

What it does:

Returns the future value of an investment based on periodic payments and interest rate.

Syntax:

=FV(rate, nper, pmt, [pv], [type])

Example:

=FV(6%/12, 120, -200)

Saving $200 monthly for 10 years at 6% annual return.

Use Cases:

- **Retirement planning**
- **Education fund growth**
- **Investment projections**

10.3 NPV Function – Net Present Value

What it Does:

Calculates the net present value of an investment based on future cash flows and a discount rate.

Syntax:

=NPV(rate, value1, [value2], ...)

Example:

=NPV (10%, 1000, 2000, 3000)

Calculates present value of $1000, $2000, and $3000 received in the future at 10% discount rate.

Notes:

- **Assumes cash flows are at the end of each period.**
- **Does not include initial investment; subtract it separately.**

Use Cases:

- **Project evaluation**
- **Investment comparisons**
- **Business expansion decisions**

10.4. Additional Financial Functions

RATE – Calculate Interest Rate

=RATE(nper, pmt, pv)

Finds interest rate per period.

NPER – Number of Periods

=NPER(rate, pmt, pv)

Calculates how long it takes to repay a loan or reach an investment goal.

PV – Present Value

=PV(rate, nper, pmt)

Returns the current value of future payments.

10.5. Common Errors in Financial Functions

Error	Reason	Fix
#NUM!	Invalid rate or nper	Check inputs

#VALUE!	Wrong data type or missing value	Use numeric values
Negative result	Sign mismatch (PV or PMT)	Use negative sign for cash outflows

10.6. Practical Exercises

- **Use PMT to calculate the EMI for a personal loan of $50,000 at 7% annual rate for 5 years.**
- **Use FV to estimate savings if you invest $150 per month for 15 years at 8% annual return.**
- **Use NPV to compare two investment options with different cash flows and discount rates.**
- **Calculate how long it will take to repay a loan of $100,000 with monthly payments of $1200 at 5% using NPER.**
- **Use RATE to determine the interest rate on a loan based on known terms.**

10.7 Summary Table

Function	Purpose	Sample Formula	Description
PMT	Loan payment	=PMT(5%/12, 60, -20000)	Monthly loan installment
FV	Future investment	=FV(6%/12, 120, -200)	Value of monthly savings in future
NPV	Net present value	=NPV(10%, 1000, 2000, 3000)	Discounted value of future income
RATE	Interest rate	=RATE(60, -377.42, 20000)	Estimate interest on a loan
NPER	Number of periods	=NPER(5%/12, -377.42, 20000)	Time to repay a loan
PV	Present value	=PV(5%/12, 60, -377.42)	Amount needed to meet future goal

Chapter 11: Data Validation 3 Dropdowns, Input Messages, Custom Rules

Introduction: Data Validation is a powerful feature in Excel that helps you control the type of data or values that users can enter into a cell. It enhances data integrity, prevents input errors, and allows for the creation of professional, user-friendly templates.

11.1 What is Data Validation?

Data Validation restricts user input according to specified rules or criteria. You can:

- **Create dropdown lists**
- **Allow only numbers, dates, or text**
- **Show custom error messages**
- **Guide users with input messages**

11.2 How to Apply Data Validation

Steps:

- **Select the cell(s).**
- **Go to the Data tab.**
- **Click Data Validation>Data Validation.**
- **Under the Settings tab, choose a validation type.**
- **Define the criteria.**
- **Use Input Message and Error Alert tabs for guidance and feedback.**

11.3. Validation Types and Their Uses

Type	Description	Example
Whole Number	Restrict to integer values	Allow only values between 1 and 100
Decimal	Allow decimal values	Between 0.0 and 1.0
List	Create a dropdown of predefined items	Apple, Banana, Orange
Date	Limit entry to specific date range	Between 01-Jan-2025 and 31-Dec-2025
Time	Limit to specific time range	Between 09:00 and 17:00
Text Length	Restrict number of characters	Max 10 characters
Custom	Use a formula for advanced logic	=ISNUMBER(A1) or =A1>B1

11.3. Creating Dropdown Lists (List Type)

From Static List:

- Select cell(s).
- Choose List under validation type.
- Enter items separated by commas: Apple, Banana, Orange

From Cell Range:

- Enter list in a range (e.g., A1:A3).
- In validation, set Source to: =$A$1:$A$3

Dynamic Named Range (Advanced):

Use formula-based named ranges for auto-updating dropdowns.

Example:

=OFFSET (Sheet1!$A$1,0,0,COUNTA(Sheet1!$A:$A),1)

11.5 Input Messages

Show guidance to users when a cell is selected.

Example: "Enter your 6-digit employee ID"

Steps:

Go to Input Message tab.

Enter a title and message

11.6 Error Alerts

- Display when users enter invalid data.
- You can customize:
- Style: Stop (prevents entry), Warning, Information
- Title and Message Example:
- Title: Invalid Entry
- Message: "Please select from the dropdown list."

11.7 Using Formulas in Custom Validation

Examples:

- Allow values only if another cell is greater:

=A1<B1

- Allow only weekdays:

=WEEKDAY (A1,2)<=5

- Allow only unique values:

=COUNTIF ($A$1:$A$100, A1)=1

11.8 Circle Invalid Data

- **Highlights cells that do not meet validation rules.**
- **Go to Data>Data Validation>Circle Invalid Data**
- **Use Clear Validation Circles to remove highlights.**

11.9 Remove Data Validation

- **Select the cells.**
- **Go to Data>Data Validation>Clear All**

11.10 Limitations and Notes

- **Doesn't prevent copy-paste of invalid values.**
- **Can be bypassed if not carefully implemented.**
- **Protect sheets to maintain validation integrity.**

11.11 Practical Exercises

- **Create a dropdown list of departments.**
- **Restrict a salary input cell to numbers > 20000.**
- **Allow only dates within the current year.**
- **Use custom formula to allow only values divisible by 5.**
- **Use input message to guide product ID entry.**
- **Use error alert to block invalid months.**

11.12 Summary Table

Feature	Purpose	Example
List	Dropdown list	Apple, Banana, Orange
Whole Number	Allow only integers	1 to 100
Decimal	Allow decimal values	0.0 to 1.0
Date	Restrict to valid date range	01-Jan to 31-Dec
Time	Restrict to valid time	09:00 AM to 05:00 PM
Text Length	Limit text characters	Max 10
Custom	Advanced rules using formula	=A1<B1
Input Message	Guide user input	Enter Product ID
Error Alert	Show message on invalid entry	"Select from the list only"
Circle Invalid	Highlight bad entries	Highlight red circles

Chapter 12: Formula Auditing and Named Ranges

Introduction: Formula Auditing and Named Ranges are powerful features in Excel that help you better understand, manage, and troubleshoot your spreadsheets, especially when working with complex formulas.

12.1 What is Formula Auditing?

Formula Auditing tools help you analyze formulas, trace relationships between cells, and identify errors in your Excel sheets.

Key Benefits:

- **Understand how formulas are connected.**
- **Debug errors or unexpected results.**
- **Visualize the source of calculations.**

12.2 Where to Find Formula Auditing Tools

- **Go to the Formulas tab on the Ribbon.**
- **Look for the Formula Auditing group.**

Tools in the Formula Auditing Group:

- **Trace Precedents: Shows arrows to cells used in a formula.**
- **Trace Dependents: Shows arrows to cells that depend on the current cell.**
- **Remove Arrows: Clears the precedent and dependent arrows.**
- **Show Formulas: Displays all formulas instead of values.**
- **Error Checking: Finds errors in formulas.**
- **Evaluate Formula: Breaks down and shows each step of a complex formula.**
- **Watch Window: Monitors values of specific cells from anywhere in the workbook.**

12.3 Formula Auditing Tools Explained

1. **Trace Precedents**
   - **Visualizes which cells affect the value of a selected cell.**
   - **Use Case: Understand what data is feeding into a formula.**
2. **Trace Dependents**
   - **Shows which cells are affected by the selected cell.**
   - **Use Case: Before deleting a cell, check if other formulas depend on it.**
3. **Remove Arrows**
   - **Clears the blue arrows drawn by tracing tools.**
4. **Show Formulas**
   - **Toggles between displaying cell formulas and formula results.**
   - **Shortcut: Ctrl + ~**
5. **Error Checking**

- Helps you step through cells with errors.
- Use Case: Review #DIV/0!, #REF!, #NAME?, and more.

6. Evaluate Formula
   - Breaks down complex formulas step-by-step.
   - Great for learning how Excel computes nested formulas.
7. Watch Window
   - Keeps an eye on critical cells, even across different sheets.
   - Use Case: Track key figures in large workbooks.

12.4. What Are Named Ranges?

- A Named Range is a meaningful name assigned to a cell or group of cells. Instead of using A1:A10, you can refer to that range as Sales Data.

Benefits:

- Makes formulas easier to read.
- Improves workbook navigation.
- Simplifies dropdowns and data validation.

Creating a Named Range:

- Select the cell(s).
- Go to Formulas → Define Name.
- Enter a meaningful name (e.g., TotalSales).
- Click OK.
- Or use the Name Box (left of the formula bar) to type a name and press Enter.

Rules for Named Ranges:

- No spaces (use underscores _ instead).
- Must start with a letter or underscore.
- Cannot be the same as a cell reference (e.g., A1).

12.5 Using Named Ranges in Formulas

Instead of writing:

```
=SUM(A1:A1
0)  You  can
write:
=SUM(SalesData)
```

Dynamic Named Ranges with Table:

- Convert a range into a Table (Ctrl + T).
- The Table column gets an automatic name.
- Use it in formulas like:
- =SUM(Table1[Revenue])

12.6 Managing Named Ranges

- **Go to Formulas → Name Manager.**
- **You can:**
- **Edit or delete existing names.**
- **See which cells a name refers to.**

12.7 Practical Use Cases

- **Use Trace Precedents to identify where a total value is coming from.**
- **Watch Window to monitor a profit cell across multiple sheets.**
- **Evaluate a formula to debug why you9re getting an error.**
- **Use named ranges like InterestRate to avoid hardcoding values in multiple places.**

12.8 Summary Table

No.	Feature	Purpose	Use Case
	Trace Precedents	Find source cells for a formula	Check which sales values feed a total cell
	Trace Dependents	See which formulas depend on a cell	Before deleting data
	Show Formulas	Display all formulas instead of results	For review or printing
	Error Checking	Find and fix formula errors	Handle #DIV/0!, #REF!, etc.
	Evaluate Formula	Step-by-step formula analysis	Debug complex calculations
	Watch Window	Monitor key values	Watch overall profit from multiple sheets
	Named Ranges	Replace cell references with names	=SUM(Expenses) instead of =SUM(B2:B10)
	Name Manager	View, edit, or delete names	Maintain clarity in long formulas

Chapter 13: Data Entry Best Practices and Forms

Introduction: Accurate and efficient data entry is essential for maintaining clean, usable
Excel workbooks. This chapter covers professional best practices for entering data, designing input systems, and using Excel Forms to streamline entry, minimize errors, and improve organization.

13.1 Importance of Clean Data Entry

- **Clean and structured data helps in better analysis, calculations, and reporting.**
- **Poorly entered data can cause errors in formulas, charts, pivot tables, and reporting.**

13.2 Data Entry Best Practices

1. Use Consistent Data Formats

- **Dates: Use one format (e.g., DD/MM/YYYY or MM/DD/YYYY).**
- **Numbers: Avoid adding symbols directly (like %, $); use formatting instead.**
- **Text: Standardize spelling, capitalization (e.g., "Completed" vs "completed").**

2. Avoid Merged Cells

- **Merged cells can cause problems with filtering, sorting, and formulas.**
- **Instead, use "Center Across Selection" for alignment without merging.**

3. Use Headers

- **Always start tables with a clear header row.**
- **Avoid blank rows and columns within data ranges.**

4. Keep One Data Type per Column

- **A column should contain only one kind of data (e.g., all dates, all numbers).**

5. No Blank Rows between Data

- **Avoid empty rows or columns that separate data blocks.**
- **Keeps data ranges contiguous for formulas and PivotTables.**

6. Use Tables (Ctrl + T)

- **Tables automatically apply formatting, filter controls, and expand with new data.**

13.3 Using Data Entry Forms in Excel

Excel offers a built-in data form for entering data into structured tables more easily.

What is a Data Form?

- **A pop-up dialog that allows users to enter data into rows without manually scrolling through the spreadsheet.**

- **Especially helpful when working with wide tables with many columns.**

Steps to Enable and Use a Data Form:

- **Select your data and press Ctrl + T to create a table.**
- **Add the Form button to your Quick Access Toolbar:**
- **File > Options > Quick Access Toolbar > Choose commands from: Commands Not in the Ribbon > Add "Form..."**
- **Click the Form button to open the entry form.**
- **Use it to add, delete, and navigate records.**

Advantages of Using Data Forms:

- **No need to scroll horizontally.**
- **Easier to focus on one record at a time.**
- **Reduces entry errors.**

13.4 Creating Custom Data Entry Interfaces (Advanced)

1. Use Excel Forms with VBA

- **Custom UserForms can be created using VBA for more flexible and user-friendly interfaces.**
- **Add input fields, dropdowns, checkboxes, and buttons.**
- **Code validation logic and button actions (e.g., save to sheet).**

2. Use Microsoft Forms (with Excel Online)

- **Create forms online and sync responses directly to Excel.**
- **Suitable for collecting data from multiple users.**

13.5 Designing Effective Forms (Principles)

- **Keep the form layout simple and intuitive.**
- **Group related fields together.**
- **Use data validation and dropdowns to control inputs.**
- **Use color coding to guide users.**
- **Lock cells that shouldn9t be edited.**

13.6 Enhancing Forms with Validation and Formatting

Combine forms with Data Validation to restrict incorrect inputs.
Use Conditional Formatting to alert when fields are blank or invalid.
Use named ranges to create dynamic dropdowns in the form.

13.7 Tips for Collaborative Data Entry

- **Use Excel Online for shared workbooks.**
- **Lock cells and protect sheets to prevent accidental edits.**

- **Use comments or notes to give instructions.**
- **Track changes or enable version history to monitor edits.**

13.8 Practice Tasks

- **Convert a contact list into a table.**
- **Add a form button and use it to enter 5 new entries.**
- **Apply data validation to allow only valid email formats.**
- **Create a custom form using VBA.**
- **Create a Microsoft Form that feeds into Excel Online.**

13.9 Summary Table

Topic	Purpose	Tools Used
Data Entry Form	Simplify row-by-row data entry	Form Button (Quick Access)
Data Validation	Ensure correct data types	Data Tab > Data Validation
Excel Tables	Structure and format data	Ctrl + T
Custom VBA Forms	Advanced entry interfaces	VBA Editor
Microsoft Forms	Online data collection	Microsoft Forms + Excel Online

Chapter 14: Consolidating Data from Multiple Sheets

Introduction: Consolidating data in Excel means combining data from different sheets or workbooks into one summary sheet. This is helpful when dealing with large datasets that are split across different departments, time periods, or categories.

14.1 What is Data Consolidation?

- **Data Consolidation helps you summarize and merge data from multiple sources into a single location.**
- **You can consolidate:**
- **Across multiple sheets in the same workbook.**
- **Across multiple workbooks.**
- **Using labels or positions.**

14.2 Types of Consolidation

1. By Position

- **Data is in the same location (same row/column structure) in each sheet.**
- **Excel merges corresponding cells.**

2. By Category (Label)

- **Data is arranged differently, but uses labels to identify categories.**
- **Excel matches and combines values by matching labels.**

14.3 Accessing the Consolidate Tool

Steps:

- **Open a new or existing summary sheet.**
- **Go to the Data tab.**
- **Click Consolidate in the Data Tools group.**

14.4 Using the Consolidate Dialog Box

Key Options:

- **Function: Choose the operation (Sum, Count, Average, Max, Min, etc.).**
- **Reference: Select the data range from each sheet.**
- **Click Add to include multiple references.**
- **Check Top row and Left column if using labels.**

14.5 Example 3 Consolidate Monthly Sales

Assume you have:

- **Sheet Jan: A2:B5 (Product, Sales)**
- **Sheet Feb: A2:B5**

- **Sheet Mar: A2:B5**

Create a new sheet named "Total Sales":

- **Go to Data > Consolidate.**
- **Function: Sum**
- **Add references from all three sheets.**
- **Check Top row and Left column if product names are in first column.**
- **Click OK.**

14.6 Linking to Source Data (Dynamic Consolidation)

- **Check Create links to source data in the Consolidate dialog.**
- **Excel creates an outline that stays connected to original data.**
- **Updating original sheets will reflect in the summary sheet.**

14.7 Consolidating from Multiple Workbooks

- **Open all workbooks.**
- **In the summary sheet, open the Consolidate tool.**
- **In Reference, navigate to other workbook sheets.**
- **Add each reference and consolidate.**

14.8 Using Formulas for Manual Consolidation

If the structure varies too much, use formulas like:

SUM with Sheet References:

=SUM(Jan!B2, Feb!B2, Mar!B2)

3D References:

=SUM(Jan:Mar!B2)

Adds B2 across all sheets between Jan and Mar.

14.9 Tips for Successful Consolidation

- **Keep consistent headings and structure where possible.**
- **Use named ranges for better clarity.**
- **Always double-check for blank cells or merged cells, which can cause errors.**
- **Use tables to manage structured data.**

14.10 Advanced Techniques

Power Query for Data Consolidation:

- **Import multiple sheets or workbooks.**
- **Transform and merge data using queries.**
- **More flexible than the standard Consolidate tool.**

VBA Macros for Automation:

- **Use macros to automate consolidation across many sheets.**

- **Helpful in large-scale reporting.**

14.11 Summary Table

Method	Best For	Tools Used
Consolidate by Position	Same structure in all sheets	Data > Consolidate
Consolidate by Category	Different structures, with labels	Data > Consolidate
Manual Formulas	More flexibility or unique layout	SUM, 3D References
Power Query	Dynamic, multiple files or transformations	Get & Transform Data
VBA Macros	Automating recurring tasks	Developer > Macros

Chapter 15: Advanced Filtering and Table Editing

Introduction: Working with large datasets in Excel can be overwhelming without efficient tools. Advanced Filtering and Table Editing help you quickly narrow down data, sort, extract, and modify records effectively.

15.1 Introduction to Filters

What is Filtering?

Filtering allows you to temporarily hide rows that don9t meet specific criteria.

Types of Filters

- **AutoFilter (basic)**
- **Advanced Filter (custom complex filters)**

15.2 Basic Filtering

Steps to Apply a Filter:

- **Select your dataset.**
- **Go to Data tab → Click Filter.**
- **Click the drop-down arrows in column headers.**
- **Choose criteria such as text match, number range, date range, etc.**

Options Available:

- **Sort A to Z / Z to A**
- **Number filters (greater than, less than, between)**
- **Text filters (begins with, ends with, contains)**
- **Date filters (this week, next month, year to date)**

15.3 Advanced Filter

Advanced Filter provides more control by using criteria ranges and copying filtered data to another location.

Setup:

- **List range: Original data.**
- **Criteria range: Cells with headers and criteria (e.g., Department = Sales).**
- **Copy to: Optional - copy the filtered results elsewhere.**

Steps:

- **Set up your criteria in a separate area.**
- **Go to Data tab → Click Advanced.**
- **Fill out the dialog box with list range and criteria.**
- **Choose to filter in place or copy to another location.**

Supports:

- **Complex AND / OR conditions**

- **Exact match or partial match**

15.4 Using Tables for Dynamic Filtering

Create a Table:

- **Select your data.**
- **Press Ctrl + T or go to Insert → Table.**

Benefits:

- **Automatic filtering in headers**
- **Structured references**
- **Easy to add calculated columns**
- **Automatic expansion when new rows are added**

15.5 Custom Views

Custom Views allow you to save different filter and layout settings.

Steps:

- **Go to View tab → Custom Views.**
- **Click Add, name your view.**
- **Save with filters, hidden columns, and more.**
- **Switch between saved views without reapplying filters.**

15.6 Table Editing Techniques

Features:

- **Resize tables using the drag handle.**
- **Add Total Row for automatic summaries.**
- **Insert Slicers for visual filtering.**
- **Use calculated columns with formulas.**

Common Edits:

- **Renaming table headers**
- **Formatting columns**
- **Using dropdowns for consistent entry**

15.7 Sort by Multiple Levels

Steps:

- **Go to Data → Sort.**
- **Add levels for sorting (e.g., Department A-Z, then Salary High to Low).**
- **Choose sorting criteria (text, number, color).**

15.8.Using Formulas with Filters

Some formulas ignore hidden rows (like SUBTOTAL, AGGREGATE).

Example:

=SUBTOTAL(9, B2:B100) → sums only visible cells

15.9.Slicers for Tables

Slicers allow for quick visual filtering, commonly used in tables and PivotTables.

Steps:

- **Click on a Table.**
- **Go to Table Design → Insert Slicer.**
- **Choose the column(s) to filter.**

Benefits:

- **Easy to use buttons**
- **Can select multiple items**
- **Great for dashboards**

15.10 Power Query for Advanced Data Manipulation

Power Query provides advanced data editing tools like merge, append, filter, split, pivot, and unpivot.

Benefits:

- **No formulas needed**
- **Automates repeat tasks Refreshable with updated data**

15.11 Practical Use Cases

- **Filter employees in HR by department and age > 30.**
- **Copy data of products under Rs. 5000 to a new location using Advanced Filter.**
- **Use Slicer to analyze sales data by region and salesperson.**
- **Sort customer list by Country, then Name.**
- **Use Power Query to combine and filter monthly sales sheets.**

15.12 Summary Table

Feature	Purpose	Key Tools/Steps
AutoFilter	Quick filters	Data → Filter
Advanced Filter	Complex criteria	Data → Advanced
Table Tools	Structured data management	Insert Table (Ctrl + T)
Custom Views	Save filter/sort setups	View → Custom Views
Slicers	Visual filters for tables	Table Design → Insert Slicer
Power Query	Advanced data editing	Data → Get & Transform

Chapter 16: Review Tools in Excel 3 Spell Check, Smart Lookup, Comments

Introduction: Excel provides a range of review tools to ensure your data is accurate, informative, and collaborative. These tools help detect spelling errors, research terms directly, and collaborate effectively using comments.

16.1 Spell Check

Spell check helps you find and correct spelling errors in text cells, comments, and textboxes.

How to Use Spell Check:

- **Go to the Review tab on the Ribbon.**
- **Click on Spelling (shortcut: F7).**
- **Excel will check the selected cell range or the entire sheet.**

Choose options:

- **Ignore Once: Skip the current suggestion.**
- **Ignore All: Skip all instances of the word.**
- **Add to Dictionary: Add custom words.**
- **Change: Replace with the suggested word.**
- **Change All: Replace all instances.**

Limitations:

- **Only checks spelling in text entries.**
- **Doesn't check for grammar or context.**

Best Practices:

- **Run spell check before finalizing a report.**
- **Avoid abbreviations unless common in context.**

16.2 Smart Lookup

Smart Lookup offers contextual research using Microsoft's online services to provide definitions, images, and web search results.

How to Use Smart Lookup:

- **Select a word or phrase in a cell.**
- **Go to Review → Smart Lookup. 3. A pane opens on the right with:**

Definitions

- **Wikipedia links**
- **Top web search results**

Use Cases:

- **Quickly define unfamiliar terms.**
- **Get background on business concepts.**
- **Look up people, products, or places.**

Requirements:

- **Internet connection.**
- **Office connected to your Microsoft account.**

16.3 Comments (Legacy) and Notes (Modern)

Excel has two types of commenting features: Comments (for conversations) and Notes (for annotations).

16.3.1 New Comments (Threaded Comments)

- **Ideal for collaboration.**
- **Allow multiple replies in conversation threads.**

How to Add Comments:

- **Right-click on a cell.**
- **Select New Comment.**
- **Type your message. Press Ctrl + Enter to save.**
- **Replies can be added by others.**

Managing Comments:

- **Edit: Click the comment bubble.**
- **Delete: Right-click and choose delete.**
- **Resolve: Mark a conversation thread as completed.**

16.3.2 Notes (Previously called Comments)

- **Used for static annotations.**
- **No threading or replies.**

How to Add a Note:

- **Right-click the cell.**
- **Choose New Note.**
- **Type your message.**

16.4 Show All Comments or Notes

- **Go to Review Tab → Show Comments or Notes.**
- **Opens a side pane listing all comments for easy access.**

16.5 Protect Sheet/Workbook (Review Security Tool)

Protect your workbook or worksheet to prevent unauthorized changes.

How to Protect:

- **Go to Review Tab.**
- **Choose Protect Sheet or Protect Workbook.**
- **Set a password (optional).**
- **Select what actions to allow (e.g., format cells, insert rows).**

Types:

- **Protect Sheet: Locks structure and formatting.**
- **Protect Workbook: Prevents structural changes (like adding/deleting sheets).**

16.6 Track Changes (Available in Shared Workbooks)

Allows tracking of who changed what and when.

Note: Track Changes is only available in legacy shared workbook mode.

Enabling:

- **Go to Review Tab → Share Workbook (Legacy).**
- **Check Track Changes while editing.**

16.7 Accessibility Checker

Ensures your workbook is usable by people with disabilities.

How to Use:

- **Go to Review Tab → Check Accessibility.**
- **Excel will list potential issues.**
- **Suggestions are provided for fixing them (e.g., missing alt text).**

16.8 Inspect Workbook

Check for hidden properties or personal information.

How to Use:

- **Go to File → Info → Check for Issues → Inspect Workbook.**
- **Choose types of content to scan.**
- **Remove or edit anything private before sharing.**

16.9 Summary Table

Tool	Purpose	Best Use Case
Spell Check	Detects misspelled words	Final proofing before reports
Smart Lookup	Provides online info/definitions	Quick research in-sheet

Comments	**Collaborate with feedback threads**	**Team-based reviewing**
Notes	**Add static notes**	**Personal reminders**
Protect Sheet	**Lock down data from edits**	**Protect formulas/tables**
Track Changes	**Show edit history**	**Legacy multi-user editing**
Accessibility	**Ensure usability for all**	**Inclusive document creation**
Inspect Workbook	**Remove hidden metadata**	**Safe document sharing**

Chapter 17: Final Formatting Touches and Shortcuts Recap

Introduction: This chapter wraps up your Excel learning by focusing on presentation polish and productivity boosts using formatting features and keyboard shortcuts.

17.1 Importance of Final Formatting

Before sharing or presenting a spreadsheet, it9s important to ensure it is:

- **Visually clear and easy to read**
- **Well-organized**
- **Professionally styled**

17.2 Basic Formatting Tools Recap

1. Fonts & Styles

- **Bold / Italics / Underline: Emphasize headings or data points**
- **Font color: Color code important data**
- **Font size and style: Enhance readability**

2. Cell Fill Color (Shading)

Highlight headers, totals, or key figures

3. Borders

Use borders to clearly define areas, such as tables or input sections

4. Number Formatting

- **Currency, Percentage, Comma Style**
- **Date and Time: Standardized formats for clarity**
- **Custom Formats: Tailored display (e.g., showing "Rs." before numbers)**

17.3 Alignment & Text Control

1. Horizontal and Vertical Alignment

Align text left, right, or center

2. Text Wrap

Use Wrap Text to show all content inside a cell

3. Merge Cells

Combine cells to create headings across columns

4. Indentation

Use for subcategories or hierarchical data

17.4 Conditional Formatting Recap

- **Highlights data dynamically based on rules**
- **Use to spot trends, errors, duplicates, etc.**
- **Apply Data Bars, Color Scales, Icon Sets**

Advanced Usage:

- Use formulas within conditional formatting for custom logic

17.5 Table Formatting

- Convert ranges to Excel Tables (Ctrl + T)
- Tables include features like:
- Filter buttons
- Alternate row shading
- Auto-expanding formulas

17.6 Page Layout for Printing

1. Print Titles
   - Repeat header rows on every printed page
2. Page Orientation & Scaling
   - Choose between Portrait / Landscape
   - Use "Fit Sheet on One Page" option
3. Header & Footer

Add date, file name, page number, company logo

4. Print Area

Define a specific area for printing using Page Layout → Print Area

17.7 Workbook Cleanup Tips

- Delete unused sheets or rows
- Remove temporary calculations or helper columns
- Unhide any hidden rows/columns if needed
- Clear unused named ranges
- Update tab names clearly

17.8 Keyboard Shortcuts Recap (Must-Know)

Navigation

- Ctrl + Arrow Keys: Jump to edge of data
- Ctrl + Home/End: Go to beginning or end of worksheet

Formatting

- Ctrl + 1: Format Cells dialog
- Alt + H + B: Add border
- Ctrl + Shift + $: Currency format
- Ctrl + Shift + %: Percentage format
- Alt + E + S + V: Paste Special → Values

Tables and Filtering

- Ctrl + T: Create table
- Alt + Down Arrow: Open filter dropdown

Editing

- **F2: Edit selected cell**
- **Ctrl + Z/Y: Undo / Redo**
- **Ctrl + D: Fill down**
- **Ctrl + R: Fill right**

Sheet and Workbook Control

- **Ctrl + Page Up/Down: Switch between sheets**
- **Ctrl + N: New workbook**
- **Ctrl + S: Save**
- **Ctrl + P: Print**

17.9 Custom Themes and Styles

Set a consistent look using:

- **Themes: Colors, fonts, effects**
- **Cell Styles: Heading, Good/Bad/Neutral, Note**

Found under Page Layout → Themes and Home → Cell Styles

17.10 Final Checks Before Sharing

- **Spell Check (F7)**
- **Remove Blank Rows/Columns**
- **Protect sensitive data (via cell locking or password protection)**
- **Use document properties (File → Info → Properties)**
- **Save as PDF for fixed layout sharing**

17.11 Summary Table

Area	Task	Shortcut / Tool
Font Formatting	Bold, Italic, Color	Ctrl + B, I, U
Cell Borders	Add borders	Alt + H + B
Conditional Format	Highlight rules, formulas	Home → Conditional Formatting
Tables	Structured tables	Ctrl + T
Print Setup	Page orientation, scaling	Page Layout tab
Final Check	Spell check, Protect data	F7, Review tab

With these finishing touches and productivity tips, your Excel sheets will not only work well but look professional and clean. This marks the end of your Excel textbook series. Congratulations on completing your learning journey!

Chapter 18: Introduction to SQL and Relational Databases

Introduction: SQL (Structured Query Language) and relational databases are essential tools for storing, managing, and retrieving structured data. A relational database organizes data into tables with rows and columns, where each table represents a specific type of information and is linked to others through relationships. SQL is the standard language used to interact with these databases, allowing users to insert, update, delete, and query data efficiently. It is widely used in applications across industries for tasks like reporting, data analysis, and backend development. Together, SQL and relational databases provide a powerful, flexible, and organized way to handle large volumes of data.

18.1 What is SQL?

Structured Query Language (SQL) is a standardized programming language specifically designed to manage and interact with relational databases. It is the universal language for databases and is used by software developers, data analysts, business intelligence professionals, and data scientists.

SQL performs the following core functions:

- **Data Querying: Retrieve data using SELECT statements.**
- **Data Manipulation: Add, update, or delete records with INSERT, UPDATE, and DELETE.**
- **Data Definition: Create or modify table structures using CREATE and ALTER.**
- **Data Control: Manage user access and permissions via GRANT and REVOKE.**

Example 18.1 – Simple SQL Query

```
SELECT FirstName, LastName
FROM Employees
WHERE Department = 'Sales';
```

This query retrieves the names of all employees working in the Sales department.

18.2 What are Relational Databases?

A relational database is a type of database that stores data in the form of tables composed of rows and columns. Each table represents an entity (like Students or Orders), and relationships between tables are formed using keys.

18.3 Key Concepts of Relational Databases

Concept	Description
Tables	Store data about specific subjects (e.g., Students, Orders).
Rows	Individual records in a table.
Columns	Attributes of the records (e.g., Name, Age).

Primary Key	A unique identifier for each row in a table.
Foreign Key	A field in one table that links to the Primary Key in another table.

Example 18.2 3 Table Structure

Students Table:

```
+------------+----------+-----+
| Student ID | Name     | Age |
+------------+----------+-----+
| 101        | Alice    | 20  |
| 102        | Bob      | 22  |
+------------+----------+-----+
```

Courses Table:

```
+----------+--------------+
| Course ID | Course Name  |
+----------+--------------+
| C001     | Mathematics  |
| C002     | Biology      |
+----------+--------------+
```

Enrollments Table:

```
+------------+----------+
| StudentID | CourseID |
+------------+----------+
| 101     | C001     |
| 102     | C002     |
+------------+----------+
```

The Enrollments table connects Students and Courses through foreign keys.

18.4 Advantages of Relational Databases

- **Data Integrity: Maintains accuracy using constraints like NOT NULL, UNIQUE, etc.**
- **Flexibility: Allows schema changes without major redesign.**
- **Scalability: Handles large datasets efficiently.**
- **Security: Grants role-based access to sensitive data.**

Example 18.3 3 Table with Constraints

```
CREATE    TABLE    Students    (

 StudentID INT PRIMARY KEY,

  Name VARCHAR(100) NOT NULL,
```

Age INT CHECK (Age >= 18)

18.5 Real-World Example: School Database

- **Imagine you are managing a school9s database:**
- **Students Table: Contains details about students**
- **Courses Table: Information about each course**
- **Enrollments Table: Records of which student is in which course**
- **This setup supports easy queries and reports.**

18.6 Why SQL is Important for Data Analysts

SQL is crucial for data analysts to:

- **Extract data with conditions and filters**
- **Aggregate and summarize datasets**
- **Join tables to build comprehensive reports**
- **Create dashboards in tools like Tableau, Power BI, or Excel**

Example 18.4 – Analyzing Sales Data

- **SELECT ProductName, SUM(Quantity) AS TotalSold**
- **FROM Sales**
- **GROUP BY ProductName**
- **ORDER BY TotalSold DESC;**

This helps a business identify its top-selling products.

18.7 Types of SQL Statements

Statement Type	Commands	Purpose
Data Query Language	SELECT	Retrieve data
Data Manipulation	INSERT, UPDATE, DELETE	Modify data
Data Definition	CREATE, ALTER, DROP	Define and change database structure
Data Control	GRANT, REVOKE	Manage access and permissions
Transaction Control	BEGIN, COMMIT, ROLLBACK	Handle data transactions securely

18.8 Timeline of SQL Evolution

Year	Milestone
1970	Edgar F. Codd proposes relational model
1974	SQL developed by IBM (SEQUEL)
1986	ANSI standard SQL introduced
2003	SQL:2003 includes XML and window funcs
Today	SQL used across MySQL, PostgreSQL, MS SQL Server, Oracle, SQLite, etc.

18.9 Graphic Suggestions for Visual Understanding

- **Entity-Relationship Diagram: Visualize tables and their relationships**
- **SQL Timeline Chart: Visual evolution of SQL from 1970s to today**
- **Annotated Query Diagram: Highlighting parts of a SQL query (SELECT, FROM, WHERE)**

18.10 Summary

In this chapter, you learned:

- **What SQL is and why it's important**
- **How relational databases work and how tables are structured**
- **Key database concepts such as primary keys and foreign keys**
- **How SQL supports real-world tasks in business and analytics**
- **Examples and syntax of common SQL operations**

SQL is a foundational tool for working with data, and mastering it empowers you to efficiently query, manage, and analyze databases.

Chapter 19: SQL Command Types 3 DDL, DML, DQL, DCL, TCL

19.1 Introduction

SQL (Structured Query Language) offers a variety of commands categorized into five major types, each serving a specific purpose in the database lifecycle:

- DDL: Defines the structure/schema of the database.
- DML: Deals with data manipulation.
- DQL: Fetches data from the database.
- DCL: Manages permissions and access control.
- TCL: Controls transactions (commit/rollback).

19.2 Data Definition Language (DDL)

DDL commands define and modify the database schema and structure (tables, indexes, constraints, etc.).

19.2.1 CREATE

Creates a new table, view, database, or other objects.

```
CREATE TABLE Students (
 StudentID INT PRIMARY KEY,
 Name VARCHAR (100),
 Age INT,
 Course VARCHAR (50)
```

19.2.2 ALTER

Modifies an existing table (add/remove columns, change data type).

```
ALTER TABLE Students
ADD Email VARCHAR (100);
```

19.2.3 DROP

Deletes the table or database permanently.

```
DROP TABLE Students;
```

This deletes structure and data.

19.2.4 TRUNCATE

Removes all records from a table, but keeps structure.

```
TRUNCATE TABLE Students;
```

cannot use WHERE clause; faster than DELETE; cannot be rolled back in some DBMS.

19.2.5 RENAME (Optional DDL Command)

Renames an object (table, column in some systems).

```
RENAME TABLE Students TO Learners;
```

19.3 Data Manipulation Language (DML)

DML commands deal with data stored in the tables.

19.3.1 INSERT

Adds new data.

```
INSERT INTO Students (StudentID, Name, Age, Course)
VALUES (1, 'Aman', 22, 'Python');
```

19.3.2 UPDATE

Modifies existing data.

```
UPDATE Students
SET Age = 23
WHERE StudentID = 1;
```

19.3.3 DELETE

Removes data based on condition.

```
DELETE FROM Students
WHERE StudentID = 1;
```

DELETE can be rolled back and uses WHERE clause.

19.3.4 MERGE (Optional in some RDBMS)

Combines INSERT, UPDATE, and DELETE in one go.

```
MERGE INTO TargetTable AS T
USING SourceTable AS S
ON T.ID = S.ID
WHEN MATCHED THEN
UPDATE SET T.Name = S.Name
WHEN NOT MATCHED THEN
INSERT (ID, Name) VALUES (S.ID, S.Name);
```

19.4 Data Query Language (DQL)

19.4.1 SELECT

It can be simple or highly complex with filtering, sorting, and aggregation.

Basic:

```
SELECT * FROM Students;
```

Specific Columns Used to retrieve data from tables.:

```
SELECT Name, Course FROM Students;
```

With WHERE Clause:

```
SELECT * FROM Students WHERE Age > 20;
```

With ORDER BY:

```
SELECT * FROM Students ORDER BY Name ASC;
```

With GROUP BY:

```
SELECT Course, COUNT(*) AS StudentCount
FROM Students
GROUP BY Course;
```

With HAVING:

```
SELECT Course, COUNT(*) AS StudentCount
FROM Students
GROUP BY Course

HAVING COUNT(*) > 2;
```

With DISTINCT:

```
SELECT DISTINCT Course FROM Students;
```

19.5 Data Control Language (DCL)

DCL commands manage user rights and access to database objects.

19.5.1 GRANT

Gives a user or role permission to perform certain actions.

```
GRANT SELECT, INSERT ON Students TO user1;
```

19.5.2 REVOKE

Takes back the permissions given using GRANT.

```
REVOKE INSERT ON Students FROM user1;
```

Permissions control data privacy and integrity.

19.6 Transaction Control Language (TCL)

TCL manages transactions, i.e., a group of SQL operations treated as a single unit.

19.6.1 COMMIT

Saves the current transaction permanently.

```
INSERT INTO Students VALUES (2, 'Ravi', 21, 'SQL');
COMMIT;
```

19.6.2 ROLLBACK

Undoes changes since the last COMMIT.

```
DELETE FROM Students WHERE StudentID = 2;
ROLLBACK;
```

19.6.3 SAVEPOINT

Creates a checkpoint so you can roll back to that point.

```
SAVEPOINT BeforeDelete;
DELETE FROM Students WHERE Age < 20;
ROLLBACK TO BeforeDelete;
```

Useful in complex transactions for partial undo.

19.7 Visual & Teaching Suggestions

- **Flowcharts for each command type**
- **Transaction timeline showing COMMIT/ROLLBACK/SAVEPOINT**
- **Comparison tables between DELETE vs TRUNCATE, GRANT vs REVOKE**
- **Infographic: SQL lifecycle (Design → Insert → Query → Control → Transact)**

19.8 Summary Table

Category	Commands	Description
DDL	CREATE, ALTER, DROP, TRUNCATE, RENAME	Structure-related commands
DML	INSERT, UPDATE, DELETE, MERGE	Data manipulation
DQL	SELECT (with clauses like WHERE, ORDER BY, etc.)	Data retrieval
DCL	GRANT, REVOKE	Permissions and access
TCL	COMMIT, ROLLBACK, SAVEPOINT	Transaction control

Chapter 20: SELECT Queries and Filtering

Introduction: SELECT queries and filtering are fundamental concepts in SQL used to retrieve specific data from a relational database. The SELECT statement allows users to choose which columns they want to view from a table, making it easy to extract only the needed information. By using filtering clauses like WHERE, users can set conditions to narrow down results—for example, selecting only customers from a specific city or orders above a certain amount. Additional keywords like ORDER BY, LIMIT, and BETWEEN further refine the output. These tools make SELECT queries powerful for extracting meaningful insights from large data sets quickly and efficiently.

20.1 The SELECT Statement

The SELECT statement is the most commonly used command in SQL. It allows users to retrieve specific data from one or more tables. The data can be displayed based on the column(s) you want, and you can filter or sort it as needed.

Basic Syntax:

```
SELECT column1, column2 FROM table_name;
```

Example:

```
SELECT Name, Age FROM Employees;
```

This query retrieves the names and ages of all employees from the "Employees" table.

20.2 Using WHERE Clause

The WHERE clause is used to filter rows that meet certain conditions.

Syntax:

```
SELECT * FROM Employees WHERE Age > 25;
```

This query selects all columns for employees whose age is greater than 25.

Operators in WHERE Clause:

- **=: Equal**
- **!= or <>: Not Equal**
- **>: Greater Than**
- **<: Less Than**
- **>=: Greater Than or Equal To**
- **<=: Less Than or Equal To**
- **BETWEEN: Filter within a range**
- **LIKE: Pattern matching**
- **IN: Check if value is within a list**
- **IS NULL / IS NOT NULL: Check for NULL values**

Examples:

- **SELECT * FROM Employees WHERE Age BETWEEN 25 AND 35;**
- **SELECT * FROM Employees WHERE Name LIKE 'A%';**
- **SELECT * FROM Employees WHERE Department IN ('HR', 'IT');**
- **SELECT * FROM Employees WHERE Email IS NOT NULL;**

20.3.ORDER BY Clause

The ORDER BY clause is used to sort the result set.

Syntax:

SELECT * FROM Employees ORDER BY Age DESC;

This query sorts employees by age in descending order. The default order is ascending (ASC).

Examples:

- **SELECT * FROM Employees ORDER BY Name ASC;**
- **SELECT * FROM Employees ORDER BY Department DESC, Age ASC;**

20.4.LIMIT Clause

The LIMIT clause is used to restrict the number of records returned.

Syntax:

SELECT * FROM Employees LIMIT 5; This query retrieves only the first 5 rows.

LIMIT with OFFSET:

SELECT * FROM Employees LIMIT 5 OFFSET 10; This returns 5 rows starting from the 11th row.

Note: TOP is used in SQL Server and FETCH FIRST in Oracle.

20.5 Combining Clauses

You can use SELECT, WHERE, ORDER BY, and LIMIT together for more advanced queries.

Example:

```
SELECT Name, Age FROM Employees
WHERE Age > 25
ORDER BY Age ASC
LIMIT 3;
```

This fetches the top 3 employees over the age of 25 sorted by age.

20.6 More Advanced Filtering

AND / OR:

```
SELECT * FROM Employees WHERE Age > 25 AND Department = 'IT';
SELECT * FROM Employees WHERE Age < 30 OR Department = 'HR';
```

NOT:

```
SELECT * FROM Employees WHERE NOT Department = 'Finance';
```

CASE WHEN (Conditional in SELECT):

```
SELECT Name, Age,
```

```
CASE
 WHEN Age >= 30 THEN 'Senior'
ELSE 'Junior'
END AS Category
FROM Employees;
```

20.7 Graphic Suggestions

- Diagram of a query pipeline (SELECT → WHERE → ORDER BY → LIMIT)
- Table example showing original data vs filtered/sorted output
- Tree diagram showing SELECT clause structure

This chapter introduces core data retrieval techniques using SELECT and filtering. Mastery of these tools forms the basis for all advanced SQL operations.

Chapter 21: Aggregations 3 COUNT, SUM, AVG, MIN, MAX, and More

21.1 What are Aggregate Functions?

Aggregate functions perform a calculation on a set of values and return a single result. They are typically used with the SELECT statement and often in combination with GROUP BY to summarize data.

21.2 COUNT() Function

The COUNT() function returns the number of rows that match a condition.

Syntax:

```
SELECT COUNT(*) FROM table_name;
```

Example:

```
SELECT COUNT(*) FROM Employees;
```

Returns the total number of employees.

```
SELECT COUNT(DISTINCT Department) FROM Employees;
```

Returns the number of unique departments.

21.3 SUM () Function

The SUM () function adds up all the numeric values in a column.

Syntax:

```
SELECT SUM(column_name) FROM table_name;
```

Example:

```
SELECT SUM(Salary) FROM Employees;
```

Calculates the total salary paid to all employees.

21.4 AVG() Function

The AVG() function returns the average value of a numeric column.

Example:

```
SELECT AVG(Age) FROM Employees;
```

Finds the average age of employees.

21.5 MIN() and MAX() Functions

- MIN() returns the smallest value.
- MAX() returns the largest value.

Example:

```
SELECT MIN(Salary), MAX(Salary) FROM Employees;
```

Shows the lowest and highest salary in the table.

21.6.GROUP BY Clause with Aggregates

Use GROUP BY to apply aggregate functions to subsets of data.

Example:

```
SELECT Department, AVG(Salary) AS Avg_Salary
FROM Employees
GROUP BY Department;
```

Returns the average salary per department.

21.7.HAVING Clause with Aggregates

HAVING filters grouped data after aggregation.

Example:

```
SELECT Department, COUNT(*)
FROM Employees
GROUP BY Department
HAVING COUNT(*) > 5;
```

Only shows departments with more than 5 employees.

21.8.Additional Aggregate Functions

1. **VARIANCE() and STDDEV():**

Used to calculate statistical values.

```
SELECT VARIANCE(Salary), STDDEV(Salary) FROM Employees;
```

Gives salary variance and standard deviation.

2. BOOL_AND(), BOOL_OR(): (Supported in PostgreSQL) Used to evaluate boolean conditions across rows.

```
SELECT BOOL_AND(Active), BOOL_OR(Active) FROM Employees;
```

Checks if all or any employee is active.

3. STRING_AGG():

Concatenates values into a single string (PostgreSQL, SQL Server)

```
SELECT STRING_AGG(Name, ', ') FROM Employees;
```

Returns a comma-separated list of employee names.

4.JSON_AGG(), ARRAY_AGG():

Returns results as JSON or arrays (PostgreSQL)

```
SELECT JSON_AGG(Name) FROM Employees;
```

21.9 Real-Life Applications

- **Counting total customers**
- **Calculating total sales**
- **Finding average order value**
- **Tracking highest and lowest selling products**

21.10 Graphic Suggestions

- **Bar chart showing MIN, MAX, AVG per department**
- **Pie chart using COUNT data for department sizes**
- **Line graph for SUM(Sales) by Month**

Chapter 22: GROUP BY and HAVING Clauses

22.1 Introduction to Data Grouping

In SQL, when you're dealing with large amounts of data, it's often important to organize or summarize that data in meaningful ways. This is where the GROUP BY clause comes in. It allows you to group rows that have the same values in specified columns into summary rows. For example, you might want to know how many sales were made by each employee, or the average marks of students in different classes.

22.2 GROUP BY Clause

- The GROUP BY clause is used with aggregate functions (like COUNT, SUM, AVG, MIN, MAX) to group the result-set by one or more columns.

Syntax:

```
SELECT column_name, AGGREGATE_FUNCTION(column_name)
FROM table_name GROUP

BY column_name;
```

Example:

```
SELECT department, COUNT(*) AS total_employees
FROM employees
GROUP BY department;
```

This query gives the total number of employees in each department.

22.3 Why GROUP BY is Useful

- It helps to summarize data.
- It reduces long rows into compact groups.
- It works great with reports and dashboards.

Real-life Use Case: In a shopping app, if you want to show how many products were sold in each category, you9d use GROUP BY to organize the data category-wise.

22.4 The HAVING Clause

While WHERE is used to filter rows before grouping, HAVING is used to filter after the data is grouped. This is essential when you want to filter groups based on aggregate values.

Syntax:

```
SELECT column_name, AGGREGATE_FUNCTION(column_name)
FROM table_name
GROUP BY column_name
```

HAVING AGGREGATE_FUNCTION(column_name) condition;

Example:

```
SELECT department, COUNT(*) AS total_employees
FROM employees
GROUP BY department
HAVING COUNT(*) > 5;
```

This query shows only those departments that have more than 5 employees.

22.5 GROUP BY with Multiple Columns

You can group by multiple columns to get more detailed summaries.

Example:

```
SELECT department, job_title, COUNT(*)
FROM employees
GROUP BY department, job_title;
```

This gives a count of employees by both department and job title.

22.6 GROUP BY with ORDER BY

You can sort grouped data using ORDER BY.

```
SELECT department, COUNT(*) AS total_employees
FROM employees
GROUP BY department
ORDER BY total_employees DESC;
```

This will display departments in order of number of employees, starting from the highest

22.7 Nested Aggregates and Subqueries

You can combine GROUP BY with subqueries for advanced analysis.

Example:

```
SELECT department
FROM (
 SELECT department, COUNT(*) AS emp_count
 FROM employees
 GROUP BY department
) AS dept_summary
WHERE emp_count = (
SELECT MAX(emp_count)
FROM (
SELECT department, COUNT(*) AS emp_count
FROM employees
GROUP BY department
 ) AS inner_dept
```

);

This finds the department with the highest number of employees.

22.8 Common Mistakes to Avoid

- Using HAVING without GROUP BY (unless you're doing full table aggregates).
- Referring to column aliases in the GROUP BY clause (use actual column names).
- Forgetting to include all non-aggregated columns in the GROUP BY clause.

22.9 Visual Example

Table: Sales

Product	Category	Quantity
A	Toys	10
B	Toys	20
C	Books	5
D	Toys	15

Query:

SELECT Category, SUM(Quantity) AS Total_Sold
FROM Sales GROUPBY Category; Output:

Category	Total_Sold
Toys	45
Books	5

22.10 Summary

- GROUP BY organizes data into groups.
- It works with aggregate functions to summarize data.
- HAVING filters grouped results based on aggregated conditions.
- Together they are essential for creating meaningful summaries in reports and dashboards.

22.11 Graphic Suggestions

- Grouped bar chart by department with employee count.
- Pie chart showing total sales by category.
- Flow diagram: SELECT → GROUP BY → HAVING → ORDER BY

Chapter 23: Inserting and Updating Data

23.1 INSERT INTO Statement

The INSERT INTO statement is used to add new records to a table in a database.

Syntax:

```
INSERT INTO table_name (column1, column2, ...)
VALUES (value1, value2, ...);
```

Example:

```
INSERT INTO students (id, name, age)
VALUES (101, 'Amit', 21);
```

Adds a new student record with ID 101.

23.2 Inserting Multiple Rows at Once

You can insert more than one row in a single statement.

Example:

```
INSERT INTO students (id, name, age) VALUES
(102, 'Neha', 20), (103, 'Ravi', 22);
```

Adds two new records in one go.

Advantages:

- **More efficient for bulk inserts.**
- **Reduces number of queries sent to the database.**

23.3 UPDATE Statement

The UPDATE statement is used to modify existing records in a table.

Syntax:

```
UPDATE table_name
SET column1 = value1, column2 = value2
WHERE condition; Example:
UPDATE students
SET age = 23
WHERE id = 101;
```

Updates the age of the student with ID 101.

Warning:

If you omit the WHERE clause, all records in the table will be updated.

```
UPDATE students
SET age = 25; -- This will change every student's age to 25
```

23.4 Visual Example: Before and After Update

Before:

ID	Name	Age
101	Amit	21

Update Query:

```
UPDATE students SET age = 22 WHERE id = 101;
```

After:

ID	Name	Age
101	Amit	22

23.5 Use Cases

INSERT: Add new employees, customers, or transactions.

UPDATE: Change prices, update contact info, correct typos.

Example Use Case:

```
INSERT INTO employees (id, name, department)
VALUES (201, 'Karan', 'Sales');
UPDATE employees
SET department = 'Marketing'
```

23.6 Best Practices for Inserting and Updating

- **Always use WHERE in UPDATE unless updating all rows is intended.**
- **Use transactions for bulk changes:**

```
BEGIN;
UPDATE students SET age = age + 1;
COMMIT;
```

- **Use parameterized queries in application development to avoid SQL injection.**
- **Validate data types and constraints before inserting.**

23.7 Additional Tips

- **Use DEFAULT values if not providing data for all columns:**

```
INSERT INTO students (id, name) VALUES (104, 'Priya');
```

- **Use RETURNING clause (in PostgreSQL) to get values back after insert/update:**

```
INSERT INTO students (name, age)
VALUES ('Sahil', 24)
RETURNING id;
```

- **To insert data from another table:**

```
INSERT INTO graduates (id, name, age)
SELECT id, name, age FROM students WHERE age > 22;
```

23.8 Graphic Suggestions

- **Table before and after INSERT**
- **Table before and after UPDATE**
- **Flow diagram: INSERT → Table → View in SELECT**

This chapter equips you with the practical know-how to add and update records safely and efficiently using SQL.

Chapter 24: Deleting Records and Data Integrity

24.1 Introduction

Data is dynamic, which means there are times when we need to remove outdated, incorrect, or unnecessary information from our database. The SQL DELETE statement allows us to do this. But removing data must be handled carefully to avoid accidental loss or breaking relationships between tables. That's where data integrity comes in — it ensures the correctness and consistency of data throughout its lifecycle.

Data deletion is common in real-life applications. For example:

- **Removing a user account when someone closes their profile**
- **Deleting expired coupons or outdated products**
- **Purging logs or temporary data after a certain time**

24.2 DELETE Statement in SQL

The DELETE statement is used to remove one or more rows from a table.

Syntax:

```
DELETE FROM table_name
WHERE condition;
```

Example:

```
DELETE FROM students
WHERE id = 104;
```

This deletes the student record where the ID is 104.

Warning: If you forget to include the WHERE clause, all records in the table will be deleted.

Deleting All Rows:

```
DELETE FROM students;
```

This removes every row but retains the table structure.

Deleting Using Subquery:

You can also delete using another query as a condition:

```
DELETE FROM orders
WHERE customer_id IN (SELECT id FROM customers WHERE status = 'inactive');
```

Deletes orders of inactive customers.

24.3.Truncate vs Delete vs Drop

These three commands are often confused. Here9s a comparison:

Command	Removes Rows	Rollback Possible	Structure Remains	Speed	Usage Notes

DELETE	Yes	Yes (with transaction)	Yes	Moderate	Most flexible, allows filtering and triggers
TRUNCATE	Yes (All)	No	Yes	Fast	Cannot filter, no trigger execution
DROP	Yes (All)	No	No	Fastest	Removes the entire table definition

24.4.Example: Delete Based on Conditions

DELETE FROM employees
WHERE salary < 3000 AND department = 'HR';
This removes all HR employees with salaries less than 3000.
Delete Using JOIN (MySQL-style):
DELETE e FROM employees e
JOIN departments d ON e.dept_id = d.id
WHERE d.name = 'Temporary';
Deletes employees from a department called 'Temporary'.

24.5 Maintaining Data Integrity

Data integrity ensures your data is accurate, complete, and consistent. SQL usesvarious constraints and rules to maintain data integrity:

1. **Entity Integrity**
   - **Ensures each row in a table is unique and identifiable.**
   - **Enforced using Primary Keys.**
2. **Referential Integrity**
   - **Maintains valid references between related tables.**
   - **Enforced using Foreign Keys.**
   - **Prevents deletion of records that are being used in another table unless handled correctly.**
3. **Domain Integrity**
   - **Ensures data falls within a specific domain, type, or format.**
   - **Enforced using:**
   - **Data types (VARCHAR, INT, etc.)**
   - **Constraints like CHECK, NOT NULL, and DEFAULT**

24.6 Example of Broken Integrity

Imagine deleting a student from the students table who is also referenced in the results table. Without proper foreign key handling, this could break the relationship, resulting in orphaned records.
To prevent this, foreign keys can be defined with cascading rules.

Example: CASCADE Delete:

```
CREATE TABLE
results ( student_id
INT, score INT,
FOREIGN KEY (student_id) REFERENCES students(id) ON DELETE CASCADE
);
```

If a student is deleted, their results are deleted automatically.

Other Options:

- **ON DELETE SET NULL: Sets the reference to NULL**
- **ON DELETE RESTRICT: Prevents deletion if there are dependent records**

24.7 Best Practices

- ✅ **Always take a backup before executing delete operations.**
- ✅ **Use WHERE with DELETE to avoid full table deletion.**
- ✅ **Use transactions with BEGIN, COMMIT, and ROLLBACK for safer deletes.**
- ✅ **Log or audit deletions especially in sensitive or financial systems.**
- ✅ **Use FOREIGN KEY constraints to protect related data.**
- ✅ **Avoid deleting frequently — consider marking data as inactive.**

24.8 Soft Delete Example

Rather than deleting data permanently, you can mark it as inactive or archived.

Update Instead of Delete:

```
UPDATE students SET status = 'inactive' WHERE id = 104;
```

This keeps the data but marks it inactive for future filtering.

Query to Exclude Inactive Records:

```
SELECT * FROM students WHERE status != 'inactive';
```

Benefits of Soft Delete:

- **Allows recovery of mistakenly deleted data**
- **Useful for audit trails**
- **Required in some compliance-driven industries**

24.9 Summary

Use DELETE to remove specific records from tables.

Be cautious and use WHERE clause to avoid removing unintended data.

Understand differences: DELETE (row-wise), TRUNCATE (bulk, fast), DROP (remove table).

Maintain data integrity using primary keys, foreign keys, and constraints.

Use soft deletes where records must be kept for history or compliance.
Wrap complex deletes in transactions and validate changes before committing.

24.10 Visual Suggestions

- **Comparison Diagram: DELETE vs TRUNCATE vs DROP**
- **Flowchart: Safe deletion workflow using backup → transaction → delete → commit**
- **ER Diagram: Foreign key with ON DELETE CASCADE explained**
- **Soft Delete Example: Before-and-after row display with status column**

Chapter 25: Copying Table Structure and Data in SQL

25.1 Introduction

When working with databases, there are times you want to create a new table that is similar to an existing one. Sometimes you need only the structure (columns, data types, constraints) without any data, and other times you want to copy both structure and data. This chapter explains how to copy the structure of a table, and how to copy both structure and data.

25.2 Copying Only the Table Structure

If you want a new table with the same columns and data types as an existing table but without any data, you can use the CREATE TABLE ... LIKE statement (supported in MySQL and some other SQL variants).

Syntax:

```
CREATE TABLE new_table LIKE existing_table;
```

Example:

```
CREATE TABLE new_employees LIKE employees;
```

- This creates a new table named new_employees.
- The new table will have exactly the same columns, data types, indexes, and constraints as the employees table.
- But it will be empty — no rows are copied over.

This is useful when you want a blank table for testing, staging, or to insert new data later without affecting the original table.

25.3 Copying Table Structure and Data Together

If you want to create a new table that copies both the structure and the data from an existing table, you can use the CREATE TABLE ... AS SELECT statement.

Syntax:

```
CREATE TABLE new_table AS
SELECT * FROM existing_table;
```

Example:

```
CREATE TABLE backup_employees AS
SELECT * FROM employees;
```

- This creates a new table backup_employees.
- It copies all rows from employees into this new table.
- The new table's columns and data types are created based on the SELECT query.

Note:

- **This method copies data but may not copy constraints (like primary keys, unique indexes, foreign keys) or some column-specific properties. You might need to add those manually if required.**
- **This syntax works in many databases like MySQL, PostgreSQL, and SQL Server (with some variations).**

25.4 Important Considerations

- **Indexes and Constraints:**

When copying structure using CREATE TABLE ... LIKE, indexes and constraints are copied.
When copying structure and data using CREATE TABLE ... AS SELECT, indexes and constraints are not copied. You may need to add them separately.

- **Data Types and Defaults:**

Using CREATE TABLE ... LIKE keeps data types, default values, and column comments intact.
Using CREATE TABLE ... AS SELECT infers data types based on the data selected.

- **Performance:**

Copying structure is faster because no data is copied. Copying data along with structure may take longer depending on the size of the data.

25.5 Examples for Practice

Example 1: Copy only structure

```
CREATE TABLE test_employees LIKE employees;
```

Example 2: Copy structure and data

```
CREATE TABLE archived_employees AS
SELECT * FROM employees
WHERE department = 'Sales';
```

This creates archived_employees with only Sales department employees.
Example 3: Manually adding constraints after copying data

```
ALTER TABLE archived_employees
ADD PRIMARY KEY (id);
```

25.6 Summary

- **Use CREATE TABLE new_table LIKE existing_table to copy table structure including indexes and constraints, without data.**
- **Use CREATE TABLE new_table AS SELECT * FROM existing_table to copy structure and data, but without indexes and constraints.**
- **After copying data, add any needed constraints manually for data integrity.**
- **Choose the method based on your need: blank table or table with data.**

Chapter 26: Constraints in SQL

26.1 Introduction: Why Constraints Matter

Constraints in SQL are rules that enforce certain conditions on data in your database tables. Without constraints, databases could easily end up storing invalid, inconsistent, or incomplete data, which makes applications unreliable and error-prone. Constraints are like guards or validation rules that protect your data's integrity automatically. This means you don9t have to rely solely on application logic to keep data clean—databases handle it for you.

26.2 NOT NULL Constraint

Purpose:

- Ensures that a column cannot have a NULL (empty) value.
- Used when a field must always have a value (e.g., a username, email, product name).

How it works:

If you try to insert or update a row without providing a value for a NOT NULL column, the database will throw an error and reject the operation.

Syntax:

```
CREATE TABLE users (
  user_id INT,
   username VARCHAR(50) NOT NULL  -- username cannot be null
    );
```

Example:

```
INSERT INTO users (user_id) VALUES (1);
```

This will fail because username is missing and it's NOT NULL

26.3 UNIQUE Constraint

Purpose:

Guarantees that every value in a column is unique across all rows.

Important for columns like email, phone number, or any identifier that must not repeat.

Difference from PRIMARY KEY:

- UNIQUE allows one NULL value (depending on DBMS).
- You can have multiple UNIQUE columns in a table.
- A table can have only one PRIMARY KEY.

Syntax:

```
CREATE TABLE customers (
 id INT,
```

email VARCHAR(100) UNIQUE -- emails must be unique
);
Example:
INSERT INTO customers (id, email) VALUES (1, 'abc@example.com');
INSERT INTO customers (id, email) VALUES (2, 'abc@example.com');
This will fail due to duplicate email

26.4 PRIMARY KEY Constraint

Purpose:

- Uniquely identifies each row in a table.
- Combines UNIQUE + NOT NULL properties.

Details:

- Each table can have only one primary key.
- Can be a single column or a combination of multiple columns (composite key).
- Often used for IDs or unique identifiers.

Syntax (single column):

```
CREATE TABLE products (
  product_id INT PRIMARY KEY,
  name VARCHAR(50)
);
```

Composite Primary Key example:

```
CREATE TABLE order_items (
 order_id INT, product_id INT,
  PRIMARY KEY (order_id, product_id) -- combination must be unique
);
```

Example:

```
INSERT INTO products (product_id, name) VALUES (1, 'Pen');
INSERT INTO products (product_id, name) VALUES (1, 'Pencil');
```

Fails because product_id must be unique

26.5 FOREIGN KEY Constraint

Purpose:

- Maintains referential integrity between two tables by linking columns.
- Ensures that a value in the child table exists in the parent table.

Why use it?

To enforce relationships like <each order must belong to a valid customer=.

Syntax:

```
CREATE TABLE orders ( order_id
INT PRIMARY KEY, customer_id
INT,
FOREIGN KEY (customer_id) REFERENCES customers(id)
```

);

ON DELETE CASCADE / ON UPDATE CASCADE:

Automatically delete or update child rows when the parent row changes.

Example: If a customer is deleted, all their orders can be deleted automatically.

FOREIGN KEY (customer_id) REFERENCES customers(id) ON DELETE CASCADE

Example:

Trying to insert an order with non-existent customer

INSERT INTO orders (order_id, customer_id) VALUES (1, 999);

This will fail if customer 999 doesn't exist

26.6 CHECK Constraint

Purpose:

- **Enforces a condition that data in a column must satisfy.**
- **Useful for enforcing business rules (e.g., salary > 0).**

Syntax:

```
CREATE TABLE employees (
 id INT,
 salary DECIMAL(10,2) CHECK (salary > 0)
    );
```

Complex conditions:

CHECK (salary >= 3000 AND salary <= 100000)

Example:

INSERT INTO employees (id, salary) VALUES (1, -500);

Fails because salary cannot be negative

26.7 DEFAULT Constraint

Purpose:

- **Automatically assigns a value to a column if none is provided during insertion.**
- **Helps avoid NULL or unexpected blanks in data.**

Syntax:

```
CREATE TABLE users (  id INT,
status VARCHAR(10) DEFAULT 'active'
);
```

Example:

```
INSERT INTO users (id) VALUES (1);
SELECT * FROM users WHERE id = 1;
```

status will be 'active' by default

26.8 Combined Constraints

You can apply multiple constraints to a single column for tighter control.

Example:

```
CREATE TABLE students ( roll_no INT PRIMARY
KEY, name VARCHAR(50) NOT NULL, email
VARCHAR(100) UNIQUE NOT NULL
);
```

- email must be unique and cannot be NULL.
- roll_no uniquely identifies students.

26.9 Naming Constraints

Naming constraints explicitly makes it easier to modify or drop them later.

Syntax:

```
CREATE TABLE sales (
 id INT,
amount DECIMAL(10, 2),
CONSTRAINT chk_amount CHECK (amount > 0)
);
```

Why name constraints?

- Easier to identify in error messages.
- Needed when altering or dropping constraints.

26.10 Altering Constraints After Table Creation

- Add a constraint:
- ALTER TABLE employees

```
ADD CONSTRAINT emp_email_unique UNIQUE (email);
```

- Drop a constraint:
- ALTER TABLE employees
- DROP CONSTRAINT emp_email_unique;

Important:

- Constraint name is required for dropping.
- Some databases use slightly different syntax (e.g., MySQL uses DROP INDEX for unique constraints).

26.11 Real-World Use Case: Online Store Database

Constraint Type	Purpose in Store Database	Example
PRIMARY KEY	Unique product or order identifier	product_id INT PRIMARY KEY
FOREIGN KEY	Link orders to customers	FOREIGN KEY (customer_id)

NOT NULL	**Mandatory product name or user email**	**name VARCHAR(100) NOT NULL**
CHECK	**Price must be positive**	**CHECK (price > 0)**
UNIQUE	**Prevent duplicate customer emails**	**email VARCHAR(100) UNIQUE**

These constraints keep the database consistent and protect it from invalid data input.

26.12 Summary

- **Constraints protect data integrity and enforce rules.**
- **Use NOT NULL to require data.**
- **Use UNIQUE to avoid duplicates.**
- **Use PRIMARY KEY to identify rows uniquely.**
- **Use FOREIGN KEY to maintain relationships between tables.**
- **Use CHECK to enforce specific conditions.**
- **Use DEFAULT to auto-assign default values.**
- **Constraints can be combined, named, and altered after creation.**
- **Applying constraints properly ensures a reliable and secure database.**

Chapter 27: Transactions in SQL

27.1 Introduction: Why Do We Need Transactions?

Imagine you9re sending money via an online banking app:

- Step 1: Deduct money from your account.
- Step 2: Add money to the receiver9s account.

If the system crashes after step 1 but before step 2, the money disappears from your account but never reaches the receiver — this causes data inconsistency and financial loss.

A transaction is a way to ensure that all steps succeed together or none at all This concept is called atomicity, meaning the entire operation is indivisible.

Without transactions, partial updates could corrupt your data.

27.2 What Exactly is a Transaction?

A transaction is a group of SQL commands executed as one single unit. If all commands succeed, the database saves the changes permanently.

If any command fails, all changes made by previous commands in that transaction are undone, restoring the database to its original state.

Example

Suppose you want to move 100 units from Account A to Account B:

```
START TRANSACTION;
UPDATE accounts SET balance = balance - 100 WHERE account_id = 'A';
UPDATE accounts SET balance = balance + 100 WHERE account_id = 'B';
COMMIT;
```

If the second update fails, you don9t want the first update to stay — you want to rollback the first one too.

27.3 ACID Properties: The Pillars of Transactions

Transactions follow ACID properties to guarantee reliability:

Property	What It Means in Simple Terms
Atomicity	Everything inside the transaction happens or nothing happens. No partial updates.
Consistency	Data must always follow all rules, constraints, and business logic before and after.
Isolation	Transactions happen independently without affecting each other9s intermediate steps.
Durability	Once committed, data changes are permanent, surviving crashes or power failures.

More on Each Property

Atomicity: If you pay for a product online, either the payment and order placement both succeed, or neither does.

Consistency: The database should never allow invalid data like negative account balances.
Isolation: If two people try to book the last seat on a flight at the same time, isolation prevents double booking.
Durability: After you complete a transaction and get confirmation, your data is safely stored even if the server suddenly shuts down.

27.4 How to Start and End a Transaction

Starting a Transaction
You can explicitly start a transaction by:

```
START TRANSACTION;
```

or

```
BEGIN;
```

Some databases start transactions implicitly, but explicit control is safer.
Ending a Transaction

- **COMMIT saves all changes permanently.**
- **ROLLBACK cancels all changes made in the current transaction.**

Example:

```
START TRANSACTION;
-- some SQL operations
COMMIT;  -- saves changes
-- or if something goes wrong
ROLLBACK;  -- undo changes
```

27.5 COMMIT: Making Changes Permanent

COMMIT finalizes your transaction.
Before COMMIT, other users might not see your changes depending on the isolation level.
Once COMMIT runs, your changes are saved and durable.
Important Notes:

- **After COMMIT, you cannot rollback those changes anymore.**
- **If the system crashes after commit, changes are still saved safely.**

27.6.ROLLBACK: Undoing Changes

If an error occurs during your transaction, ROLLBACK restores the database to the state before the transaction began.
Example:

```
START TRANSACTION;
UPDATE accounts SET balance = balance - 100 WHERE id = 1;
-- Suppose here an error occurs (e.g., account 2 doesn't exist)
ROLLBACK;
```

After ROLLBACK, the balance of account 1 remains unchanged.

27.7.SAVEPOINT: Partial Rollbacks Inside a Transaction

Sometimes transactions are long or complex. You may want to mark checkpoints inside the transaction so you can undo part of the work without discarding the entire transaction.

How to use:

```
START TRANSACTION;
UPDATE orders SET status = 'Processing';
SAVEPOINT sp1;
UPDATE orders SET status = 'Shipped';
ROLLBACK TO sp1;
COMMIT;
```

In this example:

- The first update sets status to 8Processing9.
- Then you mark a savepoint named sp1.
- Next update changes status to 8Shipped9.
- ROLLBACK TO sp1 undoes only the 8Shipped9 update, keeping 8Processing9.
- Then COMMIT saves the final state.

27.8.RELEASE SAVEPOINT: Freeing Resources

If you no longer need a savepoint:

```
RELEASE SAVEPOINT sp1;
```

This removes the savepoint marker but does not affect your transaction.

27.9.Nested Transactions: Not Fully Supported, But Possible With Save points

Most SQL databases do not support true nested transactions (transactions inside transactions).

However, savepoints allow you to simulate nested behavior by allowing partial rollbacks inside a transaction.

27.10.Auto-Commit Mode

By default, many SQL engines operate in auto-commit mode, which means:

Every SQL statement is treated as a single transaction and committed immediately.

If you want to group multiple statements into one transaction, you need to disable auto-commit:

```
SET auto commit = 0;
```

Then you control transactions manually.

27.11.Real-World Use Case: Banking Transfer

Imagine transferring money between two accounts:

```
START TRANSACTION;
UPDATE accounts SET balance = balance - 1000 WHERE acc_no = '1234';
```

UPDATE accounts SET balance = balance + 1000 WHERE acc_no = '5678';
COMMIT;
If any step fails, run:
ROLLBACK;
This ensures money isn9t lost or created accidentally.

27.12.Error Handling in Programming

When using SQL in real applications (Python, Java, PHP), you combine transactions with error handling.
Example in pseudocode (Python-style):

```
try:
 connection.start_transaction()
# Execute multiple SQL commands
cursor.  Execute(...)          cursor.
Execute(...)
connection. Commit() except
Exception as e:    connection.
Rollback()
print("Transaction failed:", e)
```

This approach ensures the database remains consistent even if errors happen in your program.

27.13.Best Practices for Using Transactions

- **Use transactions whenever multiple related updates must succeed together.**
- **Avoid very long transactions — they hold locks and can slow down other users.**
- **Use SAVEPOINTS to add checkpoints in big transactions.**
- **Always handle errors and rollback if necessary.**
- **Use transactions inside stored procedures for better control and reliability.**
- **Monitor transaction isolation levels depending on your use case for concurrency control.**

27.14 Summary: Why Transactions Matter

- **Transactions keep your database accurate, consistent, and reliable.**
- **They protect against partial updates or failures.**
- **SQL provides commands to start, commit, rollback, and checkpoint transactions.**
- **Following ACID principles ensures your system handles complex, concurrent operations safely.**

Chapter 28: SQL Joins 3 Combining Data from Multiple Tables

28.1 What are Joins, Really?

Imagine a library:

- **One table lists Books with book IDs.**
- **Another table lists Authors with author IDs.**
- **You want to know which author wrote which book.**

Since the information is split, you use a JOIN to combine these tables based on a shared column (like author_id).

Joins let you combine related rows from multiple tables into a single meaningful result.

28.2 Why Split Data into Multiple Tables?

- **To avoid duplication: If customer info repeated in every order, it wastes space and risks inconsistency.**
- **To maintain consistency: Changing a customer9s info once updates it everywhere.**
- **This method is called Normalization — organizing data to reduce redundancy.**
- **Joins let you bring normalized data back together as needed.**

28.3 Basic Terminology

Table: Collection of rows and columns storing data.

Primary Key (PK): Unique identifier in a table (e.g., customer_id).

Foreign Key (FK): Column in one table referencing the PK of another table, linking the tables.

28.4 INNER JOIN 4 The "Intersection"

- **Returns only rows with matching values in both tables.**
- **Think of overlapping circles; the intersection is returned.**
- **If no match, rows are excluded.**

Example:

Customers

customer_id	name
1	Alice
2	Bob
3	Charlie

Orders

order_id	customer_id	product
101	1	Laptop
102	3	Smartphone

Query:

```
SELECT Customers.name, Orders.product
FROM Customers
INNER JOIN Orders ON Customers.customer_id = Orders.customer_id;
```

Result:

name	product
Alice	Laptop
Charlie	Smartphone

Bob is excluded as he has no orders.

28.5 LEFT JOIN 4 "All Left + Matching Right"

- **Returns all rows from the left table.**
- **Returns matching rows from the right.**
- **For non-matches, right columns show NULL.**

Query:

```
SELECT Customers.name, Orders.product
FROM Customers
LEFT JOIN Orders ON Customers.customer_id = Orders.customer_id;
```

Result:

name	product
Alice	Laptop
Bob	NULL
Charlie	Smartphone

Bob appears with NULL product — useful when you want *all* customers regardless of orders.

28.6 RIGHT JOIN 4 "All Right + Matching Left"

- **Returns all rows from the right table.**
- **Returns matching rows from the left.**
- **For non-matches, left columns are NULL.**

Example:

```
SELECT Customers.name, Orders.product
FROM Customers
RIGHT JOIN Orders ON Customers.customer_id = Orders.customer_id;
```

If there are orders with no customers (rare but possible), they appear with NULL in Customers columns.

28.7 FULL OUTER JOIN 4 "All Rows from Both"

- **Returns all rows from both tables.**
- **Where no match exists, missing side columns show NULL.**
- **Combines LEFT and RIGHT join results.**

Example:

```
SELECT Customers.name, Orders.product
FROM Customers
FULL OUTER JOIN Orders ON Customers.customer_id = Orders.customer_id;
```

Result example:

name	product
Alice	Laptop
Bob	NULL
Charlie	Smartphone
NULL	UnmatchedProduct

Note: Not supported in MySQL natively, but available in PostgreSQL, SQL Server.

28.8 CROSS JOIN 4 "Every Possible Combination"

- **Returns all combinations of rows from both tables.**
- **Number of rows = rows in Table A × rows in Table B.**
- **Useful for generating all pairs or combinations.**

Example:

Customers: Alice, Bob
Products: Laptop, Smartphone

```
SELECT Customers.name, Products.product_name
FROM        Customers
CROSS JOIN Products;
```

Result:

name	product_name
Alice	Laptop
Alice	Smartphone
Bob	Laptop
Bob	Smartphone

28.9 SELF JOIN 4 Joining a Table to Itself

- Useful for hierarchical or recursive relationships.
- Use table aliases to differentiate.

Example: Employees with managers:

employee_id	name	manager_id
1	Alice	NULL
2	Bob	1
3	Charlie	1

```
SELECT E1.name AS Employee, E2.name AS Manager
FROM Employees E1
LEFT JOIN Employees E2 ON E1.manager_id = E2.employee_id;
```

Result:

Employee	Manager
Alice	NULL
Bob	Alice
Charlie	Alice

28.10 Join Conditions and Using Aliases

- Always use ON clause for join conditions to avoid huge unwanted results.
- Use aliases for tables to write concise and readable queries.

Especially necessary in SELF JOINs. Example:

```
SELECT c.name, o.product
FROM Customersc
INNER JOIN Orders o ON c.customer_id = o.customer_id;
```

28.11 Multiple Joins in a Single Query

You can join more than two tables together.

Example:

```
SELECT c.name, o.order_id, p.product_name, p.price
FROM Customers c
JOIN Orders o ON c.customer_id = o.customer_id
JOIN Products p ON o.product_id = p.product_id;
```

This combines customer info, order details, and product prices.

28.12 Performance Considerations

- Joins on very large tables can be slow.
- Use indexes on join columns to speed up matching.
- Filter rows early with WHERE clause to reduce the amount of data processed.
- Avoid unnecessary joins.

28.13 Practical Use Cases for Data Analysts

- **Combining customer profiles with their purchase history.**
- **Linking product catalogs with inventory data.**
- **Associating employee records with department details.**
- **Analyzing orders alongside payment and shipment status.**

28.14 Summary

Join Type	Description	When to Use
INNER JOIN	Rows matched in both tables only	When only matching data is needed
LEFT JOIN	All rows from left + matching from right	When all left data is required, plus matches
RIGHT JOIN	All rows from right + matching from left	When all right data is required, plus matches
FULL OUTER JOIN	All rows from both tables	When all data is needed regardless of match
CROSS JOIN	All combinations of both tables	When generating all pairs/combinations
SELF JOIN	Joining a table with itself	For hierarchical or recursive relationships

Chapter 29: Subqueries and Set Operations in SQL

29.1 Introduction: Why Subqueries and Set Operations?

Imagine this scenario:

You9re a data analyst. Your boss asks:

- **<Give me all customers who placed an order above the average.=**
- **<List users found in database A but not in database B.=**

To solve this, you9d need more than basic SELECT&FROM queries. You need:

- **Subqueries (queries inside other queries)**
- **Set operations (like merging or comparing results)**

These tools allow SQL to think like a human, breaking big tasks into smaller decisions.

29.2 What is a Subquery?

➤ Definition:

A subquery is a query embedded inside another SQL query.

Syntax Format:

```
SELECT column_name
FROM table_name
WHERE column_name operator (SELECT column_name FROM another_table);
```

➤ Where can Subqueries Appear?

Clause	Description
WHERE	Filter based on another query result
FROM	Treat subquery as a temporary table (inline view)
SELECT	Use subquery to calculate a column value

29.3.Types of Subqueries

Type	Description	Example Return
Single-row	Returns one value	=, <, >
Multi-row	Returns multiple rows (one column)	IN, ANY, ALL
Multi-column	Returns multiple rows and columns	Used with EXISTS, comparisons
Correlated Subquery	Refers to columns of the outer query	Executed row-by-row

29.4.Subqueries in WHERE Clause

Goal:

Filter data based on conditions coming from another query.

✅ **Example:**

Find names of customers who placed at least one order:

```
SELECT name
FROM Customers
WHERE customer_id IN (
SELECT customer_id FROM Orders
);
```

Inner Query Output:

List of customer_ids with orders.

Outer Query:

Returns names where customer_id is in the above list.

29.5.Subqueries with Comparison Operators

You can compare a column to a subquery9s result.

✅ **Example:**

Find orders greater than the average:

```
SELECT order_id, amount
FROM Orders
WHERE amount > (
SELECT AVG(amount) FROM Orders
);
```

- **SELECT AVG(amount) returns one value (scalar).**
- **Outer query filters those with amount greater than that.**

29.6 Subqueries in FROM Clause (Inline Views)

Used to temporarily calculate results and then filter them.

✅ **Example:**

Find customers with total spending > 1000:

```
SELECT customer_id, total_sales
FROM (
SELECT customer_id, SUM(amount) AS total_sales
FROM Orders
GROUP BY customer_id ) AS sales
WHERE total_sales > 1000;
```

The inner query creates a table of totals per customer. The outer query filters totals above 1000.

29.7 Correlated Subqueries

A correlated subquery depends on each row of the outer query.

✅ **Example:**

Find orders that are greater than the average of that specific customer9s other orders:

```
SELECT order_id, customer_id, amount
FROM Orders o1
WHERE amount > (
SELECT AVG(amount)
FROM Orders o2
WHERE o1.customer_id = o2.customer_id
);
```

- **Inner query calculates average per customer.**
- **Compared row-by-row using aliases o1 and o2.**

29.8 Performance Tips for Subqueries

Tip	Why?
Use EXISTS instead of IN	Faster with large datasets
Test subqueries separately	Easier to debug
Avoid correlated subqueries	They're slow (run per row)
Use JOIN if possible	Joins are usually more optimized

29.9 Set Operations Overview

Set operations combine result sets from two or more SELECT queries.

Operation	Purpose
UNION	Combine and remove duplicates
UNION ALL	Combine and keep duplicates
INTERSECT	Return only matching rows in both
EXCEPT/MINUS	Return rows in first not in second

Columns in both queries must have same number and data type.

29.10 UNION vs UNION ALL

✅ **Example:**

```
SELECT customer_id FROM Customers_US
UNION
```

SELECT customer_id FROM
Customers_Europe;
Removes duplicate customer IDs.
SELECT customer_id FROM Customers_US
UNION ALL
SELECT customer_id FROM Customers_Europe; Keeps duplicates (shows frequency).

29.11 INTERSECT

Returns only records present in both result sets.

✅ **Example:**

SELECT customer_id FROM Customers_US
INTERSECT
SELECT customer_id FROM Customers_Europe;

- **Shows customers who exist in both regions.**

Not supported in MySQL directly. Use JOIN with GROUP BY workaround.

29.12 EXCEPT / MINUS

Shows records from first query not in the second.

✅ **Example:**

SELECT customer_id FROM Customers_US
EXCEPT
SELECT customer_id FROM Customers_Europe;

- **Shows customers exclusive to US.**

MySQL doesn9t support EXCEPT directly. Use LEFT JOIN + WHERE IS NULL.

29.13.Real-World Use Cases

Task	Use
Customers with above-average orders	Subquery
Products not sold this year	EXCEPT
All clients from multiple sources	UNION
Common records across departments	INTERSECT
Employee earning more than dept average	Correlated Subquery

29.14.Subqueries vs JOINs

Subquery	JOIN
More readable for nested logic	Better performance
Great for filtering	Great for combining data from tables
Harder to optimize sometimes	Easier for DB engine to optimize

29.5.Summary Table

Concept	Description
Subquery	Query inside another query
WHERE Clause	Filter using result from another query
FROM Clause	Use subquery as a temporary table
Correlated Subquery	Inner query depends on outer
UNION	Combine, remove duplicates
UNION ALL	Combine, keep duplicates
INTERSECT	Common records in both
EXCEPT / MINUS	Only in first, not in second

Chapter 30: Window Functions in SQL

In SQL, window functions allow you to perform advanced calculations across rows related to the current one without aggregating or filtering them. They are powerful for analytics, reporting, ranking, and running totals.

30.1 What Are Window Functions?

- **Window functions calculate results across a "window" of rows related to the current row.**
- **Unlike aggregate functions (SUM, AVG, etc.), window functions do not reduce the number of rows.**
- **Each row retains its individuality while still showing the result of a calculation based on a group.**

Key Components:

- **OVER() clause: Defines the window.**
- **PARTITION BY: Breaks the result set into partitions (like GROUP BY).**
- **ORDER BY: Defines the order of rows within each partition.**

30.2 Syntax of Window Functions

```
FUNCTION_NAME() OVER (
 PARTITION BY column_name
 ORDER BY column_name [ASC|DESC]
)
```

- **PARTITION BY: Optional; divides data into subsets.**
- **ORDER BY: Optional but important for ranking or cumulative operations.**

30.3 Types of Window Functions

Category	Common Functions
Ranking Functions	ROW_NUMBER(), RANK(), DENSE_RANK(), NTILE()
Value Functions	LEAD(), LAG(), FIRST_VALUE(), LAST_VALUE(), NTH_VALUE()
Aggregate Functions	SUM(), AVG(), COUNT(), MIN(), MAX() with OVER()
Statistical/Other	CUME_DIST(), PERCENT_RANK()

30.4 Ranking Functions

30.4.1 ROW_NUMBER()

- **Assigns a unique sequential number to each row.**
- **Ties are broken arbitrarily — no duplicates.**

```
SELECT name, salary,
 ROW_NUMBER() OVER (ORDER BY salary DESC) AS row_num
FROM Employees;
```

Use case: When each row must have a unique number, even if values are the same.

30.4.2 RANK()

- **Assigns a rank, same rank to ties.**
- **Skips the next ranks if there's a tie.**

```
SELECT name, salary,
 RANK() OVER (ORDER BY salary DESC) AS salary_rank
FROM Employees;
```

Use case: Leaderboards where ties share ranks, and gaps are needed.

30.4.3 DENSE_RANK()

Like RANK(), but does not skip the next ranks.

```
SELECT name, salary,
 DENSE_RANK() OVER (ORDER BY salary DESC) AS dense_rank
FROM Employees;
```

Use case: When consecutive ranking is required without gaps.

30.4.4 NTILE(n)

Divides rows into n buckets of nearly equal size.
Assigns a bucket number.

```
SELECT name, score,
 NTILE(4) OVER (ORDER BY score DESC) AS quartile
FROM Students;
```

Use case: To create quartiles, deciles, or buckets in analysis.

30.5 Value-Based (Offset) Functions

These functions access data from another row in the result set relative to the current row.

30.5.1 LAG(column, offset, default)

Looks backward by N rows.

```
SELECT name, sales,
LAG(sales, 1, 0) OVER (ORDER BY date) AS previous_sales
FROM Sales;
```

Use case: Calculate difference from previous row.

30.5.2 LEAD(column, offset, default)

Looks forward by N rows.

```
SELECT name, sales,
 LEAD(sales, 1, 0) OVER (ORDER BY date) AS next_sales
FROM Sales;
```

Use case: Project or compare with future rows.

30.5.3 FIRST_VALUE(column)

Returns the first value in the window.

```
SELECT name, sales,
FIRST_VALUE(sales) OVER (PARTITION BY region ORDER BY date) AS first_sale
FROM Sales;
```

30.5.4 LAST_VALUE(column)

Returns the last value in the window (beware of frame definition).

```
SELECT name, sales,
LAST_VALUE(sales) OVER (PARTITION BY region ORDER BY date ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS last_sale
FROM Sales;
```

30.5.5 NTH_VALUE(column, n)

- Returns the nth value in the ordered window.

```
SELECT name, sales,
 NTH_VALUE(sales, 3) OVER (ORDER BY sales DESC) AS third_top_sale
FROM Sales;
```

30.6 Aggregate Window Functions

Applies aggregate functions without collapsing rows.

```
SELECT name, salary,
AVG(salary) OVER (PARTITION BY department) AS avg_dept_salary
FROM Employees;
```

Supported functions:

- SUM()
- AVG()
- COUNT()
- MIN()
- MAX()

30.7 Statistical/Distribution Functions

30.7.1 CUME_DIST()

Cumulative distribution: Fraction of rows with values less than or equal to current row.

```
SELECT name, score,
CUME_DIST() OVER (ORDER BY score DESC) AS cum_dist
FROM Students;
```

30.7.2 PERCENT_RANK()

- Relative rank between 0 and 1 based on total rows.

```
SELECT name, score,
PERCENT_RANK() OVER (ORDER BY score DESC) AS percent_rank
FROM Students;
```

30.8 Frame Clause (Advanced Control)

You can define what portion of data the window function should process:

```
ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
```

Examples:

- ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING

- **RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW**

Helps control moving averages, cumulative totals, etc.

30.9 Real-World Examples

1. Top N salaries per department

```
SELECT * FROM (
SELECT name, department, salary,
ROW_NUMBER() OVER (PARTITION BY department ORDER BY salary DESC) AS rn
FROM Employees
) AS ranked
WHERE rn <= 3;
```

2. Compare sales with previous month

```
SELECT customer_id, sale_month, amount,
LAG(amount, 1, 0) OVER (PARTITION BY customer_id ORDER BY sale_month) AS prev_amount
FROM Sales;
```

30.10.Window Functions vs Aggregate Functions

Feature	Aggregate Functions	Window Functions
Collapse rows	Yes	No
Retain all rows	No	Yes
Use cases	Summarizing data	Ranking, comparisons, trends
Example	SUM(sales)	SUM(sales) OVER (...)

Best Practices

- **Always use ORDER BY for consistent results.**
- **Use PARTITION BY to reset calculations for groups.**
- **Use filters (CTEs/subqueries) to isolate top-N or ranges.**
- **Prefer RANK()/DENSE_RANK() for tie-sensitive rankings.**

Summary

- **Window functions let you do ranking, comparisons, running totals without grouping rows.**
- **Key categories:**
- **Ranking: ROW_NUMBER(), RANK(), DENSE_RANK(), NTILE()**
- **Value: LEAD(), LAG(), FIRST_VALUE(), etc. Aggregate: SUM() OVER, AVG() OVER, etc. Statistical: PERCENT_RANK(), CUME_DIST()**

Chapter 31: Views, Indexes, Stored Procedures, and Triggers in SQL

31.1 Introduction

When databases grow in size and complexity, managing them efficiently becomes crucial. SQL provides powerful features that go beyond basic data manipulation: Views for simplified and secure data access.

- **Indexes for speeding up searches.**
- **Stored Procedures to automate repetitive tasks.**
- **Triggers to automatically enforce rules and log changes.**
- **These features are essential for building robust, maintainable, and high-performance database systems.**

31.2 Views

31.2.1 What is a View?

- **A View is a virtual table created by a SELECT query.**
- **It doesn9t store actual data; it just displays the result of a stored query.**
- **You can query a view as if it were a table.**

Think of it as a saved SELECT statement you can use repeatedly.

Example:

```
CREATE VIEW EmployeeOverview AS
SELECT employee_id, name, department
FROM Employees;
```

You can now run:

```
SELECT * FROM EmployeeOverview;
```

31.2.2 Benefits of Views

- ✅ **Simplifies complex queries: Avoid writing joins repeatedly.**
- **Increases security: Hide sensitive data (like salary).**
- **Data abstraction: Users don9t need to know actual table structure.**
- **Reusability: Use the view in multiple queries or applications.**

31.3 Indexes

31.3.1 What is an Index?

- **An Index improves the speed of data retrieval operations.**
- **It works like a book index – helps the system locate rows faster.**
- **Used automatically by the SQL engine when available.**

31.3.2 Types of Indexes

Type	Description
Primary Key	Auto-created on primary key; ensures uniqueness.
Unique Index	Ensures no duplicate values in a column.

Composite Index	Created on two or more columns.
Full-Text Index	Used for fast searches in large text fields (MySQL, etc.)

31.3.3 How Indexes Work

- Indexes store column values in sorted form.
- When a query runs with WHERE, JOIN, or ORDER BY, the engine uses the index to find matches quickly.
- Without an index, a full table scan is required — much slower!

31.3.4 Creating Indexes

CREATE INDEX idx_employee_name
ON Employees(name);

Tip: Use indexes on columns used frequently in search or sorting, but don9to veruse (too many indexes slow down INSERT/UPDATE).

31.4 Stored Procedures

31.4.1 What is a Stored Procedure?

- A Stored Procedure is a named SQL program stored in the database. ⍰ It can include multiple SQL statements, conditions, and parameters.
- Great for automating logic like report generation, validations, or custom updates.

31.4.2 Benefits of Stored Procedures

Advantage	Description
⚡ Fast	Executes quickly as it's precompiled.
澝 Secure	Users can run it without seeing underlying tables.
澦 Reusable	Can be called with different inputs.
嬇 Maintainable	Logic is stored centrally in one place.

31.4.3 Creating a Stored Procedure

```
CREATE PROCEDURE GetEmployeesByDepartment(IN dept_name VARCHAR(50))
BEGIN
 SELECT employee_id, name, salary  FROM Employees
 WHERE department = dept_name;
```

31.4.4 Calling a Procedure

```
CALL GetEmployeesByDepartment('Marketing');
```

31.5 Triggers

31.5.1 What is a Trigger?

- **A Trigger is an automatic response to a database event (INSERT, UPDATE, DELETE).**
- **Runs a predefined block of code when conditions are met.**
- **Often used for auditing, validation, or automation.**

31.4.4 Calling a Procedure

CALL GetEmployeesByDepartment('Marketing');

31.5 Triggers

31.5.1 What is a Trigger?

- **A Trigger is an automatic response to a database event (INSERT, UPDATE, DELETE).**
- **Runs a predefined block of code when conditions are met.**
- **Often used for auditing, validation, or automation.**

31.5.2 When to Use Triggers

Use Case	Example
✍️🗃️ Audit Log	Save old values to a log table during an update.
❌ Prevent Invalid Entry	Block data entry based on complex logic.
🔄 Automatic Update	Update related rows in another table.

31.5.3 Creating a Trigger

```
CREATE TRIGGER LogSalaryChange
AFTER UPDATE ON Employees
FOR EACH ROW
BEGIN
 INSERT INTO SalaryAudit(employee_id, old_salary, new_salary, change_date)
 VALUES (OLD.employee_id, OLD.salary, NEW.salary, NOW());
END;
```

This trigger logs every salary change to a SalaryAudit table automatically.

31.6 Best Practices

Feature	Best Practice Tip
Views	Use for complex joins, sensitive data hiding, or reusable summaries.
Indexes	Index frequently filtered columns; avoid low-cardinality (many repeated values).
Stored Procedures	Centralize logic for validation or batch processes.

Triggers	**Keep logic simple and transparent; avoid overusing (can hide business logic).**

31.7 Summary

- ✅ **Views = virtual tables; great for abstraction and simplification.**
- ⚡ **Indexes = speed up search and sorting but should be used wisely.**
- **Stored Procedures = reusable, secure, and powerful SQL blocks.**
- **Triggers = automate logic based on table changes; useful for audits and validations.**

Chapter 32: Query Optimization and NULL Handling in SQL

32.1 Introduction

Writing SQL queries that work is one thing — writing SQL queries that work well and produce accurate results is a skill. This chapter focuses on:
Making queries faster and more efficient (Query Optimization) Understanding and handling missing or unknown data (NULL Handling)

32.2 What is Query Optimization?

- **Definition: Query optimization is the process of rewriting and tuning SQL statements to improve performance.**
- **Goal: To return results with the least amount of time and system resources.**

32.3.Why is Query Optimization Important?

✅ **Faster execution**
✅ **Better scalability**
✅ **Lower server load**
✅ **Cost-efficiency (especially on cloud services like AWS, Azure, etc.)**

32.4.How the SQL Query Optimizer Works

When you run a query, the Query Optimizer:
Parses the SQL. o Analyzes various execution paths (called execution plans).
Chooses the most efficient path based on:

- **Indexes & Joins**
- **Filters & Table sizes**

32.5 Key Techniques to Optimize Queries

32.5.1 Select Only What You Need

❌ **SELECT ***

✅ **SELECT employee_id, name 32.5.2 Use WHERE Clauses Effectively**
Filter early to reduce data processed.

```
-- Inefficient
SELECT * FROM Orders;
-- Efficient
SELECT * FROM Orders WHERE order_status = 'Completed';
```

32.5.3 Avoid Functions on Indexed Columns in WHERE

```
-- Prevents index use
WHERE UPPER(name) = 'JOHN'
-- Better
```

WHERE name = 'John' 32.5.4 Use JOINs Smartly

- **Avoid unnecessary tables.**

- Always define clear ON conditions.
- Prefer INNER JOIN over OUTER JOIN when possible.

32.6 Use EXPLAIN and ANALYZE Tools

These tools show how your query is interpreted by the DB engine.
EXPLAIN SELECT * FROM Orders WHERE customer_id = 1;
Gives info about:

- Index usage
- Join method & Estimated cost

32.7 Indexing for Optimization

- Indexes are like a table of contents for your database.
- Use them on frequently searched, sorted, or joined columns.

Types of Indexes:

- Single-column
- Composite
- Unique
- Full-text (for text search)

32.8 Other Optimization Tips

- Use LIMIT when not all rows are needed.
- Use EXISTS instead of IN for subqueries.
- Normalize data properly to avoid duplication.
- Use COVERING INDEXES to avoid table lookups.

32.9 What is NULL in SQL?

- NULL = "No value", "Unknown", or "Missing"
- It9s not 0, it9s not an empty string '', it means absence of a value.

32.10 Importance of Handling NULLs

NULLs affect:

- Comparisons Aggregations
- Joins
- Logic in WHERE and HAVING

Ignoring NULLs can lead to bugs and incorrect reports.

32.11 Working with NULL in SQL

32.11.1 Checking for NULL

Use IS NULL or IS NOT NULL
SELECT * FROM Customers WHERE phone IS NULL;

32.11.2 Replacing NULL with Default Values

Use COALESCE() or IFNULL():

```
SELECT name, COALESCE(phone, 'N/A') FROM Customers;
```

32.12 NULL and Comparison Logic

```
SELECT * FROM Students WHERE grade = NULL;   -- Wrong!
```

Always use:

```
WHERE grade IS NULL
```

NULL in any comparison returns UNKNOWN, which acts like FALSE.

32.13 NULL and Aggregate Functions

Function	Effect on NULL
COUNT(*)	Includes NULLs
COUNT(col)	Excludes NULLs
SUM(col)	Ignores NULLs
AVG(col)	Ignores NULLs

32.14 NULL in Joins

32.14.1 INNER JOIN

NULLs are excluded if they appear in join condition columns.

32.14.2 LEFT OUTER JOIN

NULLs appear when no match is found on the right side.

```
SELECT A.id, B.name
FROM A
LEFT JOIN B ON A.bid = B.id;
```

32.15 NULL and Indexing

- Some databases include NULLs in indexes; others ignore them.
- Always check your database documentation.

32.16 Advanced Query Optimization Topics

32.16.1 Materialized Views

Pre-computed result sets stored as tables.

Improve performance on complex queries.

32.16.2 Query Caching

- DB stores results for repeated queries.
- Avoids re-execution.

32.16.3 Partitioning

- Splits large tables into smaller parts.
- Speeds up queries on big datasets.

32.17 Real-Life Optimization Example

Task: Find top 5 best-selling products in the last 6 months

```
-- Poor version
SELECT product_id, SUM(quantity)
FROM Sales
GROUP BY product_id;
-- Optimized version
SELECT product_id, SUM(quantity) AS total_qty
FROM Sales
WHERE sale_date >= DATE_SUB(CURDATE(), INTERVAL 6 MONTH)
GROUP BY product_id
ORDER BY total_qty DESC
LIMIT 5;
```

Chapter 33: Final SQL Projects

Project 1: Retail Sales Management System

Schema and Code

```
CREATE TABLE Customers (
CustomerID INT PRIMARY KEY AUTO_INCREMENT,
Name VARCHAR(100) NOT NULL,
Email VARCHAR(100) UNIQUE,
Phone VARCHAR(15),
City VARCHAR(50) NOT NULL
);
CREATE TABLE Products (
ProductID INT PRIMARY KEY AUTO_INCREMENT,
ProductName VARCHAR(150) NOT NULL,
Price DECIMAL(10,2) NOT NULL CHECK (Price > 0),
Stock INT NOT NULL CHECK (Stock >= 0)
);
CREATE TABLE Suppliers (
SupplierID INT PRIMARY KEY AUTO_INCREMENT,
SupplierName VARCHAR(100) NOT NULL,
ContactEmail VARCHAR(100)
);
CREATE TABLE Employees (
EmployeeID INT PRIMARY KEY AUTO_INCREMENT,
Name VARCHAR(100) NOT NULL,
Position VARCHAR(50)
);
CREATE TABLE Orders (
OrderID INT PRIMARY KEY AUTO_INCREMENT,
CustomerID INT NOT NULL,
EmployeeID INT,
OrderDate DATETIME DEFAULT CURRENT_TIMESTAMP,
FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID),
FOREIGN KEY (EmployeeID) REFERENCES Employees(EmployeeID)
);
CREATE TABLE OrderDetails (
OrderDetailID INT PRIMARY KEY AUTO_INCREMENT,
OrderID INT NOT NULL,
ProductID INT NOT NULL,
Quantity INT NOT NULL CHECK (Quantity > 0),
FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),
FOREIGN KEY (ProductID) REFERENCES Products(ProductID)
```

```
);
Sample Data Insert
INSERT INTO Customers (Name, Email, Phone, City) VALUES ('Ravi
Sharma', 'ravi@example.com', '9876543210', 'Delhi'),
('Anita Singh', 'anita@example.com', NULL, 'Mumbai'),
('Rajesh Kumar', 'rajesh@example.com', '9123456789', 'Delhi');
INSERT INTO Products (ProductName, Price, Stock) VALUES
('Laptop', 50000, 10),
('Smartphone', 20000, 25),
('Headphones', 1500, 50);
INSERT INTO Employees (Name, Position) VALUES
('Suresh Patel', 'Sales Manager'),
('Neha Gupta', 'Sales Executive');
INSERT INTO Orders (CustomerID, EmployeeID) VALUES
(1, 1),
(2, 2);
INSERT INTO OrderDetails (OrderID, ProductID, Quantity) VALUES
    (1, 1, 1),
    (1, 3, 2), (2,
    2, 3);
Complex Queries & Explanation
Query 1: Total Sales per Product with Join and Aggregation
SELECT p.ProductName, SUM(od.Quantity) AS TotalSold,
    SUM(od.Quantity * p.Price) AS TotalRevenue
FROM OrderDetails od
JOIN Products p ON od.ProductID = p.ProductID
GROUP BY p.ProductName ORDER
BY TotalRevenue DESC;
```

What it does:

- Joins OrderDetails and Products to calculate total quantity sold and revenue per product.
- Key concepts: Join, aggregation (SUM), GROUP BY.

```
Query 2: Customers with More Than 1 Order
SELECT c.Name, COUNT(o.OrderID) AS NumberOfOrders
FROM Customers c
JOIN Orders o ON c.CustomerID = o.CustomerID
GROUP BY c.CustomerID, c.Name
HAVING COUNT(o.OrderID) > 1;
```

What it does:

- Lists customers who placed more than one order.
- Key concepts: HAVING clause to filter groups.

Stored Procedure: Add New Order

```
DELIMITER //
CREATE PROCEDURE AddOrder(
 IN p_CustomerID INT,
 IN p_EmployeeID INT,
 IN p_ProductID INT,
 IN p_Quantity INT
)
BEGIN
DECLARE v_OrderID INT;
DECLARE v_Stock INT;
START TRANSACTION;
-- Check stock availability
SELECT Stock INTO v_Stock FROM Products WHERE ProductID = p_ProductID FOR UPDATE;
 IF v_Stock < p_Quantity THEN
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Not enough stock available';
END IF;
 -- Insert new order
 INSERT INTO Orders (CustomerID, EmployeeID) VALUES (p_CustomerID, p_EmployeeID);
SET v_OrderID = LAST_INSERT_ID();
 -- Insert order detail
  INSERT INTO OrderDetails (OrderID, ProductID, Quantity) VALUES (v_OrderID, p_ProductID, p_Quantity);
 -- Update stock
UPDATE Products SET Stock = Stock - p_Quantity WHERE ProductID = p_ProductID;
    COMMIT;
END //
DELIMITER ;
```

Explanation:

This stored procedure inserts a new order only if there is enough stock. Uses transactions and FOR UPDATE lock to prevent race conditions.

Trigger: Stock Check on OrderDetail Insert

```
DELIMITER //
CREATE TRIGGER trg_CheckStockBeforeInsert
BEFORE INSERT ON OrderDetails
FOR EACH ROW
BEGIN
DECLARE v_Stock INT;
SELECT Stock INTO v_Stock FROM Products WHERE ProductID = NEW.ProductID;
```

```
 IF v_Stock < NEW.Quantity THEN
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Insufficient stock to place order';
END IF;
END //
DELIMITER ;
```

Explanation:

Ensures stock is sufficient before allowing an OrderDetails insert.

Optimization

Create index on foreign keys for faster joins:

```
CREATE INDEX idx_orders_customer ON Orders(CustomerID); CREATE INDEX idx_orderdetails_product ON OrderDetails(ProductID); 
```

Use EXPLAIN to analyze query plans.

Project 2: Hospital Appointment System

Schema

```
CREATE TABLE Patients (
 PatientID INT PRIMARY KEY AUTO_INCREMENT,
 Name VARCHAR(100) NOT NULL,
 DOB DATE,
 Phone VARCHAR(15),
 Email VARCHAR(100) UNIQUE
);
CREATE TABLE Doctors (
    DoctorID INT PRIMARY KEY AUTO_INCREMENT,
    Name VARCHAR(100) NOT NULL,
    Specialty VARCHAR(50)
);
CREATE TABLE Appointments (
    AppointmentID INT PRIMARY KEY AUTO_INCREMENT,
 PatientID INT,
    DoctorID INT,
    AppointmentDate DATETIME NOT NULL,
    Status ENUM('Scheduled', 'Completed', 'Cancelled') DEFAULT 'Scheduled',
    FOREIGN KEY (PatientID) REFERENCES Patients(PatientID),
 FOREIGN KEY (DoctorID) REFERENCES Doctors(DoctorID)
);
```

Sample Query: Doctor Availability Check

```
SELECT DoctorID
FROM Appointments
WHERE DoctorID = 1
AND AppointmentDate = '2025-06-10 10:00:00'
AND Status = 'Scheduled';
```

If this returns a row, doctor is busy.

Trigger: Prevent Double Booking

```
DELIMITER //
CREATE TRIGGER trg_PreventDoubleBooking
BEFORE INSERT ON Appointments
FOR EACH ROW
BEGIN
    IF EXISTS (
 SELECT 1 FROM Appointments
 WHERE DoctorID = NEW.DoctorID
 AND AppointmentDate = NEW.AppointmentDate
 AND Status = 'Scheduled'
 ) THEN
  SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Doctor already booked at this
                                              time';
 END IF;
END //
DELIMITER ;
```

View: Upcoming Appointments Per Doctor

```
CREATE VIEW UpcomingAppointments AS
SELECT d.Name AS DoctorName, p.Name AS PatientName, a.AppointmentDate
FROM Appointments a
JOIN Doctors d ON a.DoctorID = d.DoctorID
JOIN Patients p ON a.PatientID = p.PatientID
WHERE a.Status = 'Scheduled' AND a.AppointmentDate > NOW();
```

Project 3: Library Management

Schema

```
CREATE TABLE Members (
 MemberID INT PRIMARY KEY AUTO_INCREMENT,
 Name VARCHAR(100) NOT NULL,
 Email VARCHAR(100),
 Phone VARCHAR(15)
);
CREATE TABLE Books (
 BookID INT PRIMARY KEY AUTO_INCREMENT,
 Title VARCHAR(200) NOT NULL,
 Author VARCHAR(100),
 CopiesAvailable INT NOT NULL CHECK (CopiesAvailable >= 0)
);
CREATE TABLE Loans (
    LoanID INT PRIMARY KEY AUTO_INCREMENT,
 MemberID INT,
 BookID INT,
```

```
 LoanDate DATE DEFAULT CURDATE(),
 DueDate DATE,
 ReturnDate DATE,
 FOREIGN KEY (MemberID) REFERENCES Members(MemberID),
 FOREIGN KEY (BookID) REFERENCES Books(BookID)
);
```

Stored Procedure: Issue Book

```
DELIMITER //
CREATE PROCEDURE IssueBook(
 IN p_MemberID INT,
 IN p_BookID INT
)
BEGIN
DECLARE v_Copies INT;
 SELECT CopiesAvailable INTO v_Copies FROM Books WHERE BookID = p_BookID FOR
UPDATE;
 IF v_Copies <= 0 THEN
 SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'No copies available';
ELSE
 INSERT INTO Loans (MemberID, BookID, DueDate)
 VALUES (p_MemberID, p_BookID, DATE_ADD(CURDATE(), INTERVAL 14 DAY));
 UPDATE Books SET CopiesAvailable = CopiesAvailable - 1 WHERE BookID = p_BookID;
END IF;
END //
DELIMITER ;
```

Trigger: Return Book (Increase CopiesAvailable)

```
DELIMITER //
CREATE TRIGGER trg_AfterReturn
AFTER UPDATE ON Loans
FOR EACH ROW
BEGIN
 IF OLD.ReturnDate IS NULL AND NEW.ReturnDate IS NOT NULL THEN
    UPDATE Books SET CopiesAvailable = CopiesAvailable + 1 WHERE BookID =
    NEW.BookID;
END IF;
END //
DELIMITER ;
```

Explanation: When a book is returned (ReturnDate updated from NULL to a date), increase available copies.

Project 4: Online Course Enrollment

Schema

```
CREATE TABLE Students (
 StudentID INT PRIMARY KEY AUTO_INCREMENT,
 Name VARCHAR(100) NOT NULL,
 Email VARCHAR(100) UNIQUE NOT NULL
);
CREATE TABLE Courses (
 CourseID INT PRIMARY KEY AUTO_INCREMENT,
 Title VARCHAR(150) NOT NULL,
 Description TEXT
);
CREATE TABLE Enrollments (
 EnrollmentID INT PRIMARY KEY AUTO_INCREMENT,
 StudentID INT,
 CourseID INT,
 EnrollmentDate DATE DEFAULT CURDATE(),
 FOREIGN KEY (StudentID) REFERENCES Students(StudentID),
 FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)
);
```

Query: List Students Enrolled in a Course

```
SELECT s.Name, s.Email, c.Title, e.EnrollmentDate
FROM Enrollments e
JOIN Students s ON e.StudentID = s.StudentID
JOIN Courses c ON e.CourseID = c.CourseID
WHERE c.CourseID = 2;
```

View: Course Enrollment Counts

```
CREATE VIEW CourseEnrollmentCounts AS
SELECT c.CourseID, c.Title, COUNT(e.StudentID) AS NumberOfStudents
FROM Courses c
LEFT JOIN Enrollments e ON c.CourseID = e.CourseID
GROUP BY c.CourseID, c.Title;
```

Project 5: Employee Attendance System

Schema

```
CREATE TABLE Employees (
EmployeeID INT PRIMARY KEY AUTO_INCREMENT,
Name VARCHAR(100) NOT NULL,
Department VARCHAR(50)
);
CREATE TABLE Attendance (
 AttendanceID INT PRIMARY KEY AUTO_INCREMENT,
 EmployeeID INT,
 AttendanceDate DATE,
```

```
 Status ENUM('Present', 'Absent', 'Leave') DEFAULT 'Absent',
 FOREIGN KEY (EmployeeID) REFERENCES Employees(EmployeeID),
 UNIQUE(EmployeeID, AttendanceDate)
);
```

Query: Calculate Attendance Percentage per Employee

```
SELECT e.Name,
 COUNT(CASE WHEN a.Status = 'Present' THEN 1 END) * 100.0 / COUNT(*) AS AttendancePercentage
FROM Employees e
JOIN Attendance a ON e.EmployeeID = a.EmployeeID
GROUP BY e.EmployeeID, e.Name;
```

Trigger: Prevent Duplicate Attendance Entry

```
DELIMITER //
CREATE TRIGGER trg_PreventDuplicateAttendance
BEFORE INSERT ON Attendance
FOR EACH ROW
BEGIN
IF EXISTS (
SELECT 1 FROM Attendance WHERE EmployeeID = NEW.EmployeeID AND AttendanceDate = NEW.AttendanceDate
  ) THEN
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Attendance for this date already recorded';
END IF;
END // DELIMITER
;
```

Summary

- Each project uses primary keys, foreign keys, constraints, and indexes for data integrity and performance.
- We use complex JOINs and aggregations (GROUP BY, HAVING) for analysis.
- Transactions, stored procedures, and triggers automate and protect business logic. Views simplify reporting.
- We handle NULLs and special cases carefully (e.g., COALESCE can be used to display default values).
- Use EXPLAIN to analyze and optimize queries.

Chapter 34: Python Installation and IDE Setup

Introduction

Before you can begin writing Python code, you need to prepare your computer with the proper tools. This includes installing Python itself and setting up an environment where you can write and test your code easily. This environment is known as an IDE, which stands for Integrated Development Environment.

In this chapter, you will learn:

- **How to download and install Python on different operating systems**
- **How to verify your installation**
- **What an IDE is and how to set one up**
- **How to write and run your first Python program**

Let9s begin your Python journey!

34.1 What is Python?

Python is a widely used programming language known for being simple, powerful, and easy to learn. It is used in many areas such as:

- **Web development**
- **Data analysis**
- **Artificial Intelligence**
- **Machine Learning**
- **Game development**
- **Automation and scripting**

Python is free and open-source, which means anyone can download and use it without paying.

34.2 What is an IDE?

An IDE (Integrated Development Environment) is a software application that helps you write code efficiently. It usually includes:

- **A code editor**
- **A terminal or console**
- **A debugger**
- **Features like syntax highlighting and auto-complete**

Examples of popular Python IDEs:

- **IDLE (comes with Python)**
- **VS Code (Visual Studio Code)**
- **PyCharm**
- **Thonny**
- **Jupyter Notebook (especially for data science)**

You can also write Python code in a simple text editor like Notepad or Notepad++, but using an IDE makes things easier.

34.3 Downloading Python

Step 1: Go to the Official Python Website

Open your web browser and go to:

https://www.python.org

Step 2: Download Python

On the homepage, you9ll see a yellow button like this:

<Download Python 3.x.x=

(The x.x will show the latest version like 3.12.3)

Click the button to start downloading the Python installer for your system.

✅ Always download Python 3, not Python 2 (which is outdated).

34.4 Installing Python on Windows

Step-by-step:

1. Run the Installer
   - o Double-click the downloaded .exe file.
2. IMPORTANT: Check this box before clicking Install
   - ✅ <Add Python to PATH=

This allows you to run Python from the command line easily.

3. Click <Install Now=
   - o The installer will begin installing Python.
4. Wait for installation to complete
   - o Once done, you9ll see a message that says <Setup was successful=.

34.5 Verifying Python Installation on Windows

To make sure Python was installed correctly:

1. Open the Command Prompt (search for cmd in the start menu).
2. Type the following command:
3. python --version
4. If Python is installed, you9ll see the version number like:
5. Python 3.12.3

If this works, your installation was successful!

34.6 Installing Python on macOS

1. Go to https://www.python.org
2. Download the .pkg file for macOS.
3. Open the downloaded file and follow the installation instructions.
4. After installation, open the Terminal and type:
5. python3 --version
6. You should see the installed version of Python.

On macOS, you must use python3 instead of python.

34.7 Installing Python on Linux

Python often comes pre-installed on Linux, but you can update or install it manually.
For Ubuntu/Debian:

1. **Open Terminal.**
2. **Run these commands:**
3. **sudo apt update**
4. **sudo apt install python3**
5. **Check the version:**
6. **python3 --version**

You can also install pip (Python9s package manager) by running:
sudo apt install python3-pip

34.8.Choosing an IDE for Python

You can choose from many IDEs depending on your needs.

IDE	Best For	Notes
IDLE	Beginners	Comes with Python
Thonny	Kids & new learners	Simple and clean interface
VS Code	All levels	Lightweight, powerful
PyCharm	Professionals	Has a free and paid version
Jupyter Notebook	Data Science	Runs in a browser, great for visualization

34.9.Installing Visual Studio Code (VS Code)

VS Code is one of the most popular editors for Python.
Steps:

1. **Download it from:**
   https://code.visualstudio.com
2. **Install it on your computer.**
3. **Open VS Code → Go to Extensions (left sidebar) → Search for <Python= → Click Install**
4. **Now create a file with .py extension and start writing your Python code!**

34.10 Running Your First Python Program

You are now ready to write your first Python program.
Using IDLE:

1. **Open IDLE from your start menu.**
2. **Click File → New File**
3. **Type:**
4. **print("Hello, world!")**

5. **Save the file with the name hello.py**
6. **Press F5 to run the code.**

Output:

Hello, world!

Using VS Code:

1. **Open VS Code.**
2. **Create a new file hello.py.**
3. **Write:**
4. **print("Hello, world!")**
5. **Right-click → Click Run Python File in Terminal.**

34.11 Running Python from the Command Line

1. **Open Command Prompt or Terminal.**
2. **Type:**
3. **python**

or

python3

4. **You9ll enter the Python shell:**
5. **>>> print("Hello")**
6. **Hello**

To exit the shell:

exit()

34.12 Common Installation Problems and Fixes

Problem	Reason	Solution
'python' is not recognized	Python not added to PATH	Reinstall and check the "Add to PATH" box
Can't open IDLE	Corrupted install	Uninstall and reinstall Python
VS Code doesn't run Python	Python extension missing	Install the Python extension
Wrong version shown	Conflict with old version	Use python3 instead of python

Summary

In this chapter, you learned:

- **How to install Python on Windows, macOS, and Linux**
- **How to verify your installation**
- **What IDEs are and how to choose one**
- **How to install and use IDLE and VS Code**
- **How to write and run your first Python program**

With everything set up, you are now ready to begin writing real Python code. In the next chapter, we will explore Python syntax and learn how to write programs step by step.

Chapter 35: Data Types and Variables

Introduction

Understanding data types and variables is one of the most important steps in learning any programming language. In Python, data types define what kind of value a variable can hold, such as numbers, text, or lists.

This chapter covers:

- **What variables are and how to use them**
- **Different data types in Python**
- **Common string functions with examples**
- **Types of comments used in Python**

35.1 What is a Variable?

A variable is like a container that holds a value. You can give the container any name and store a value in it.

Example: name = "John" age = 25

- **name is a variable that stores the text "John"**
- **age is a variable that stores the number 25**
- **Rules for Naming Variables:**
- **Use letters, numbers, and underscores (_)**
- **Cannot start with a number**
- **Cannot use Python keywords (like if, while, def)**

35.2 Data Types in Python

Python has different types of data. Let9s understand the most common ones:

Type	Example	Description
int	10	Integer (whole number)
float	3.14	Decimal number
str	"Hello"	String (text)
bool	True / False	Boolean (true or false values)
list	[1, 2, 3]	List of items
tuple	(1, 2)	Unchangeable list
dict	{"name": "Alice"}	Dictionary (key-value pairs)
set	{1, 2, 3}	Unordered collection with no duplicates

35.3 Type Checking and Conversion

Check the type of a value:

```
 = 100
print(type(x))      #  Output:  <class  'int'>
```

Convert between types:

```
str_num = "123"
num = int(str_num)   # Now num = 123 as integer
```

35.4 Strings in Python

A string is a collection of characters written inside quotes.

Example:

```
greeting = "Hello, World!"
```

Python provides many built-in string functions that help work with text.

35.5 Common String Functions

Let9s explore the most useful string functions:

1. startswith()

Checks if the string begins with a specific word. text = "Python is fun" print(text.startswith("Python")) # True

2. endswith()

Checks if the string ends with a specific word. text = "Learn Python" print(text.endswith("Python")) # True

3. find()

Returns the index where a word first appears.

ext = "Hello World" print(text.find("World")) # 6

Returns -1 if not found.

4. replace()

Replaces part of the string with another.

text = "I like Java" new_text = text.replace("Java", "Python") print(new_text) # I like Python

5. upper() and lower() Converts string to uppercase or lowercase.

s = "Hello" print(s.upper()) # HELLO

print(s.lower()) # hello

6. strip()

Removes whitespace from beginning and end. s = " Hello " print(s.strip()) # "Hello"

7. split()

Breaks the string into a list. s = "a,b,c"

print(s.split(",")) # ['a', 'b', 'c']

8. join()

Joins items of a list into a string. items = ['a', 'b', 'c']
print(",".join(items)) # a,b,c

9. enumerate()
   Returns index and item in a loop. for i, ch in enumerate("abc"):
      print(i, ch) # Output:
   # 0 a # 1 b # 2 c

10. isalpha()
Checks if all characters are letters. name = "Hello" print(name.isalpha()) # True

11. isdigit()
   Checks if the string is only digits. num = "1234"
   print(num.isdigit()) # True

   12.count()
Counts how many times a word appears. text = "apple apple banana" print(text.count("apple")) # 2

13.capitalize()
Capitalizes the first letter. s = "python"
print(s.capitalize()) # Python

14.title()
   Capitalizes the first letter of every word. s = "learn python easily" print(s.title()) # Learn Python Easily

15.len()
   Returns the length of a string.
   s = "hello" print(len(s)) # 5

35.6 Mutable and Immutable Data Types

Type	Mutable?
int, float, str	❌ Immutable
list, dict, set	✅ Mutable

- Mutable: Can change the value (like list)
- Immutable: Value cannot be changed once created (like string)

35.7 Commenting in Python

Comments are used to explain code and make it more readable. Python ignores comments when running the program.

1. Single-line Comment

Use the # symbol.

```
# This is a single-line comment x
= 10  # This sets x to 10 2. Multi-
line Comment (using #)
You can write multiple # lines:
# This is a multi-line
# comment using hash symbols
print("Hello")
```

3. Multi-line Comment (using triple quotes)

Triple quotes (''' or """) can also be used for multi-line comments, although it's mostly used for docstrings.

```
'''
This is another way to write
multi-line comments
'''
print("Python")
```

Best Practice:

- **Use single-line comments for short explanations.**
- **Use multi-line or docstring comments to explain complex logic or functions.**

Summary

- **In this chapter, you learned:**
- **What variables are and how to use them**
- **The basic data types in Python: int, float, str, bool, list, tuple, dict, set**
- **String functions such as startswith(), endswith(), find(), replace(), upper(), enumerate(), and more**
- **How to convert between types**
- **How to write comments in Python**

Chapter 36: Basic Operators and Input / Output

Introduction

In any programming language, operators are special symbols used to perform operations on variables and values. They are like tools that help us do calculations, comparisons, logic checks, and more.

This chapter also introduces input and output—how to get data from the user and how to show results on the screen.

36.1 What Are Operators?

Operators perform actions on values or variables.

For example:

```
a = 10 b = 5 print(a + b)    # Output: 15
```

(Addition operator) Python has 7 types of basic operators:

36.2 Types of Operators in Python

No.	Operator Type	Description
1	Arithmetic Operators	Do math operations
2	Assignment Operators	Assign values to variables
3	Comparison Operators	Compare two values
4	Logical Operators	Combine conditions (True/False)
5	Bitwise Operators	Work with binary numbers
6	Membership Operators	Check if a value exists in a list, string, etc.
7	Identity Operators	Compare object identity

36.2.1 Arithmetic Operators

Used for basic math operations.

Operator	Description	Example	Result
+	Addition	5 + 3	8
-	Subtraction	10 - 4	6
*	Multiplication	2 * 3	6
/	Division	10 / 2	5.0
//	Floor Division	7 // 2	3
%	Modulus	10 % 3	1

**	Exponent	2 ** 3	8

36.2.1 Arithmetic Operators

Used for basic math operations.

Operator	Description	Example	Result
+	Addition	5 + 3	8
-	Subtraction	10 - 4	6
*	Multiplication	2 * 3	6
/	Division	10 / 2	5.0
//	Floor Division	7 // 2	3
%	Modulus	10 % 3	1
**	Exponent	2 ** 3	8

36.2.2 Assignment Operators

Used to assign values to variables.

Operator	Example	Meaning
=	x = 5	Assign 5 to x
+=	x += 3	x = x + 3
-=	x -= 2	x = x - 2
*=	x *= 4	x = x * 4
/=	x /= 2	x = x / 2
//=	x //= 3	x = x // 3
%=	x %= 2	x = x % 2
**=	x **= 2	x = x ** 2

36.2.3 Comparison Operators

Used to compare values. Result is either True or False.

Operator	Example	Result
==	5 == 5	True
!=	4 != 3	True
>	7 > 3	True
<	2 < 5	True
>=	4 >= 4	True
<=	6 <= 3	False

36.2.4 Logical Operators

Used to combine multiple conditions.

Operator	Description	Example	Result
and	True if both are True	5 > 3 and 4 > 2	True
or	True if at least one is True	5 > 3 or 4 < 2	True
not	Reverses the result	not(5 > 3)	False

36.2.5 Bitwise Operators

Works on binary (bit) level.

Operator	Meaning	Example (x=4, y=5)	Result
&	AND	x & y = 4	0100 & 0101 = 0100 (4)
`	`	OR	`x
^	XOR	x ^ y = 1	0100 ^ 0101 = 0001 (1)
~	NOT	~x = -5	~0100 = -0101
<<	Left Shift	x << 1 = 8	0100 << 1 = 1000
>>	Right Shift	x >> 1 = 2	0100 >> 1 = 0010

36.2.6 Membership Operators

Checks if a value exists in a sequence (like list, string, etc.)

Operator	Example	Result
in	'a' in 'apple'	True
not in	2 not in [1,3]	True

36.2.7 Identity Operators

Checks if two variables point to the same memory location.

Operator	Example	Result
is	x is y	True if x and y refer to same object
is not	x is not y	True if x and y are different objects

36.3 Operator Precedence in Python

When multiple operators are used in a single expression, Python follows a specific order to evaluate them. This is called operator precedence.
Precedence Table (High to Low):

Precedence	Operators
1 (Highest)	() (parentheses)
2	** (exponent)
3	+x, -x, ~x (unary operations)
4	*, /, //, %
5	+, -
6	<<, >>
7	&
8	^
9	`
10	Comparison: ==, !=, >, <, >=, <=
11	not
12	and
13	or
14 (Lowest)	=, +=, -= ... (assignment operators)

Example:
result = 2 + 3 * 4 # 3*4 is done first = 12; 2+12 = 14
Use parentheses () to control the order of evaluation.
result = (2 + 3) * 4 # Now, 2+3 = 5; 5*4 = 20
36.4 Input in Python
Use the input() function to get data from the user.
Example: name = input("Enter your name: ") print("Hello,", name)
) Input is always received as a string. If you want numbers, you must convert:
age = int(input("Enter your age: ")) print("Next year you will be", age + 1)

36.5 Output in Python

Use the print() function to display output on the screen.
Example:
print("Welcome to Python!")
You can print multiple values:
name = "Alice" age = 30 print(name, "is", age, "years old.") You can also use f-

strings: print(f"{name} is {age} years old.")

36.6 Summary

In this chapter, you learned:

The 7 types of Python operators:

- **Arithmetic**
- **Assignment**
- **Comparison**
- **Logical**
- **Bitwise**
- **Membership**
- **Identity**
- **How to use each operator with examples**
- **Operator precedence rules and how Python evaluates expressions**
- **How to use input() to get user data**
- **How to use print() to show results**

Chapter 37: String Handling and List/Tuple Basics

Introduction

In any programming language, data like names, words, sentences, or even numbers stored as text are called strings. In Python, strings, lists, and tuples are basic data structures used to store and manipulate data.

In this chapter, you will learn:

- How strings work in Python
- All slicing techniques
- All important string methods
- Different ways to format strings
- The difference between lists and tuples
- Basic operations on lists and tuples

37.1 Understanding Strings in Python

A string is a collection of characters, enclosed in quotes:

text1 = 'Hello' text2 = "Python" text3 = '''This is a multi-
line string'''

37.2 String Indexing and Slicing

Python treats strings like sequences, meaning each character has a position (index). Positive Indexing

word = "Python"

Index: 0 1 2 3 4 5

print(word[0]) # P

print(word[3]) # h

Negative Indexing

Index: -6 -5 -4 -3 -2 -1

print(word[-1]) # n

print(word[-3]) # h

Slicing Strings

Slicing allows you to extract a portion of a string.

Syntax: string[start:stop:step]

Part	Meaning
start	Starting index (included)
stop	Ending index (excluded)
step	Interval of characters

Examples:

text = "Programming" print(text[0:5]) # Progr print(text[:6]) # Progra (start from 0) print(text[6:]) # mming (till

end) print(text[::2]) # Pormig (every 2nd char)
print(text[::-1]) # gnimmargorP (reverse string)

37.3 String Manipulation Functions

Here are the most commonly used string functions in Python:

Method	Description	Example
len()	Returns length of string	len("hello") → 5
lower()	Converts to lowercase	"HELLO".lower() → hello
upper()	Converts to uppercase	"hello".upper() → HELLO
capitalize()	Capitalizes first character	"python".capitalize() → Python
title()	Capitalizes every word	"hello world".title() → Hello World
strip()	Removes spaces from both sides	" hello ".strip() → "hello"
lstrip()	Removes spaces from left	" hello".lstrip() → "hello"
rstrip()	Removes spaces from right	"hello ".rstrip() → "hello"
replace(a, b)	Replace a with b	"apple".replace("a","o") → opple
split(sep)	Splits string into list by separator	"a,b,c".split(",") → ['a', 'b', 'c']
join()	Joins list into string	" ".join(["Hello", "Python"]) → Hello Python
find(sub)	Finds position of first occurrence	"banana".find("a") → 1
rfind(sub)	Finds last occurrence of sub	"banana".rfind("a") → 5
count(sub)	Counts number of occurrences of substring	"banana".count("a") → 3
startswith(sub)	Returns True if string starts with sub	"hello".startswith("he") → True
endswith(sub)	Returns True if string ends with sub	"hello".endswith("lo") → True
isalpha()	True if all chars are alphabets	"abc".isalpha() → True
isdigit()	True if all chars are digits	"123".isdigit() → True
isalnum()	True if all chars are alphanumeric	"abc123".isalnum() → True
isspace()	True if all are spaces	" ".isspace() → True

swapcase()	Switch case of each letter	"AbC".swapcase() → aBc

37.4 String Formatting in Python

Formatting allows you to insert variables into strings.

1. Using f-strings (Python 3.6+) name = "Alice" age = 25 print(f"My name is {name} and I am {age} years old.")
2. Using format() method

print("My name is {} and I am {} years old.".format("Alice", 25)) 3. Using % formatting (old style)

name = "Bob" age = 30 print("My name is %s and I am %d years old." % (name, age))

37.5 Lists in Python

A list stores multiple values in a single variable. Lists are mutable, meaning their values can be changed.

Example:

```
colors = ["red", "green", "blue"]
print(colors[0]) # red
colors[1] = "yellow" # Change green to yellow
```

37.5.1 List Methods

Method	Description	Example
append()	Add item at end	colors.append("black")
insert()	Insert at specific index	colors.insert(1, "white")
remove()	Remove first matching item	colors.remove("red")
pop()	Remove last or indexed item	colors.pop() or colors.pop(0)
sort()	Sort list in ascending order	colors.sort()
reverse()	Reverse list order	colors.reverse()
clear()	Remove all items	colors.clear()
index()	Find index of item	colors.index("blue")
count()	Count number of times item appears	colors.count("red")

37.5.2 List Slicing nums = [10, 20, 30, 40, 50]

```
print(nums[1:4])        # [20, 30, 40]
print(nums[:3])         # [10, 20, 30]
print(nums[::2])  # [10, 30, 50]
```

37.6 Tuples in Python

A tuple is similar to a list but immutable (cannot be changed). Tuples are written using round brackets ().

Example:

```
person = ("John", 25, "Male")
print(person[0])  # John
```

37.6.1 Tuple Methods

Method	Description	Example
count()	Count how many times value appears	person.count("John")
index()	Return index of first match	person.index("Male")

37.6.2 Tuple Unpacking

You can assign values from a tuple directly to variables:

```
name, age, gender = person
print(name)  # John
```

37.6.3 Why Use Tuples Instead of Lists?

- Tuples use less memory (faster)
- Tuples are safer (can9t be changed by mistake)
- Useful for fixed data (like days of week)

37.7 Difference Between List and Tuple

Feature	List ([])	Tuple (())
Mutable	Yes	No
Speed	Slower	Faster
Memory	Uses more memory	Uses less memory
Use Case	Dynamic data	Fixed data

Summary

In this chapter, you learned:

- What strings are and how to use them
- How to access, slice, and manipulate strings
- All major string functions and formatting styles
- The basics of lists and how to modify them
- Tuple basics and when to use them The difference between lists and tuples

Chapter 39: Functions and Lambda Expressions

Introduction

Functions are reusable blocks of code designed to perform a particular task. They improve code reusability, structure, and readability. In Python, you can define named functions using the def keyword or anonymous functions using lambda. You can also pass varying numbers of arguments using *args and **kwargs, and even define functions within functions (nested functions).

In this chapter, we will explore all these topics in depth, including:

- Regular and default functions
- *args and **kwargs
- Difference between return and print
- Nested functions
- Lambda and nested lambda functions

39.1 What is a Function?

A function is a named block of code that is designed to do one specific job. When you want to perform that job, you call the function by its name.

39.1.1 Defining and Calling Functions

```
def greet():
 print("Hello, welcome to Python!") greet()  # Output: Hello, welcome to Python!
```

39.1.2 Function with Parameters and Return Values

```
def add(a, b): return a + b
result = add(5, 3) print(result)  # Output: 8
```

39.2 Function Arguments

39.2.1 Default Arguments def greet(name="Guest"):

```
 print(f"Hello, {name}!")
greet()        # Output: Hello, Guest! greet("Alice")  # Output: Hello, Alice!
```

39.2.2 Keyword Arguments def student_info(name, age): print(f"Name: {name}, Age: {age}")

```
    student_info(age=20, name="John")
```

39.3 *args and **kwargs

39.3.1 *args: Non-keyword variable-length arguments

*args allows you to pass any number of non-keyword arguments to a function. def add_all(*args): total = 0 for num in args: total += num return total

```
print(add_all(1, 2, 3, 4))  # Output: 10
```

39.3.2 **kwargs: Keyword variable-length arguments

kwargs lets you pass any number of keyword arguments (like dictionaries). def print_details(kwargs):

```
  for key, value in kwargs.items(): print(f"{key}: {value}")
```

print_details(name="Alice", age=25, city="Delhi")

39.4 return vs print

39.4.1 print()

- Used to display output to the console.
- Does not return anything.

def add(a, b):
print(a + b) x = add(2, 3) # Output: 5
print(x) # Output: None

39.4.2 return

- Used to send data back to the calling environment.
- Value can be stored or passed further. def add(a, b):

return a + b x = add(2, 3) print(x) # Output: 5 Key Difference:

- Use print() for displaying output.
- Use return when you want to reuse the output further in the program.

39.5 Nested Functions

A nested function is a function defined inside another function. Useful for encapsulating helper logic.

39.5.1 Example 1: Basic Nested Function

def outer(): print("Inside outer function") def inner():
print("Inside inner function") inner()
outer()

39.5.2 Example 2: Returning Inner Function

def outer(): def inner():
return "Hello from inner" return inner
func = outer() print(func()) # Output: Hello from inner

39.6 Lambda Functions

Lambda functions are anonymous functions that can take any number of arguments but have only one expression.

39.6.1 Syntax lambda arguments: expression

39.6.2 Example 1: Simple Lambda

square = lambda x: x * x print(square(4)) # Output: 16

39.6.3 Example 2: Multiple Arguments

add = lambda a, b: a + b print(add(10, 5)) # Output: 15

39.6.4 Example 3: Use in Sorting

items = [("apple", 2), ("banana", 1), ("cherry", 3)] items.sort(key=lambda item: item[1])
print(items) # Output: [('banana', 1), ('apple', 2), ('cherry', 3)]

39.7 Nested Lambda Functions

You can have a lambda function that returns another lambda. 39.7.1 Example 1: Basic Nested Lambda

```
nested = lambda x: (lambda y: x + y)
add_five = nested(5)
print(add_five(3))  # Output: 8
```

39.7.2 Example 2: Nested with Immediate Call

```
print((lambda x: (lambda y: x * y))(4)(5))  # Output: 20
```

These are compact and powerful for certain functional programming tasks.

Summary

- **In this chapter, we covered:**
- **How to define, call, and return values from functions**
- **Different types of function arguments including default, keyword, *args, and **kwargs**
- **The important distinction between print() and return**
- **Creating and using nested functions**
- **Writing lambda functions and even nesting them**

Chapter 40: Error Handling with try-except

Introduction

Errors are a part of programming, and they can occur for many reasons—such as incorrect input, missing files, or division by zero. If errors are not handled properly, they can cause a program to crash. Python provides a powerful mechanism to handle such situations gracefully using exception handling. This allows the program to catch and deal with errors during execution.

In this chapter, we will explore how to use try, except, else, and finally blocks to handle errors effectively. We will also look at raising custom exceptions, explaining each concept thoroughly with examples.

40.1 What is an Exception?

An exception is an error that occurs during program execution. Python stops executing the program when it encounters an error unless the error is properly handled. Exceptions are different from syntax errors because exceptions occur during execution, not during code writing or compilation.

40.1.1 Common Built-in Exceptions

Exception Name	Description
ZeroDivisionError	Raised when dividing a number by zero
ValueError	Raised when a function receives an argument of the correct type but an inappropriate value
TypeError	Raised when an operation is performed on the wrong data type
FileNotFoundError	Raised when a file or directory is requested but does not exist
IndexError	Raised when accessing an invalid index in a list or tuple
KeyError	Raised when accessing a dictionary with a non-existent key

Example of an unhandled exception:

```
print(10 / 0)  # This raises ZeroDivisionError
```

40.2 Basic try-except Structure

The try block lets you test a block of code for errors. The except block lets you handle the error. Syntax: try:

```
   # Code that might raise an error except:
   # Code that runs if there is an error Example: try:
   x = 10 / 0
except:  print("An error occurred!")
```

Output:

```
An error occurred!
```

Explanation:

- If the code inside the try block executes without errors, the except block is skipped.
- If an error occurs, the rest of the try block is skipped and the except block is executed.

40.3 Handling Specific Exceptions

- You can catch specific exceptions by naming them in the except block. This allows you to handle different error types differently.
- Example: try: num = int(input("Enter a number: "))
- print(100 / num)
- except ZeroDivisionError:
- print("You can't divide by zero!") except ValueError:
- print("Invalid input. Please enter a number.")

Why This is Better:

Handling specific exceptions helps avoid hiding unexpected bugs and makes the code more readable.

40.4 Using else Block

The else block runs only if the try block does not raise an exception.

Example:

```
try:
num = int(input("Enter a number: ")) except ValueError:      print("That was not a number!")
else:
   print(f"You entered: {num}")
```

Use Case:

Use else when you want to execute certain code only when no exception occurs.

40.5 Using finally Block

The finally block always runs, whether an error occurred or not. It is useful for cleanup operations like closing files, releasing resources, or displaying a final message.

Example:

```
try:
   file = open("example.txt", "r")     data = file.read()
except FileNotFoundError:
   print("File not found!")
finally:  print("Execution completed.")
```

Output:

```
File not found!
Execution completed.
```

40.6 Raising Exceptions with raise

Python allows you to raise your own exceptions using the raise keyword. This is helpful when you want to enforce specific constraints in your code.

Syntax:

```
raise ExceptionType("Error message")
```

Example:

```
def divide(a, b):  if b == 0:
      raise ZeroDivisionError("Cannot divide by zero!")  return a / b
try:
   print(divide(10, 0)) except ZeroDivisionError as e:
   print(e)
```

Output:

```
Cannot divide by zero!
```

40.7 Creating Custom Exceptions

Python allows you to create your own custom exception classes for application-specific errors. You do this by creating a new class that inherits from the built-in Exception class.

Syntax:

```
class CustomError(Exception):
   pass Example:
class NegativeNumberError(Exception):
   pass
def check_positive(number):   if number < 0:          raise
  NegativeNumberError("Negative number not allowed")
  return True
try:
check_positive(-5)          except
NegativeNumberError as e:
   print(e)
```

Output:

```
Negative number not allowed
```

Benefits:

- Helps in identifying and handling specific cases unique to your application.
- Improves code clarity and documentation.

Summary

In this chapter, we learned:

- What exceptions are and how they affect program execution
- How to use try, except, else, and finally blocks to manage exceptions
- The importance of handling specific exceptions for better control
- How to raise built-in and custom exceptions for special scenarios

Error handling is crucial for writing robust and user-friendly programs. By planning for possible failures, you make your code more reliable and easier to debug.

Chapter 41: Data Structures in Python

41.1 Lists

What is a List?

- A list is an ordered collection of elements enclosed in square brackets [ ].
- Lists are mutable, meaning you can change, add, or remove items after creation.
- Lists can contain mixed data types — numbers, strings, other lists, etc.

Example:

```
fruits = ["apple", "banana", "cherry"]
mixed = [1, "two", 3.0, [4, 5]]
```

Indexing in Lists

- Use an index inside square brackets to access elements.
- Indexing starts at 0 for the first element.
- Negative indices access elements from the end: -1 is the last item.

```
print(fruits[0]) # Output: apple print(fruits[-
1]) # Output: cherry
```

Slicing Lists

- Slicing extracts a portion of a list using start:stop:step.
- start is inclusive; stop is exclusive.
- Step indicates how many elements to skip.

```
print(fruits[1:3]) # ['banana', 'cherry'] (from index 1 up to 2)
print(fruits[:2]) # ['apple', 'banana'] (start to index 1)
print(fruits[::2]) # ['apple', 'cherry'] (every 2nd element)
print(fruits[::-1]) # ['cherry', 'banana', 'apple'] (reverse the list)
```

Adding Elements

- append(value) adds an element to the end.
- insert(index, value) adds an element at a specific position.
- extend(iterable) adds all elements from another list or iterable.
  fruits.append("orange") fruits.insert(1, "mango") fruits.extend(["kiwi", "pineapple"]) print(fruits)

Removing Elements

- remove(value) deletes the first occurrence of a value.
- pop(index) removes element at given index and returns it; defaults to last element.
- del list[index] deletes element at index.
- clear() empties the list.

```
fruits.remove("banana")
last = fruits.pop() del fruits[0] fruits.clear()
```

Searching and Counting

- index(value) returns the first index of a value.
- count(value) returns how many times a value occurs.

fruits = ["apple", "banana", "apple"] print(fruits.index("banana")) # 1 print(fruits.count("apple")) # 2

- Sorting and Reversing
- sort() sorts the list in ascending order (in-place).
- sorted(list) returns a new sorted list without modifying original. reverse() reverses the list in place.

numbers = [3, 1, 4, 2] numbers.sort() print(numbers) # [1, 2, 3, 4] rev = sorted(numbers, reverse=True) print(rev) # [4, 3, 2, 1] numbers.reverse() print(numbers) # [4, 3, 2, 1]

List Comprehensions (Efficient Way to Create Lists) squares = [x*x for x in range(5)] print(squares) # [0, 1, 4, 9, 16] Iterating Over Lists

for fruit in fruits:

41.2 Tuples

What is a Tuple?

- Tuples are ordered, like lists, but immutable — their values cannot be changed after creation.
- Tuples use parentheses ( ).
- Useful for fixed collections of items or as keys in dictionaries.

coordinates = (10.0, 20.0)

Indexing and Slicing Works the same as lists.

print(coordinates[0]) # 10.0 print(coordinates[:1]) # (10.0,)

Tuple Methods

- Only two methods: .index(value) and .count(value).

colors = ("red", "green", "blue", "red") print(colors.index("blue")) # 2 print(colors.count("red")) # 2 Tuple Unpacking

Assign elements to variables in one line. x, y = coordinates

print(x, y) # 10.0 20.0 Immutability Example coordinates[0] = 5 # Error! Tuples cannot be changed.

41.3 Sets

What is a Set?

- An unordered collection of unique elements.
- Use curly braces {} or set() constructor.
- Automatically removes duplicates.
- Useful for membership tests, eliminating duplicates, and mathematical operations.

numbers = {1, 2, 3, 3, 4} print(numbers) # {1, 2, 3, 4}

Adding and Removing Elements numbers.add(5) numbers.remove(2) # Raises error if not found numbers.discard(10) # No error if not found numbers.clear() # Empties the set

Set Operations

A = {1, 2, 3}

B = {3, 4, 5}
print(A.union(B)) # {1, 2, 3, 4, 5} print(A.intersection(B)) # {3}
print(A.difference(B)) # {1, 2} print(A.symmetric_difference(B)) # {1, 2, 4, 5}
Iterating Over Sets for item in A: print(item)

41.4 Dictionaries

What is a Dictionary?

- Collection of key-value pairs.
- Keys are unique and immutable types (strings, numbers, tuples).
- Values can be any type.
- Dictionaries use curly braces {} with colon : separating key and value.

student = {"name": "Alice", "age": 20, "grade": "A"} Accessing and Modifying Values print(student["name"]) # Alice
student["age"] = 21 # Update age student["city"] = "Delhi" # Add new key-value
Common Dictionary Methods
student.keys() # dict_keys(['name', 'age', 'grade', 'city']) student.values() # dict_values(['Alice', 21, 'A', 'Delhi']) student.items() # dict_items([('name', 'Alice'), ('age', 21), ('grade', 'A'), ('city', 'Delhi')] student.pop("grade") # Removes and returns value for 'grade' student.update({"age": 22}) # Updates multiple keys
Looping Through Dictionaries
for key, value in student.items():
print(f"{key}: {value}")
Nested Dictionaries Useful for hierarchical data. school = {
"class1": {"name": "John", "marks": 85},
"class2": {"name": "Jane", "marks": 92}
}
print(school["class1"]["name"]) # John
Summary & Tips:

- Use lists when order matters and you may need to change the data.
- Use tuples when you want a fixed collection, which shouldn9t change.
- Use sets to remove duplicates and perform mathematical operations like unions.
- Use dictionaries to associate keys with values for fast lookups.

Chapter 42: Iterating Through Nested Structures

Introduction

When working with Python, data is often organized in nested structures—data structures that contain other data structures inside them. This is common in real-world data like JSON responses, matrices, tabular data, or hierarchical data models. Iterating (looping) through these nested structures allows you to access and manipulate every element inside, no matter how deeply nested it is. This chapter explains the techniques and best practices to iterate nested lists, dictionaries, and combinations thereof.

42.1 Understanding Nested Structures

What is Nesting?

Nesting means placing one data structure inside another. This can be:

- List inside a list (List of lists)
- Dictionary containing lists or dictionaries
- List of dictionaries
- Mixed types combined in any depth

Example of nesting:

```
nested_list = [[1, 2], [3, 4, 5], [6]]
nested_dict = {
  "group1": {"name": "Alice", "scores": [85, 90]},
  "group2": {"name": "Bob", "scores": [78, 88]}
```

Here, nested_list is a list containing multiple lists; nested_dict is a dictionary containing dictionaries with lists.

42.2 Iterating Nested Lists

Basic Nested List Iteration

Given a list of lists (matrix), you typically use nested loops:

```
matrix = [
 [1, 2, 3],
 [4, 5, 6],
 [7, 8, 9]
for row in matrix:  # Outer loop: iterates over each list (row)
for element in row:  # Inner loop: iterates over each element in the row
 print(element, end=' ')  print()  #
New line after each row
```

Explanation:

- The outer for loop takes one row (which itself is a list) at a time.
- The inner for loop goes through each element inside that row.
- end=' ' prints elements on the same line separated by space.
- After inner loop completes for a row, print() moves to the next line.

42.3 Iterating Nested Dictionaries

Dictionary Containing Lists Example inventory = {
"fruits": ["apple", "banana", "mango"],
"vegetables": ["carrot", "potato"]
}
To access every item, use:
for category, items in inventory.items():
print(category + ":") for item in items:
print(f" - {item}")

Explanation:

- .items() returns key-value pairs.
- category is a key like 'fruits'.
- items is the corresponding list.

The inner loop prints each item in the list.
Dictionary with Nested Dictionary Example
students = {
"Alice": {"age": 20, "grades": [88, 92, 85]},
"Bob": {"age": 22, "grades": [75, 78, 80]}
}
To iterate:
for student, info in students.items():
print(f"Student: {student}") print(f"Age: {info['age']}") print("Grades:") for grade in info["grades"]:
print(f" {grade}")

42.4 Iterating List of Dictionaries

List of dictionaries is very common when working with tabular or JSON-like data.
people = [
{"name": "Alice", "city": "New York"},
{"name": "Bob", "city": "Paris"},
{"name": "Charlie", "city": "London"}
]
Iterate through each dictionary: for person in people: print(f"{person['name']} lives in {person['city']}.")

42.5 Handling Mixed Nested Structures

Nested data may contain several types:
data = [
{"name": "Alice", "scores": [85, 90, 92]},
{"name": "Bob", "scores": [70, 78, 75]}
]
To iterate and access deeply nested lists inside dictionaries: for entry in data:

```
 print(f"Name:
{entry['name']}")  for score in
entry["scores"]:
print(f"Score: {score}")
```

42.6 Recursive Iteration for Deeply Nested Structures

When you have arbitrary depth or unknown levels of nesting, recursion helps.

Recursive Function Example for Nested Lists

```
def   flatten_list(nested_list):
  for element in nested_list:
      if isinstance(element, list):
flatten_list(element)  # Recursion for sublist
      else:
        print(element)
complex_list    =    [1,    [2,    3],    [4,    [5,    6]],    7]
flatten_list(complex_list)
```

Output:

```
1
2
3
4
5
6
7
```

How recursion works here:

Function calls itself when it finds a list.

- If it finds a non-list, it prints the element.
- This continues until all nested elements are printed.

42.7 Using enumerate() in Nested Iterations

enumerate() is helpful to keep track of indices when looping:

```
matrix = [
   [10, 20],
   [30, 40]
]
for i, row in enumerate(matrix):
  for j, val in enumerate(row):
      print(f"matrix[{i}][{j}] = {val}")
```

Output:

```
matrix[0][0]        =        10
matrix[0][1]        =        20
```

matrix[1][0] = 30
matrix[1][1] = 40

42.8 List Comprehensions with Nested Loops

List comprehensions simplify nested iteration in concise syntax.
Flatten a Nested List
matrix = [[1, 2], [3, 4], [5, 6]] flat_list = [item for
sublist in matrix for item in sublist]
print(flat_list) # Output: [1, 2, 3, 4, 5, 6]
Note: The order of for loops inside list comprehension follows the same order as nested for loops.

42.9 Practical Examples

Example 1: Sum All Elements in Nested List
matrix = [
[1, 2, 3],
[4, 5],
[6, 7, 8, 9]

total_sum = 0 for
row in matrix:
for num in row:
total_sum +=
num
print(f"Total sum: {total_sum}") # Output: 45 Example 2: Count
Frequency of Words in Nested Lists words = [["apple", "banana"],
["apple", "orange"], ["banana", "apple"]]
freq = {} for sublist
in words: for word
in

sublist:
freq[word] = freq.get(word, 0) + 1 print(freq) #
Output: {'apple': 3, 'banana': 2, 'orange': 1}

42.10 Common Mistakes & Best Practices

- Avoid Reusing Loop Variables: Use meaningful variable names instead of generic i or j when nesting multiple loops.
- Check Types with isinstance(): When recursion is involved or mixed types exist, use isinstance() to handle the correct data type.

- **Be Mindful of Indentation: Nested loops and recursive functions require careful indentation to avoid syntax errors and improve readability.**
- **Use Generators for Large Data: When iterating huge nested structures, consider generators to save memory.**
- **Debug with Print Statements: Print intermediate results to understand nesting levels during iteration.**

Summary

- **Nested structures contain data structures inside other data structures.**
- **Iteration through nested lists, dictionaries, and mixed types requires nested loops.**
- **For unknown depth, recursion helps to explore all nested elements.**
- **enumerate() provides indices for nested loops.**
- **List comprehensions allow concise nested iterations.**
- **Practice with real-world examples strengthens understanding.**

Chapter 43: Introduction to Pandas and NumPy

43.1 Introduction

Python is widely used for data science, analysis, and numerical computing because of its powerful libraries. Two of the most important libraries in this space are NumPy and Pandas.

- NumPy (Numerical Python) is the foundational package for numerical computing in Python. It provides powerful n-dimensional arrays and functions for mathematical operations.
- Pandas builds on NumPy and offers data structures and tools designed for working with structured (tabular) data, like spreadsheets or databases.

This chapter covers both libraries9k ey concepts, structures, and complete methods/functions to handle data efficiently.

43.2 NumPy (Numerical Python)

43.2.1 What is NumPy?

- Provides support for large, multi-dimensional arrays and matrices.
- Offers mathematical functions for operations on these arrays.
- Enables vectorized operations (operations on entire arrays without explicit Python loops).

43.2.2 Creating NumPy Arrays

```
import numpy as np
# 1D array arr1 =
np.array([1, 2, 3, 4])
# 2D array
arr2 = np.array([[1, 2, 3], [4, 5, 6]])
```

43.2.3 Array Attributes

- .shape — returns the dimensions of the array
- dtype — data type of elements
- .size — total number of elements

```
print(arr2.shape) # (2, 3) print(arr2.dtype) # int64
(or int32 depending on system)
print(arr2.size) # 6
```

43.2.4 Array Creation Functions

Function	Description	Example
np.zeros(shape)	Creates array filled with zeros	np.zeros((2,3))
np.ones(shape)	Creates array filled with ones	np.ones((3,2))

np.arange(start, stop, step)	Creates array with evenly spaced values	np.arange(0, 10, 2)
np.linspace(start, stop, num)	Creates array of num evenly spaced points	np.linspace(0, 1, 5)
np.eye(n)	Creates identity matrix	np.eye(3)

43.2.5 Indexing and Slicing

Similar to Python lists, but supports multi-dimensional slicing.

```
arr = np.array([10, 20, 30, 40, 50])
print(arr[2])  # 30 print(arr[1:4])  # [20 30 40] arr2d = np.array([[1, 2, 3], [4, 5, 6]]) print(arr2d[1, 2])  # 6 print(arr2d[:, 1])  # Second column: [2
5]
```

43.2.6 Array Operations

- Element-wise operations:

```
a = np.array([1, 2, 3]) b = np.array([4, 5, 6]) print(a + b)  # [5 7 9] print(a * b)  # [4 10 18] print(a - b)  # [-3 -3 -3]
print(a / b)  # [0.25 0.4 0.5]
```

43.2.7 Useful NumPy Mathematical Functions

Function	Description	Example
np.sum()	Sum of elements	np.sum(arr)
np.mean()	Mean of elements	np.mean(arr)
np.median()	Median of elements	np.median(arr)
np.std()	Standard deviation	np.std(arr)
np.min()	Minimum value	np.min(arr)
np.max()	Maximum value	np.max(arr)
np.sqrt()	Square root of elements	np.sqrt(arr)
np.exp()	Exponential (e^x) of elements	np.exp(arr)

43.2.8 Reshaping and Combining Arrays

- Reshape arrays without changing data:

```
arr = np.arange(6)
print(arr.reshape(2, 3))
```

- Stack arrays vertically and horizontally:

```
a = np.array([1, 2, 3]) b = np.array([4, 5, 6]) print(np.vstack((a, b)))  # vertical stack
print(np.hstack((a, b)))  # horizontal stack
```

43.3 Pandas

43.3.1 What is Pandas?

- Built on top of NumPy.
- Provides two primary data structures:
- Series: 1D labeled array
- DataFrame: 2D labeled table, similar to spreadsheets or SQL tables.
- Designed for easy data cleaning, transformation, and analysis.

Like a 1D array but with labels (index).

43.3.2 Pandas Series import

```
pandas as pd
s = pd.Series([10, 20, 30], index=['a', 'b', 'c'])
print(s['b']) # 20
```

43.3.3 Pandas DataFrame

Two-dimensional, size-mutable, heterogeneous tabular data.

```
data = {'Name': ['John', 'Anna', 'Peter'],
     'Age': [28, 24, 35],
     'City': ['NY', 'Paris', 'London']}
df = pd.DataFrame(data)
print(df)
```

43.3.4 Reading and Writing Data

- Read data from various formats:

```
df = pd.read_csv('file.csv')
df =
pd.read_excel('file.xlsx')
df = pd.read_json('file.json')
```

- Write data to formats:

```
df.to_csv('output.csv', index=False)
df.to_excel('output.xlsx')
```

43.3.5 Accessing Data in DataFrame

- Access columns:

```
print(df['Name'])
```

- Access rows by position or label: print(df.iloc[0]) # First row (by position)

```
print(df.loc[0])  # Row with index label 0
```

- **Filter rows by condition:**

```
print(df[df['Age'] > 25])
```

43.3.6 Modifying DataFrames

Add new column:

```
df['Salary'] = [50000, 60000, 70000]
```

Update values:

```
df.loc[df['Name'] == 'Anna', 'Salary'] = 65000
```

Delete columns or rows:

```
df.drop('Salary', axis=1, inplace=True)  # Drop column
df.drop(0, axis=0, inplace=True)  # Drop row by index
```

43.3.7 Handling Missing Data

- **Check for missing values:**

```
df.isnull()
df.isnull().sum()
```

- **Fill missing values:** `df.fillna(0, inplace=True)`
- **Drop rows/columns with missing values:** `df.dropna(inplace=True)`

43.3.8 Pandas DataFrame Methods

Method	Description
.head(n)	Returns first n rows
.tail(n)	Returns last n rows
.info()	Summary of DataFrame
.describe()	Statistical summary of numerical data
.shape	Returns (rows, columns)
.columns	List of columns
.index	Index labels
.sort_values()	Sort DataFrame by column
.groupby()	Group data for aggregation
.apply()	Apply function along axis

43.3.9 Example: Grouping and Aggregation

```
df.groupby('City')['Age'].mean()
```

43.3.10 Merging and Joining DataFrames

Merge two DataFrames based on key columns:

```
pd.merge(df1, df2, on='ID')
```

- **Concatenate vertically or horizontally: pd.concat([df1, df2], axis=0) # Vertically pd.concat([df1, df2], axis=1) # Horizontally**

43.4 Summary

- **NumPy is best for numerical computations with homogeneous data using powerful multi-dimensional arrays.**
- **Pandas is best for structured, labeled, heterogeneous data for analysis and manipulation.**
- **Both libraries integrate seamlessly and are essential tools for data science and analytics.**

Chapter 44: Reading Data (CSV, Excel, JSON)

44.1 Introduction

In data analysis, reading data from external files is a crucial first step. The most common data file formats you will encounter are:

- CSV (Comma Separated Values)
- Excel spreadsheets (.xls, .xlsx)
- JSON (JavaScript Object Notation)

Python9s Pandas library provides powerful functions to read these file types easily, allowing you to convert the raw data into DataFrames for analysis and manipulation.

44.2 Reading CSV Files

44.2.1 What is CSV?

- CSV files store tabular data in plain text format.
- Each row is a line in the file.
- Columns are separated by commas (or other delimiters).

44.2.2 Basic CSV Reading

```
import pandas as pd df =
pd.read_csv('data.csv')
print(df.head())
```

- pd.read_csv() reads the CSV file into a DataFrame.
- By default, it assumes the first row contains column headers.

44.2.3 Important Parameters of read_csv()

Parameter	Description	Example
filepath_or_buffer	Path or URL of the CSV file	'data.csv'
sep	Delimiter/separator (default is comma)	sep=';' for semicolondelimited
header	Row number to use as column names (default 0)	header=None if no headers
names	List of column names to use	names=['A', 'B', 'C']
index_col	Column(s) to set as index	index_col=0
usecols	Columns to read	usecols=['A', 'C']
dtype	Data types for columns	dtype={'A': int, 'B': float}
skiprows	Rows to skip at the start	skiprows=2
na_values	Additional strings to recognize as NaN	na_values=['NA', 'missing']

parse_dates	Columns to parse as dates	parse_dates=['date_col']
nrows	Number of rows to read	nrows=100
encoding	File encoding	encoding='utf-8'

44.2.4 Example: Reading CSV with options

```
df = pd.read_csv('data.csv', sep=';', header=0, index_col='ID', usecols=['ID', 'Name', 'Date', 'Score'], parse_dates=['Date'], na_values=['NA', '--'])
```

44.2.5 Reading Large CSV Files

- Use chunksize to read the file in chunks (for big files).

```
chunksize = 10000 for chunk in pd.read_csv('large_data.csv', chunksize=chunksize):
    process(chunk)  # Process each chunk separately
```

44.3 Reading Excel Files

44.3.1 Excel File Formats

- Excel files can be .xls (older format) or .xlsx (newer XML-based format).
- They can contain multiple sheets.

44.3.2 Reading Excel with Pandas

```
df = pd.read_excel('data.xlsx')
print(df.head())
```

By default, reads the first sheet.

44.3.3 Important Parameters of read_excel()

Parameter	Description	Example
io	File path or ExcelFile object	'data.xlsx'
sheet_name	Sheet name or index to read (default 0 - first sheet)	'Sheet2' or [0, 2]
header	Row number for column headers (default 0)	header=None
names	List of column names	names=['A', 'B', 'C']
index_col	Column(s) to set as index	index_col=0
usecols	Columns to read	usecols="A:C" or usecols=[0,1,2]
skiprows	Rows to skip at the beginning	skiprows=3
dtype	Data types for columns	dtype={'A': int}
parse_dates	Columns to parse as dates	parse_dates=['Date']
nrows	Number of rows to read	nrows=500

44.3.4 Reading Multiple Sheets

```
xls = pd.ExcelFile('data.xlsx')
print(xls.sheet_names)  # List all sheet names
# Read specific sheets df1 = pd.read_excel(xls, sheet_name='Sheet1') df2 = pd.read_excel(xls, sheet_name='Sheet2') # Read multiple sheets into a dictionary of DataFrames dfs = pd.read_excel(xls, sheet_name=['Sheet1', 'Sheet2'])
```

44.4 Reading JSON Files

44.4.1 What is JSON?

- JSON (JavaScript Object Notation) is a lightweight data interchange format.
- Stores data as key-value pairs, arrays, or nested structures.
- Common in APIs and web data.

44.4.2 Reading JSON with Pandas

```
df = pd.read_json('data.json')
print(df.head())
```

44.4.3 JSON Formats Supported

- JSON object: { "col1": [1,2], "col2": [3,4] }
- JSON array: [{"col1":1,"col2":3}, {"col1":2,"col2":4}]
- JSON lines (each line is a JSON object): pd.read_json('file.json', lines=True)

44.4.4 Important Parameters of read_json()

Parameter	Description	Example
path_or_buf	File path or JSON string	'data.json'
orient	Format of JSON string	'records', 'split', 'index'
typ	Return type: 'frame' (DataFrame) or 'series'	'frame'
convert_dates	Columns to parse as dates	convert_dates=['date']
lines	Whether file is JSON lines format	lines=True

44.4.5 Example: Reading JSON Lines

```
df = pd.read_json('data_lines.json', lines=True)
```

44.5 Additional Tips and Tricks

44.5.1 Handling Encoding Issues

- Files may use encodings like UTF-8, ISO-8859-1, etc.
- Specify encoding to avoid errors: df = pd.read_csv('data.csv', encoding='utf-8')

44.5.2 Parsing Dates Automatically

Pass parse_dates parameter with column names to convert strings to datetime objects:

```
df = pd.read_csv('data.csv', parse_dates=['Date'])
```

44.5.3 Reading from URLs

Pandas can read files directly from web URLs:

```
url = 'https://example.com/data.csv'
df = pd.read_csv(url)
```

44.6 Summary

File Type	Reading Function	Key Parameters	Notes
CSV	pd.read_csv()	sep, header, index_col, usecols, parse_dates	Most common, simple tabular data
Excel	pd.read_excel()	sheet_name, header, index_col, usecols	Supports multiple sheets
JSON	pd.read_json()	orient, lines, convert_dates	Supports nested and linedelimited JSON

Chapter 45: Data Selection, Filtering, and Sorting

Introduction

When working with large datasets, it's important to extract the relevant data. Pandas provides efficient and intuitive methods for selecting specific rows and columns, filtering data based on conditions, and sorting it for analysis. These operations form the core of data manipulation and are essential for data analysis, machine learning, and reporting.

45.1 Data Selection in Pandas

45.1.1 Selecting Columns

To select specific columns, use the column names with single or double brackets:

```
import pandas as pd data = pd.DataFrame({
'Name': ['Alice', 'Bob', 'Charlie'],
 'Age': [24, 27, 22],
 'City': ['Delhi', 'Mumbai', 'Kolkata']
# Selecting a single column (returns a Series) print(data['Name'])
# Selecting multiple columns (returns a DataFrame) print(data[['Name', 'City']])
```

45.1.2 Selecting Rows by Index

Use .loc[] for label-based selection and .iloc[] for position-based selection:

```
# loc selects rows by label (index name) print(data.loc[0])  # First row as Series
print(data.loc[0:1])  # Rows from index 0 to 1 inclusive
# iloc selects rows by integer position print(data.iloc[1])  # Second row as Series
print(data.iloc[0:2])  # First two rows
```

45.1.3 Selecting Specific Rows and Columns

```
You can combine row and column selection: # Row 0, only 'Name' column
print(data.loc[0, 'Name'])
# Row 0 and 2, 'Name' and 'City' columns
print(data.loc[[0, 2], ['Name', 'City']])
```

45.2 Filtering Data

Filtering allows you to select rows that meet certain conditions.

45.2.1 Basic Condition Filtering

```
# Filter rows where Age > 23 filtered = data[data['Age'] > 23] print(filtered)
```

45.2.2 Multiple Conditions

```
Use parentheses around each condition: # Age > 23 AND City == 'Delhi'
print(data[(data['Age'] > 23) & (data['City'] == 'Delhi')])
# Age < 25 OR City == 'Mumbai' print(data[(data['Age'] < 25) | (data['City'] ==
'Mumbai')])
```

45.2.3 Using isin() isin() checks whether each element is contained in a list:

```
# City in ['Delhi', 'Kolkata'] print(data[data['City'].isin(['Delhi', 'Kolkata'])])
```

45.2.4 Using str.contains()

Search for substrings in string columns: # Filter names containing 'a' (case-insensitive) print(data[data['Name'].str.contains('a', case=False)])

45.2.5 Using .notna() and .isna()

Filter rows based on missing values: # Drop rows where Age is missing print(data[data['Age'].notna()]) # Find rows where City is missing print(data[data['City'].isna()])

45.3 Sorting Data

45.3.1 Sorting by One Column

Sort by Age ascending print(data.sort_values(by='Age')) # Sort by Age descending print(data.sort_values(by='Age', ascending=False))

45.3.2 Sorting by Multiple Columns

Sort by City, then Age print(data.sort_values(by=['City', 'Age']))

45.3.3 Sorting the Index

You can also sort by the DataFrame's index:
print(data.sort_index())

45.4 More Data Selection Techniques

45.4.1 The query() Method

Use SQL-like syntax to filter rows: print(data.query('Age > 23 and City == "Delhi"'))

45.4.2 Using between()

Check for values within a range:
print(data[data['Age'].between(23, 26)])

45.4.3 Finding and Removing Duplicates

Check for duplicates print(data.duplicated()) # Remove duplicates print(data.drop_duplicates()) 45.4.4 Resetting the Index

After filtering or sorting:
filtered = data[data['Age'] > 23].reset_index(drop=True)

45.4.5 Checking for Missing Values

Total missing values per column print(data.isnull().sum())
Total missing values in entire DataFrame print(data.isnull().values.sum())

45.5 Summary

In this chapter, you learned:

- How to select specific columns, rows, or combinations using .loc, .iloc, and direct indexing.
- How to filter rows using single and multiple conditions.
- How to use methods like isin(), between(), and str.contains().
- How to sort by one or more columns or by index.
- How to use advanced filtering like query().
- How to handle missing data, check for duplicates, and reset the index.

Chapter 46: Data Cleaning 4 Missing Values, Duplicates, and Type Conversion

Introduction

Real-world datasets often come with inconsistencies such as missing values, duplicates, or incorrectly typed data. These issues can lead to incorrect analysis if not handled properly. This chapter focuses on techniques in Pandas to clean data effectively, ensuring it is ready for analysis or modeling.

46.1 Handling Missing Values

Missing values can occur for various reasons such as human error, data corruption, or unavailability of information. Pandas provides multiple tools to identify and handle missing data.

46.1.1 Detecting Missing Values

Missing values in Pandas are represented as NaN (Not a Number).

```
import pandas as
pd import numpy
as np data =
pd.DataFrame({
   'Name': ['Alice', 'Bob', None],
   'Age': [24, np.nan, 30],
   'City': ['Delhi', 'Mumbai', None]
# Check where values are missing (returns True for NaNs)
print(data.isnull())
# Count total missing values per column
print(data.isnull().sum())
```

46.1.2 Dropping Missing Values

You can remove missing data either row-wise or column-wise depending on the analysis requirement.

```
# Drop rows that have any missing value print(data.dropna())
# Drop columns that have any missing value print(data.dropn(axis=1))
# Drop only rows where all columns are missing print(data.dropna(how='all'))
```

46.1.3 Filling Missing Values

Filling missing values with placeholders, computed values, or interpolated data is often better than deleting them.

```
# Replace all NaNs with a fixed value print(data.fillna('Unknown'))
# Replace NaNs in 'Age' with the mean of the column data['Age'] =
data['Age'].fillna(data['Age'].mean())
# Fill NaNs with different values for each column using a dictionary
print(data.fillna({'Name': 'Not Provided', 'City': 'Not Specified'}))
```

46.1.4 Forward/Backward Fill

Used when missing values can be inferred from nearby data (common in time series).

```
# Forward fill: propagate last valid value forward
print(data.fillna(method='ffill'))
# Backward fill: propagate next valid value backward
print(data.fillna(method='bfill'))
```

46.1.5 Interpolation

Used to estimate missing data by interpolating between existing values.

```
# Interpolate numerical data
print(data['Age'].interpolate())
```

46.2 Handling Duplicates

Duplicate rows can inflate statistics and lead to biased results.

46.2.1 Finding Duplicates

```
# Returns a boolean Series marking duplicated rows
print(data.duplicated())
```

46.2.2 Removing Duplicates

```
# Removes all duplicate rows keeping the first
occurrence print(data.drop_duplicates()) # Keep
last occurrence of duplicates
print(data.drop_duplicates(keep='last'))
```

46.2.3 Dropping Duplicates Based on Specific Columns

```
# Remove duplicates based on specific column(s)
print(data.drop_duplicates(subset='Name'))
```

46.3 Type Conversion

Columns can have the wrong data type (e.g., numeric data stored as strings). Type conversion is essential for correct calculations.

46.3.1 Checking Data Types

```
# Show current data types print(data.dtypes)
```

46.3.2 Converting Data Types

```
# Convert to integer (after handling missing values) data['Age'] =
data['Age'].fillna(0).astype(int)
# Convert Age to string data['Age'] = data['Age'].astype(str)
```

46.3.3 Using to_datetime, to_numeric, to_timedelta

These functions are more flexible and can handle errors during conversion.

```
# Convert to datetime
dates = pd.DataFrame({'date': ['2022-01-01', '2022-02-01']})
print(pd.to_datetime(dates['date']))
```

 # Convert strings to numbers; invalid parsing will be set as NaN numbers = pd.Series(['1', '2', 'three']) print(pd.to_numeric(numbers, errors='coerce'))

 # Convert strings to time duration timedeltas = pd.Series(['1 days', '2 days', '3 days']) print(pd.to_timedelta(timedeltas))

46.4 Summary

In this chapter, you learned how to clean data using Pandas. Key operations included:

- **Missing Values: Detect using isnull(), clean using dropna(), fillna(), and interpolate().**
- **Duplicates: Identify with duplicated(), remove using drop_duplicates() with various configurations.**
- **Type Conversion: Use astype() for basic conversion or to_datetime(), to_numeric() for more flexible conversion.**

Chapter 47: String Operations and Column Creation

Introduction

When working with textual data in Pandas, string manipulation becomes a vital part of the data wrangling process. Additionally, creating new columns using existing data is fundamental to transforming datasets into a usable form. This chapter explores common string operations and various methods of column creation in Pandas.

47.1 String Operations in Pandas

String operations in Pandas are performed using the .str accessor on Series objects. This allows vectorized string functions, meaning operations are applied efficiently across all values in a column.

47.1.1 Common String Methods

import pandas as pd data = pd.DataFrame({
'Names': ['Alice', 'BOB', 'charlie', 'david', 'Eva']

str.lower() and str.upper()

Converts strings to lowercase or uppercase:
print(data['Names'].str.lower()) print(data['Names'].str.upper()) str.capitalize() and str.title()
Capitalizes the first letter or every word:
print(data['Names'].str.capitalize()) print(data['Names'].str.title())

str.strip(), str.lstrip(), str.rstrip()

Removes whitespace:
text = pd.Series([' hello ', ' world ']) print(text.str.strip()) str.replace() Replaces substrings: print(data['Names'].str.replace('a', '@', case=False)) str.contains()
Checks if a string contains a substring: print(data['Names'].str.contains('a', case=False)) str.startswith() and str.endswith() Check for start or end characters:
print(data['Names'].str.startswith('A')) print(data['Names'].str.endswith('a'))

str.len()

Returns the length of each string: print(data['Names'].str.len()) str.split() and str.join() Split strings and join list elements:
emails = pd.Series(['john.doe@example.com'])
print(emails.str.split('@')) words = pd.Series(['hello world'])
print(words.str.split().str.join('-')) str.find() and str.index() Return position of substring: print(data['Names'].str.find('a'))

str.zfill()

Pads numeric strings on the left with zeros: numbers = pd.Series(['1', '23', '456'])
print(numbers.str.zfill(5)) str.get()

Extract a specific character by position: print(data['Names'].str.get(0)) # first letter of each name str.pad()

Pads strings on the left, right, or both: s = pd.Series(['cat', 'dog']) print(s.str.pad(5, side='left', fillchar='-')) str.center() and str.ljust() / str.rjust()

Align strings within a given width:

print(s.str.center(6, '*')) print(s.str.ljust(6, '*')) print(s.str.rjust(6, '*')) str.extract()

Extract groups from strings using regex: info = pd.Series(['Tom-25', 'Jerry-30']) print(info.str.extract(r'(\w+)-(\d+)')) # returns two columns

47.2 Creating New Columns

You can create new columns in Pandas by combining or transforming existing columns.

47.2.1 Creating Columns from Operations

```
df = pd.DataFrame({
   'A': [10, 20, 30],
   'B': [1, 2, 3]
# Create column C as sum of A and B
df['C'] = df['A'] + df['B'] print(df)
```

47.2.2 Creating Columns Using Conditions

```
# Create column D where D is 'High' if A > 15, else 'Low'
df['D'] = df['A'].apply(lambda x: 'High' if x > 15 else 'Low') print(df)
```

47.2.3 Using assign() Method

```
# Create new column using assign (does not modify in-place by default) df = df.assign(E=df['A'] * 2) print(df)
```

47.2.4 Using np.where() import numpy as np

```
# Vectorized conditional creation df['F'] = np.where(df['B'] > 1, 'Multiple', 'Single') print(df)
```

47.2.5 Using map(), apply(), and applymap()

```
# Apply a function to each element in a column capitalize = lambda x: x.upper() data['Names_Upper'] = data['Names'].map(capitalize) print(data)
```

47.2.6 Using lambda with Multiple Columns

```
df['G'] = df.apply(lambda row: row['A'] * row['B'], axis=1) print(df)
```

47.2.7 Using agg() for Multiple Aggregations df_summary = df.agg({

```
    'A': ['sum', 'mean', 'max'],
    'B': ['min', 'count']
print(df_summary)
```

47.3 Summary

In this chapter, you learned how to:

- Use powerful string manipulation methods using .str like lower(), upper(), strip(), replace(), split(), find(), extract(), pad(), and zfill().
- Create new columns using mathematical operations, conditionals, string formatting, and lambda expressions.
- Combine multiple columns using apply(), agg(), and vectorized logic like np.where().

Chapter 48: Merging, Joining, and Concatenating

Introduction

Data often comes from multiple sources and needs to be combined into a single, cohesive DataFrame for analysis. Pandas provides three powerful tools to combine data: merge(), join(), and concat(). This chapter explains their usage, differences, and best practices, with detailed examples.

48.1 Merging DataFrames

merge() is similar to SQL joins. It combines two DataFrames based on common columns or indices. Syntax pd.merge(left, right, how='inner', on=None, left_on=None, right_on=None)

Parameters

- how: Type of join – 'inner', 'outer', 'left', 'right'
- on: Column(s) to join on
- left_on, right_on: When the column names are different in each DataFrame Example import pandas as pd left = pd.DataFrame({

'ID': [1, 2, 3],
'Name': ['Alice', 'Bob', 'Charlie']
right = pd.DataFrame({
'ID': [2, 3, 4],
'Score': [85, 90, 95]
# Inner join result = pd.merge(left, right,
on='ID', how='inner') print(result)

Types of Joins

- Inner Join: Only matching rows.
- Left Join: All rows from left DataFrame, matched with right.
- Right Join: All rows from right DataFrame, matched with left.
- Outer Join: All rows from both DataFrames. Merge on Multiple Columns

pd.merge(df1, df2, on=['key1', 'key2'], how='outer')

48.2 Joining DataFrames

join() is a convenient method for combining DataFrames using the index or a key column. Syntax df1.join(df2, how='left', on=None) Example df1 = pd.DataFrame({

'ID': [1, 2, 3],
'Name': ['Alice', 'Bob', 'Charlie']
Set index to ID df1.set_index('ID', inplace=True) scores = pd.DataFrame({
'Score': [88, 76, 95] }, index=[1, 2, 3]) result = df1.join(scores) print(result)

- Useful when working with indexed data.
- Similar to merge(), but easier for index-based combining.

48.3 Concatenating DataFrames

concat() stacks DataFrames either vertically (rows) or horizontally (columns). Syntax

pd.concat(objs, axis=0, join='outer', ignore_index=False)

- axis=0: Concatenate rows
- axis=1: Concatenate columns Example – Vertical Concatenation df1 = pd.DataFrame({

'A': ['A0', 'A1'],

'B': ['B0', 'B1']

df2 = pd.DataFrame({

'A': ['A2', 'A3'],

'B': ['B2', 'B3']

result = pd.concat([df1, df2], axis=0) print(result)

Example – Horizontal Concatenation df3 = pd.DataFrame({

'C': ['C0', 'C1']

result = pd.concat([df1, df3], axis=1) print(result) Ignoring Index result = pd.concat([df1, df2], ignore_index=True) print(result)

48.4 Handling Missing Data While Combining

When combining DataFrames with different indices or columns, missing data (NaN) may appear.

Outer join will introduce NaNs pd.merge(left, right, how='outer', on='ID')

Use fillna() or dropna() to handle them:

result.fillna(0)

result.dropna()

48.5 Summary

- Use merge() for flexible, SQL-style joins on columns.
- Use join() for index-based joins with fewer parameters.
- Use concat() for stacking DataFrames vertically or horizontally.
- Handle missing values after combining with fillna() or dropna()

Chapter 49: Exploratory Data

Analysis (EDA): Descriptive Statistics and GroupBy

49.1 What is EDA (Exploratory Data Analysis)?

EDA is like exploring a new city before moving in. You try to understand:

- What kind of data is available?
- What are the averages?
- Are there any unusual values (outliers)?
- Are there missing values?
- What patterns can be observed?

EDA helps you know your data well before using it for predictions or building models.

49.2 Descriptive Statistics: Getting a Quick Summary

Descriptive statistics help us get a quick overview of the data.

Imagine you have student exam data:

- You want to know the average marks, highest score, lowest score, and so on.

✅ Common Pandas Functions

Function	What it does	Example Use
.mean()	Average value	Mean salary
.median()	Middle value	Median age
.mode()	Most frequent value	Most common department
.min()	Minimum value	Lowest marks
.max()	Maximum value	Highest marks
.count()	Number of non-missing values	Count of employees
.std()	Standard deviation (how spread out it is)	Salary variation
.var()	Variance (spread in squared units)	Salary consistency
.describe()	Full summary of stats (mean, std, etc.)	Overall summary

49.3 Using Descriptive Statistics in Python

Example:

```
import pandas as pd # Create sample data
data = {
   'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eva'],
   'Age': [25, 30, 35, 40, 28],
   'Salary': [50000, 60000, 70000, 80000, 52000]
df = pd.DataFrame(data) # View basic stats print(df.describe())
```

Output:

Age Salary

count 5.000000 5.000000 mean 31.600000 62400.000000 std 6.114038 12247.448713 min 25.000000 50000.000000 25% 28.000000 52000.000000

50% 30.000000 60000.000000 75% 35.000000 70000.000000 max 40.000000 80000.000000

Explanation:

- **mean: Average value**
- **std: How spread out the data is**
- **min and max: Lowest and highest values**
- **25%, 50%, 75%: Percentiles (helpful for understanding distribution)**

49.4.Individual Statistical Functions

You can also use each function one by one.

```
# Average salary df['Salary'].mean()     # 62400.0
    # Maximum and minimum age
df['Age'].max()        # 40 df['Age'].min()        # 25
# Standard deviation of salary df['Salary'].std()      # 12247.44
    # Mode (most frequent) df['Age'].mode()        # May return one or more values
```

49.5.What is GroupBy?

Let9s say you have a supermarket dataset with sales info by departments like "Fruits", "Dairy", "Beverages".

You want to:

- **Know total sales per department**
- **Know average sales per department**

The GroupBy operation helps you do that.

Think of it as:

1. **Splitting the data into groups**
2. **Applying a function (like mean or sum)**
3. **Combining the result**

49.6 GroupBy Example: Average Salary per Department

```
data = {
   'Department': ['HR', 'IT', 'IT', 'HR', 'Finance'],
   'Employee': ['Alice', 'Bob', 'Charlie', 'David', 'Eva'],
   'Salary': [50000, 60000, 70000, 52000, 75000]
}
df = pd.DataFrame(data)
```

```
# Group by Department and calculate average salary
print(df.groupby('Department')['Salary'].mean())
```

Output:

```
Department
Finance    75000.0
HR         51000.0
IT         65000.0
```

Each department9s average salary is shown separately.

49.7 GroupBy with Multiple Aggregations

You can apply more than one function using .agg():

```
df.groupby('Department')['Salary'].agg(['mean', 'max', 'min', 'count'])
```

Output:

Department	mean	max	min	count
Finance	75000	75000	75000	1
HR	51000	52000	50000	2
IT	65000	70000	60000	2

- count: Number of employees in each department
- mean: Average salary
- min and max: Range of salaries

49.8 Grouping by Multiple Columns

You can group by more than one column, for example: Department and Gender.

```
data = {
   'Department': ['HR', 'IT', 'IT', 'HR', 'Finance'],
   'Gender': ['F', 'M', 'M', 'M', 'F'],
   'Salary': [50000, 60000, 70000, 52000, 75000]
}
df = pd.DataFrame(data) print(df.groupby(['Department', 'Gender'])['Salary'].mean())
```

Output:

```
Department  Gender
Finance     F       75000.0
HR          F       50000.0
HR          M       52000.0
IT          M       65000.0
```

49.9 Summary: Key Takeaways

Topic	Usefulness
.describe()	Gives a full statistical summary
.groupby()	Groups your data for deeper analysis

.agg()	Apply multiple summary functions at once
.count(), .mean()	Very useful for summaries and insights

Practice Exercise

Try the following:

```
# Create a DataFrame with columns: Product, Category, Sales
# Group by Category and find:
# - Total Sales
# - Average Sales #
- Count of Products
```

Example:

```
data = {
   'Product': ['A', 'B', 'C', 'D', 'E'],
   'Category': ['Electronics', 'Clothing', 'Clothing', 'Electronics', 'Grocery'],
   'Sales': [1000, 500, 800, 1200, 300]
    }
    df = pd.DataFrame(data)
```

Group and analyze

```
print(df.groupby('Category')['Sales'].agg(['sum', 'mean',
'count']))
```

Chapter 50: Visualization with Matplotlib and Seaborn (Detailed Guide)

50.1 What is Data Visualization?

Data visualization is the process of transforming data into visual formats like charts and plots to:

- Understand trends and relationships
- Spot outliers or mistakes
- Summarize large datasets
- Communicate findings effectively

Python has two powerful libraries for this:

- Matplotlib – Low-level, highly customizable plotting library
- Seaborn – High-level statistical visualization library built on top of Matplotlib

50.2.Getting Started

Installation pip install matplotlib seaborn Importing Libraries import matplotlib.pyplot as plt import seaborn as sns import pandas as pd

50.3.Matplotlib: Deep Dive into Each Method

50.3.1 plt.plot() – Line Plot Used

for plotting lines (x-y values).

x = [1, 2, 3, 4] y = [10, 20, 25, 30] plt.plot(x, y, color='blue', linestyle='--', marker='o', label='Growth') plt.title("Line Plot Example") plt.xlabel("X Axis") plt.ylabel("Y Axis") plt.legend() plt.grid(True) plt.show()

Common Parameters:

- color: 'red', 'blue', '#ff0000', etc.
- linestyle: '-', '--', '-.', ':'
- marker: 'o', 's', '^', '*'
- label: used with legend()

50.3.2 plt.bar() – Vertical Bar Plot

categories = ['A', 'B', 'C'] values = [5, 7, 4] plt.bar(categories, values, color='green') plt.title("Bar Chart") plt.xlabel("Categories") plt.ylabel("Values") plt.show()

50.3.3 plt.barh() – Horizontal Bar Plot plt.barh(categories, values, color='purple') plt.title("Horizontal Bar Chart") plt.show()

50.3.4 plt.scatter() – Scatter Plot

x = [1, 2, 3, 4, 5] y = [2, 4, 1, 3, 7]

```
plt.scatter(x, y, color='orange', s=100,
marker='*') plt.title("Scatter Plot")
plt.xlabel("X") plt.ylabel("Y") plt.show()
```

Parameters:

- s: size of points
- marker: symbol
- c: color

50.3.5 plt.hist() – Histogram

```
data = [22, 55, 62, 45, 22, 30, 20, 21, 22, 29]
plt.hist(data, bins=5, color='skyblue',
edgecolor='black') plt.title("Histogram")
plt.xlabel("Bins") plt.ylabel("Frequency")
plt.show()
```

Note: Used to show the distribution of values.

50.3.6 plt.pie() – Pie Chart

```
sizes = [25, 35, 40]
labels = ['A', 'B', 'C']
plt.pie(sizes, labels=labels, autopct='%1.1f%%', explode=(0, 0.1, 0),
startangle=90) plt.title("Pie Chart") plt.show()
```

50.3.7 Other Useful Matplotlib Functions

Function	Purpose
plt.xlim()	Set X-axis range
plt.ylim()	Set Y-axis range
plt.grid()	Enable or disable grid
plt.legend()	Show legend (labels from plot/bar/etc)
plt.figure()	Create a new figure for multiple plots
plt.subplot()	Divide plot area into multiple subplots
plt.tight_layout()	Adjust spacing to avoid overlap

50.4 Seaborn: Deep Dive into Each Plot

We9ll use a sample dataset:

```
df = sns.load_dataset('tips') # Contains columns: total_bill, tip, sex, smoker, day,
time, size
```

50.4.1 sns.barplot()

Shows average value of a numeric column grouped by categories.

```
sns.barplot(x='day', y='total_bill', data=df,
ci=None, palette='muted') plt.title("Barplot - Avg Total Bill per
Day") plt.show()
```

50.4.2 sns.countplot()
Counts number of occurrences of categories. sns.countplot(x='day', data=df, palette='pastel') plt.title("Count of Records per Day") plt.show()

50.4.3 sns.boxplot() – Box and Whisker Plot
sns.boxplot(x='day', y='total_bill', data=df) plt.title("Boxplot - Distribution of Bills by Day") plt.show() Shows:

- **Median**
- **Quartiles**
- **Outliers**

50.4.4 sns.violinplot() – Combines KDE and boxplot
sns.violinplot(x='day', y='total_bill', data=df, inner='quartile') plt.title("Violin Plot") plt.show()

50.4.5 sns.histplot() – Histogram
sns.histplot(df['total_bill'], bins=10, kde=True) plt.title("Histogram of Total Bill") plt.show()

50.4.6 sns.kdeplot() – Kernel Density Estimation
sns.kdeplot(df['total_bill'], shade=True, color='r') plt.title("KDE Plot") plt.show()

50.4.7 sns.scatterplot()
sns.scatterplot(x='total_bill', y='tip', hue='sex', style='smoker', data=df) plt.title("Scatter Plot with Hue and Style") plt.show()

50.4.8 sns.lmplot() – Linear Regression Plot
sns.lmplot(x='total_bill', y='tip', hue='sex', data=df) plt.title("Linear Regression Plot") plt.show()

50.4.9 sns.pairplot() – Grid of Plots sns.pairplot(df, hue='sex') plt.suptitle("Pairplot - Relationship among Numerical Columns") plt.show()

50.4.10 sns.heatmap() – Correlation Heatmap
corr = df.corr()
sns.heatmap(corr, annot=True, cmap='coolwarm', linewidths=0.5) plt.title("Correlation Heatmap") plt.show()

50.4.11 Styling and Themes
sns.set_style("darkgrid") # Options: white, whitegrid, dark, darkgrid, ticks
sns.set_palette("Set2") # Set color palette

50.5 Saving Figures

Matplotlib

```
plt.savefig("plot.png",
dpi=300)
```

50.6 Summary Table: Matplotlib vs Seaborn

Feature	Matplotlib	Seaborn
Level	Low-level	High-level
Customization	Manual	Pre-built aesthetics
Plots	Line, Bar, Pie, Scatter, etc.	Statistical: bar, count, violin, etc.
Style	Basic	Stylish
Best For	Full control	Fast and beautiful stats visualizations

50.7 Practice Exercises

1. **Plot a line chart of monthly sales.**
2. **Create a histogram of customer ages.**
3. **Use a heatmap to show correlation between numeric columns in a dataset.**
4. **Use pairplot() to explore multiple columns in the Titanic dataset.**
5. **Plot a countplot for number of male vs female passengers in Titanic dataset.**

Chapter 51: Power BI Interface and Ecosystem

51.1 What is Power BI?

Power BI is a Microsoft tool that allows users to connect, transform, analyze, and visualize data to gain business insights. It is part of the Power Platform, which includes Power Apps, Power Automate, and Power Virtual Agents.

Key Benefits:

- Makes raw data understandable through charts and graphs.
- Helps track key business metrics using dashboards and KPIs.
- Supports collaboration via cloud sharing.
- Allows integration with tools like Excel, SQL Server, Azure, etc.

51.2.Power BI Ecosystem Components

Power BI is not a single software but a suite of tools working together:

Component	Description
Power BI Desktop	A Windows application for creating data models and reports. Most work is done here.
Power BI Service	A cloud platform for sharing, publishing, and collaborating on reports (app.powerbi.com).
Power BI Mobile	Mobile apps for Android/iOS/Windows for on-the-go access to dashboards.
Power BI Gateway	Acts as a bridge between on-premises data sources and cloud services.
Power BI Report Server	Allows you to host and share reports in an on-premise environment (not cloud-based).
Power BI Embedded	A developer tool to embed Power BI visuals into web or mobile applications.

51.3.Download and Install Power BI Desktop

Power BI Desktop is available for free.

Installation:

- From Website: https://powerbi.microsoft.com/desktop/
- From Microsoft Store: Search "Power BI Desktop"

51.4 Understanding Power BI Desktop Interface

Once launched, the interface contains several panes and ribbons. Here's a breakdown:

51.4.1 Home Ribbon

Functions in the Home Ribbon:

Feature	Description
Get Data	Connect to Excel, SQL Server, Web, SharePoint, etc.
Transform Data	Opens the Power Query Editor
Recent Sources	Reopen previously used datasets
Enter Data	Manually create a small table
Refresh	Reload data from the source
Publish	Send your report to Power BI Service (cloud)
Manage Parameters	Create and modify input variables for filtering
Quick Insights	AI-generated insights from your data

51.4.2 Left Side Navigation Views

Icon View Name Use
Report View Design visual reports
Icon View Name Use
Data View View raw tables and calculated columns
Model View Manage relationships between tables
You can switch between them using the icons on the left.

51.4.3 Fields Pane

Located on the right side, it shows your dataset and tables.

- Click a table name to see all columns and measures.
- Right-click to create calculated columns, measures, or hierarchies.
- Search for fields if your model is large.

51.4.4 Visualizations Pane

This pane contains all the charts and visuals you can use.

Visual Type	Description
Bar/Column Charts	Compare values
Pie/Donut Charts	Show proportions
Line/Area Charts	Show trends over time

Table/Matrix	Show tabular data
Card/Multiro Card	Show KPIs
Map (Filled/ArcGIS)	Geospatial data
Slicer	Interactive filter
Decomposition Tree	Drill-down hierarchy
Gauge	Target vs actual visualization

You can drag fields into:

- Values → numerical data
- Axis → category
- Legend → subgrouping
- Tooltip → data shown on hover

51.4.5 Filters Pane

This pane allows filtering data at:

- Visual Level – filter only that chart
- Page Level – filter all visuals on that page
- Report Level – filter across all pages

You can also use relative date filters (e.g., <last 30 days=).

51.5.Power Query Editor: Data Preparation Area

Clicking Transform Data opens Power Query Editor, used for ETL (Extract, Transform, Load).

Main Tasks:

Task	Examples
Remove rows/columns	Delete unwanted data
Change data types	Text, Date, Number
Split/Merge columns	Based on delimiter
Pivot/Unpivot	Convert between wide and long formats
Replace values	Fix typos or empty strings
Group by	Summarize data
Merge Queries	Combine data from multiple tables

Every action you take is stored as a step on the right sidebar.

51.6.Relationships and Data Modeling

Power BI uses data models to combine multiple tables.

Types of Relationships:

Type	Description
One-to-Many	Most common, e.g., Customer ID from Customers to Sales
Many-to-Many	Advanced modeling
Single/Both Cross Filter	Controls direction of data filtering between tables

You can use Model View to:

- View tables as boxes with fields
- Drag a column from one table to another to create relationships

51.7 DAX: Data Analysis Expressions

DAX is Power BI9s formula language, used for:

- Calculated Columns
- Measures
- Calculated Tables

Examples:

Type	Syntax	Description
Calculated Column	IF([Sales] > 1000, "High", "Low")	Row-level logic
Measure	SUM(Sales[Amount])	Aggregate summary
Time Intelligence	TOTALYTD(SUM(Sales[Amount]), 'Date'[Date])	Year-to-date total

51.8 Report Design:

1. Connect Data: Excel, SQL, Web, SharePoint, etc.
2. Transform in Power Query: remove errors, clean data.
3. Model Relationships: build a star schema.
4. Build Visuals: drag fields, select chart types.
5. Add Filters/Slicers: interactive controls.
6. Create DAX Measures: total revenue, profit margin.
7. Design and Theme: add titles, backgrounds, consistent colors.
8. Save and Publish: to the Power BI Service.

51.9.Power BI Service: Cloud-Based Platform

Visit: https://app.powerbi.com

Features:

Feature	Use
Workspaces	Group reports and dashboards by team/project
Dashboards	Pin visuals from different reports
Sharing	Email or give access to others
Data Refresh	Schedule updates from sources
Alerts	Notify if KPI crosses threshold
Comments	Collaborate on insights

51.10.Power BI Mobile App

- Download from iOS/Android stores
- Supports interactive reports
- Set alerts and subscribe to dashboards
- Great for executives and field workers

51.11 Power BI Gateway

Needed when your data is on-premises (not in cloud).

Types:

- Personal Gateway: For one user
- Enterprise Gateway: For organizations

Used to schedule refreshes of data from local SQL, Excel files, etc.

51.12.Common Visuals with Use Cases

Visual	Use Case
Line Chart	Trends over time (e.g., sales by month)
Bar/Column Chart	Compare categories (e.g., revenue by region)
Pie/Donut Chart	Show part-to-whole (e.g., market share)
Matrix/Table	Tabular view like Excel
Card/Multiro Card	Show totals or KPIs

Slicer	**Allow user to filter reports**
Map	**Show data by location (region, city, etc.)**
Gauge	**Performance against target**
Decomposition Tree	**Break down KPIs by multiple dimensions**

51.13.Best Practices

- **Use star schema in data modelling**
- **Create measures instead of calculated columns for performance**
- **Use themes for consistent design**
- **Create a dashboard layout plan before building**
- **Don't overload with visuals – keep it clear and focused**
- **Enable row-level security (RLS) for user-specific access**
- **Use bookmarks to create interactive stories**

51.14 Practice Exercises

1. **Import Excel file with sales data, clean it using Power Query.**
2. **Create relationship between Orders and Customers tables.**
3. **Build bar chart showing Total Sales by Region.**
4. **Add slicer for filtering by year.**
5. **Add a DAX measure Profit Margin = Profit / Sales.**
6. **Publish report to Power BI Service and share it.**
7. **Schedule data refresh using a personal gateway.**

51.15 Summary

Power BI is a complete, end-to-end platform for creating business intelligence solutions. From importing data and transforming it with Power Query to building stunning reports and dashboards using DAX and sharing them via the Power BI service – it enables everyone from analysts to managers to make data-driven decisions.

Chapter 52: Connecting to Data Sources (Excel, SQL, CSV) in Power BI

Power BI is a powerful data visualization tool, but its value comes from how easily it can connect to and transform data from multiple sources. In this chapter, we will explore how to connect Power BI to different data sources like:

- Microsoft Excel files
- SQL Server databases
- CSV (Comma-Separated Values) files

We will also cover best practices, common errors, and how to manage queries using Power Query Editor.

52.1 Introduction to Data Sources in Power BI

Power BI allows connections to various types of data, including:

- Local files (Excel, CSV, JSON, XML)
- Databases (SQL Server, MySQL, Oracle, etc.)
- Cloud services (SharePoint, OneDrive, Azure, etc.)
- Online services (Google Analytics, Salesforce, etc.)
- Web pages and APIs

For this chapter, we will focus on three of the most commonly used sources:

- Excel
- CSV
- SQL Server

52.2 Connecting to Excel Files

Excel files are widely used for storing and sharing structured data. Power BI can easily connect to .xls or .xlsx files.

Steps to Connect:

1. Open Power BI Desktop.
2. Click on Home > Get Data > Excel.
3. Browse and select the Excel file.
4. Navigator pane will open – select the sheet(s) or named range(s) you want.
5. Click Load to import, or Transform Data to open Power Query Editor for cleaning.

Example:

You have a file SalesData.xlsx with a sheet named Jan_Sales.

Steps:

- Go to Home > Get Data > Excel
- Select SalesData.xlsx
- Choose Jan_Sales in the Navigator
- Click Load

Notes:

- **Excel file must not be open in write mode while importing.**
- **If your sheet contains headers, Power BI detects them automatically.**
- **Use <Transform= for cleaning like removing nulls, renaming columns, etc.**

52.3 Connecting to CSV Files

CSV files are plain text files with values separated by commas. These are ideal for exporting/importing lightweight datasets.

Steps to Connect:

1. **Go to Home > Get Data > Text/CSV.**
2. **Select the .csv file.**
3. **Power BI auto-detects delimiter (usually comma).**
4. **Click Load or Transform Data.**

Example:

You have a file products.csv with the following content:

ProductID,ProductName,Price

101,Mouse,500

102,Keyboard,700

- **Choose Text/CSV from the Get Data menu**
- **Browse and select products.csv**
- **Load data into your model**

Tips:

- **If your CSV uses semicolons or tabs as delimiters, select it manually.**
- **Watch out for date/time format issues while importing.**

52.4 Connecting to SQL Server Databases

SQL Server is one of the most popular relational databases in enterprises.

Steps to Connect:

1. **Go to Home > Get Data > SQL Server.**
2. **Enter:**
   - **Server name (e.g., localhost, DESKTOP-SQL2019)**
   - **Optional: Database name**
3. **Choose authentication method:**
   - **Windows o Database**
4. **Click OK.**
5. **Choose tables or write a custom SQL query.**

Example:

Suppose you have a database named SalesDB on a local server:

- **Open Power BI > Get Data > SQL Server**
- **Enter localhost as the server name**
- **Choose SalesDB**
- **Select Sales table > Click Load Advanced: Use a Custom SQL Query**

```
SELECT ProductName, SUM(SalesAmount) AS TotalSales
FROM Sales
GROUP BY ProductName
```

Paste this query in the SQL Server connector dialog instead of loading an entire table.

52.5.Power Query Editor Overview

Once data is connected, Power BI opens Power Query Editor (if you click "Transform").

Here you can:

Task	Description
Remove columns	Delete unnecessary fields
Rename columns	Make names readable
Change data types	Text, Date, Number, etc.
Filter rows	Keep only needed records
Split columns	Divide full names, dates, etc.
Merge/Append queries	Combine data from different tables/files
Add custom columns	Use formulas to derive new data

Every transformation is stored as a step and can be reverted or edited later.

52.6.Refreshing and Managing Data Sources

Once data is loaded, you may need to refresh it when the original data changes.

To Refresh:

- Click Home > Refresh
- To schedule automatic refresh:
- Publish your report to Power BI Service
- Set refresh frequency in the dataset settings

Common Issues:

- File path changed or deleted
- SQL Server credentials expired
- Network/firewall blocks server

Use Data Source Settings under File > Options and Settings to manage these.

52.7.Best Practices

Tip	Description
Use folder structure	Organize source files in a consistent location
Use parameters for connections	Allow easy changes (e.g., database names)
Document your queries	Rename steps clearly in Power Query Editor
Avoid loading unused columns	Reduces size and speeds up performance
Keep data models clean	Don9t over-import unnecessary tables

52.8.Summary

Data Source	Access Method	Notes
Excel	Get Data > Excel	Must be closed during import
CSV	Get Data > Text/CSV	Auto-detects delimiter
SQL Server	Get Data > SQL Server	Can use credentials & custom queries

52.9.Practice Tasks

1. Load an Excel file with monthly sales data.
2. Import a CSV file of customer records and remove null rows.
3. Connect to a SQL Server database and load only the Products table.
4. Rename all columns to meaningful names using Power Query Editor.
5. Merge two Excel sheets using <Append Queries=.

Chapter 53: Power Query Editor Basics

53.1 Introduction to Power Query Editor

Power Query Editor is a powerful data connection and transformation tool in Power BI that allows users to connect to various data sources, transform raw data, and prepare it for visualization and analysis. It simplifies the data cleansing and shaping process using a userfriendly interface, eliminating the need for complex coding.

53.2 Launching Power Query Editor

To open Power Query Editor:

1. Open Power BI Desktop.
2. Click on the "Home" tab.
3. Select "Transform Data."

This opens a new window: the Power Query Editor.

53.3 Power Query Editor Interface Overview

The Power Query Editor interface contains several important sections:

1. Ribbon:

Contains tabs like Home, Transform, Add Column, View, and Tools. Each tab has groups of commands to perform data operations.

2. Queries Pane:

Located on the left, this shows all queries (tables of data) loaded into Power Query.

3. Data Preview:

The central section displays a preview of the data from the selected query.

4. Formula Bar:

Displays the M code (Power Query formula language) for each transformation.

5. Query Settings Pane:

Located on the right, it shows the name of the query and the list of applied steps.

53.4 Key Tabs in the Ribbon

Home Tab:

- New Source: Connect to a new data source.
- Manage Columns: Keep, remove, reorder columns.
- Reduce Rows: Remove top, bottom, duplicates, or blank rows.
- Sort: Ascending or descending order.
- Close & Apply: Load transformed data to Power BI.

Transform Tab:

- Data Type: Change column data type (e.g., text, number, date).
- Replace Values: Replace specific text or values.
- Split Column: Split by delimiter or number of characters.
- Group By: Aggregate data based on one or more columns.

Add Column Tab:

- **Custom Column: Add a column using custom logic.**
- **Index Column: Add a row index.**
- **Conditional Column: Add based on if-else logic.**

View Tab:

- **Query Settings Pane: Show/hide right-side pane.**
- **Advanced Editor: Open M code editor for detailed editing.**

53.5 Common Data Transformations

Renaming Columns:

Right-click column header > Rename.

Changing Data Types:

Select column > Transform Tab > Data Type.

Removing Columns:

Right-click on column header > Remove.

Filtering Rows:

Click filter icon on column header and apply conditions.

Removing Duplicates:

Home Tab > Remove Rows > Remove Duplicates.

Splitting Columns:

Select column > Transform Tab > Split Column > By Delimiter.

Replacing Values:

Select column > Transform Tab > Replace Values.

Merging Queries:

Home Tab > Merge Queries > Combine two queries using key columns.

Appending Queries:

Home Tab > Append Queries > Stack data from multiple queries.

53.6 Query Settings Pane

- **This section on the right shows:**
- **Properties: Name of the current query.**
- **Applied Steps: Each transformation applied to the query, in sequence. You can click the gear icon to edit a step or 'X' to delete it.**

53.7.Formula Bar and M Code

The formula bar shows the M code for each transformation. You can enable it from the View tab if it's not visible.

Example:

```
= Table.RemoveColumns(Source, {"Column1"})
```

53.8.Using the Advanced Editor

You can view or edit all M code for a query in one place.

- **View Tab > Advanced Editor**

It9s useful for:

- **Writing complex logic**
- **Copying queries**
- **Understanding transformation steps**

53.9 Closing and Loading Data

Once transformations are complete:

- **Click Close & Apply in the Home tab.**
- **The data will be loaded into Power BI for analysis and visualization.**

53.10 Best Practices

- **Name queries and columns clearly.**
- **Reduce the number of steps to improve performance.**
- **Use Group By for summarization.**
- **Validate transformations using data preview.**

53.11 Summary

Power Query Editor is an essential tool for data preparation in Power BI. It allows users to clean, shape, and transform data visually, making it ready for further analysis and reporting.

Understanding Power Query basics is critical for building reliable and insightful Power BI reports.

Chapter 54: Creating Visualizations - Bar, Pie, and Column Charts in Power BI

54.1 Introduction

Visualizations in Power BI help you present your data in a graphical format that is easy to understand and interpret. Charts not only allow analysts to convey insights clearly but also help decision-makers interact with and explore the data visually.

- The most fundamental charts include:
- Bar Charts – Ideal for comparing data between different groups horizontally.
- Column Charts – Good for displaying data over time or grouped by categories.
- Pie Charts – Best used to show how parts make up a whole (percentages or proportions).

These charts form the foundation of data storytelling in Power BI.

54.2 Getting Started with Visualizations in Power BI

Steps to Create a Visualization:

1. Load your dataset into Power BI using "Get Data" from Excel, CSV, SQL Server, etc.
2. Navigate to the Report View by clicking on the middle icon in the vertical toolbar (with the bar chart symbol).
3. From the Visualizations Pane on the right-hand side, select the chart type you want to create.
4. Use the Fields Pane to drag and drop fields into different chart buckets like Axis, Value, Legend, or Tooltips.
5. Use the Format Pane (paint roller icon) to adjust visual elements, including colors, labels, titles, legends, and more.

Example Dataset Fields:

- Category: Product, Region, Department
- Numerical Measure: Sales, Profit, Quantity
- Time: Month, Year, Quarter

54.3 Bar Chart

Purpose:

Bar charts are ideal for comparing values across categories. These charts display horizontal bars where the length of the bar represents the value.

How to Create:

1. Click on the Bar Chart icon (e.g., Clustered Bar Chart).
2. Drag a categorical field to the Y-axis (e.g., Product Names).
3. Drag a numerical field to the X-axis (e.g., Sales).

Example:

- **Y-axis: Region (East, West, North, South)**
- **X-axis: Total Sales**

Format Options:

- **Data Colors: Customize the color of each bar (static or based on rules).**
- **Data Labels: Show numeric values at the end of each bar.**

Axis Titles: Name each axis clearly for readability.

Tooltip: Add extra fields to display details on hover.

Sort Order: Change ascending/descending based on values or categories.

- **Bar Padding: Adjust spacing between bars.**
- **Border & Transparency: Highlight or soften visual impact.**

Best Practices:

- **Limit the number of categories to avoid clutter.**
- **Sort bars to quickly show largest/smallest values.**
- **Use conditional formatting to highlight key values.**

54.4 Column Chart

Purpose:

Column charts are similar to bar charts but use vertical bars. These are great for showing changes over time or category comparisons.

How to Create:

1. **Select the Clustered Column Chart visual.**
2. **Drag a field like Month or Product to the X-axis.**
3. **Drag a numeric field like Profit or Sales to the Y-axis.**

Example:

- **X-axis: Month (January to December)**
- **Y-axis: Total Profit**

Variants:

- **Stacked Column Chart: Displays multiple series stacked vertically.**
- **100% Stacked Column Chart: Shows proportional contributions from each series as percentages.**

Format Options:

- **Legend: Used to separate categories (like Region or Product Type).**
- **Gridlines: Show or hide horizontal reference lines.**
- **Title & Axis Font: Adjust font size, color, alignment.**
- **Column Width: Adjust the gap between columns.**
- **Data Labels: Add values on top of columns.**
- **Color by Value: Useful for KPI-based visualization.**
- **Best Practices:**
- **Use stacked variants when comparing parts of a whole.**
- **Use appropriate intervals and sorting on time-based axes.**
- **Use consistent colors across reports.**

Purpose:

Pie charts are ideal when you want to show proportions that make up a whole. Each slice represents a category's share of the total.

How to Create:

1. **Click on the Pie Chart icon in the Visualizations pane.**
2. **Drag a categorical field (e.g., Department) to the Legend.**
3. **Drag a numeric field (e.g., Total Sales) to the Values.**

Example:

- **Legend: Department (HR, Finance, Marketing)**
- **Values: Total Budget per department**

Format Options:

- **Detail Labels: Customize to show category, value, percentage.**
- **Legend: Position (top, bottom, right, left), font, color.**
- **Slices Colors: Assign fixed colors or use conditional formatting.**
- **Explode Slices: Use for emphasis by separating one slice.**
- **Inner Radius: Transform pie into donut chart.**

Best Practices:

- **Use only when there are 2–5 categories.**
- **Avoid using with similar-sized values, which may confuse viewers.**
- **Don9t overuse pie charts—bar/column charts are often more precise.**

54.6 Tips for Effective Chart Design

- **Always include axis titles to make charts self-explanatory.**
- **Use clear, contrasting colors to differentiate categories.**
- **Apply data labels sparingly to avoid clutter.**
- **Avoid 3D effects that can distort data interpretation.**
- **Use tooltips and interactivity to add detail without overcrowding.**
- **Keep charts simple and focused on a specific insight.**
- **Align your color scheme with your company branding if applicable.**
- **Group similar visuals together to maintain layout consistency.**

54.7 Summary

Bar, Column, and Pie charts are foundational tools in Power BI for representing and exploring data visually. These charts are simple to create but powerful in the insights they deliver when used effectively. By understanding how to create and format these charts properly, you can build dashboards and reports that not only look good but also tell compelling data stories.

Chapter 55: Using Filters and Slicers

55.1 What Are Filters in Power BI?

Filters are tools that restrict the data shown in reports or visuals. By applying filters, you can focus on specific data points or ranges, making your report clearer and more relevant.

Why filters matter:

- Help analyze targeted data segments
- Improve report performance by limiting data volume
- Make dashboards interactive and user-friendly

55.2 Types of Filters

Power BI offers several filter levels that control data visibility differently:

- Visual Level Filters:

Affect only the visual they are applied to. For example, a bar chart showing sales filtered only for "2024" without affecting other visuals.

- Page Level Filters:

Affect all visuals on the current page. Useful to filter all data on a report page to one region or product.

- Report Level Filters:

Affect the entire report, including all pages and visuals. Great for broad filtering like selecting only one country across the entire report.

- Drillthrough Filters:

Used when you want to click on a data point in one report page and jump to a detailed page showing related data filtered for that point.

55.3 How to Add Filters

Step-by-step process:

1. Click on the visual or page to which you want to add a filter.
2. Drag the field (e.g., Country, Date, Product) to the Filters pane on the right.
3. Select the type of filter — basic (checkboxes), advanced (e.g., "greater than 100"), or relative date filters.
4. Customize filter settings: Include/exclude specific values, set ranges, or search for items.
5. Visuals will update immediately to reflect the applied filter.

55.4 What Are Slicers?

Slicers are visual filter controls that appear on the report canvas. Unlike filters that are hidden in panes, slicers provide interactive buttons or dropdowns users can click or select to filter data visually.

Benefits:

- Intuitive to use
- Make dashboards interactive

- **Can be styled and arranged to fit report design**

55.5 Types of Slicers

List Slicer:

- **Displays a list of values with checkboxes for multi-selection.**
- **Dropdown Slicer:**
- **A compact slicer showing a dropdown menu for selections, saving report space.**

Date Slicer:

Filters by date with options like "between," "before," or "relative date" (e.g., last 7 days).

- **Numeric Slicer:**

Used for ranges of numbers; users can select min and max values.

- **Hierarchical Slicer:**

Allows filtering based on multiple levels (e.g., Year > Quarter > Month), ideal for drilling into time-based or category hierarchies.

55.6 How to Add and Configure Slicers

Steps to add:

1. **In the Visualizations pane, click the Slicer icon (looks like a funnel with a checklist).**
2. **Drag a field (e.g., Region, Category, Date) into the slicer.**
3. **Position the slicer on the report page.**
4. **Configure slicer options:**
    - **Choose single select (only one option at a time) or multi-select. o Set orientation: vertical or horizontal.**
    - **Enable search box for large lists.**
    - **Use Sync slicers (under View menu) to apply slicer selections across multiple report page**

55.7 Advanced Slicer Features

Sync Slicers:

- **Synchronize slicer selections across multiple pages, maintaining consistent filtering.**
- **Slicer Panels and Bookmarks:**
- **You can hide slicers in panels and use bookmarks/buttons to toggle visibility, improving dashboard cleanliness.**
- **Formatting Slicers:**
- **Customize colors, fonts, borders, and selection controls in the Format pane to match report branding.**

55.8 Best Practices for Filters and Slicers

Use filters for backend data restrictions and slicers for front-end user interaction.

- **Limit the number of slicers to avoid overwhelming users.**
- **Group related slicers together.**
- **Use hierarchical slicers for complex data structures.**
- **Always test performance; too many filters/slicers can slow reports.**
- **Provide clear labels and titles for slicers.**

55.9 Conclusion

Filters and slicers are powerful tools in Power BI that enable users to explore and analyze data interactively and efficiently. Mastering their use helps create professional, userfriendly reports that provide actionable insights at a glance.

Chapter 56: Data Modeling and Relationships

56.1 What is Data Modeling?

Data modeling in Power BI is the process of organizing your data tables and defining how they relate to each other. A good data model helps in:

- Combining data from multiple sources effectively
- Making your reports faster and more accurate
- Enabling complex calculations and insightful analytics

Think of it as building a blueprint that structures your data so Power BI can analyze it efficiently.

56.2 Why Are Relationships Important?

When you import data from different tables or sources (like Excel files, SQL databases), these tables usually have some logical connections. For example:

- A Sales table might reference a Customer table by Customer ID.
- A Product table connects with a Sales table by Product ID.

These connections are called relationships. Defining them correctly is essential because:

- It allows Power BI to understand how tables relate.
- Enables you to create visuals that pull data from multiple tables seamlessly.
- Prevents data duplication or incorrect aggregations.

56.3 Types of Relationships

In Power BI, relationships between tables can be:

- One-to-Many (1:*):
- The most common type. One record in the first table relates to many records in the second. For example, one customer can have many sales.
- Many-to-One (*:1):
- The inverse of one-to-many, often viewed the same way.
- Many-to-Many (:):
- Both tables can have multiple matching records. This requires special handling with composite models and can affect performance.
- One-to-One (1:1):
- Each record in one table corresponds to exactly one record in another.

56.4 How to Create Relationships

Automatically:

When you load multiple tables, Power BI tries to detect relationships automatically based on column names and data types.

Manually:

You can create or edit relationships by:

1. Go to Model view (icon with connected boxes on left pane).

2. **Drag a column from one table to the matching column in another table.**
3. **Power BI opens a window where you set:**

- **Cardinality: one-to-many, many-to-one, etc.**
- **Cross filter direction: single or both. o Make relationship active/inactive.**

56.5 Understanding Cardinality

- **One-to-Many (1:*):**

One unique value in Table A matches multiple rows in Table B.

- **Many-to-Many (:):**

Multiple values in Table A relate to multiple values in Table B.

- **One-to-One (1:1):**

Unique matching in both tables.

Selecting the right cardinality ensures correct data aggregation and filtering.

56.6 Cross Filter Direction

Cross filter direction controls how filters flow between related tables:

- **Single Direction:**

Filters flow from one table to another (usually from a lookup/dimension table to a fact table). This is the default and usually best for performance.

- **Both Directions:**

Filters flow both ways. Useful for some advanced modeling scenarios but can cause circular dependencies or performance issues.

56.7.Active vs Inactive Relationships

You can have multiple relationships between the same tables but only one can be active at a time. The active relationship is used by default in calculations.

Inactive relationships can be used with DAX functions like USERELATIONSHIP to activate temporarily during specific calculations.

56.8.Star Schema vs Snowflake Schema

Star Schema

A Star Schema is a type of data modeling structure that is simple, flat, and easy to use, especially in reporting and data analysis tools like Power BI.

Structure:

- **Central Fact Table: This contains numerical data (like sales, revenue, quantity) and foreign keys to connect with dimension tables.**
- **Dimension Tables: These contain descriptive attributes (like customer name, product name, region, date, etc.).**
- **Each dimension table connects directly to the fact table — creating a star-like shape, hence the name.**

Example:

Imagine a Sales Fact Table:

Sale_ID	Date_ID	Customer_ID	Product_ID	Amount

And dimension tables like:

- **Date Dimension: Date_ID, Day, Month, Quarter, Year**
- **Customer Dimension: Customer_ID, Name, Region, Age Group**
- **Product Dimension: Product_ID, Product Name, Category, Brand**

Key Features:

- **Denormalized: Dimension tables are not split into smaller sub-tables.**
- **Faster performance: Because fewer joins are needed.**
- **Easy to understand: Great for business users and report creators.**
- **Best for Power BI: Because it runs faster queries and simplifies DAX formulas.**

Snowflake Schema

The Snowflake Schema is a more normalized version of the star schema. This means the dimension tables are split into multiple related tables, to remove redundancy.

Structure:

- **Still has a central fact table, just like the star schema.**
- **But dimension tables are broken down into sub-dimensions to normalize data.**
- **This creates a snowflake-like pattern — hence the name.**

Example:

Same Sales Fact Table as in Star Schema.

But now, the Product Dimension might be split like:

- **Product Table: Product_ID, Product Name, Category_ID**
- **Category Table: Category_ID, Category Name, Department_ID**
- **Department Table: Department_ID, Department Name**

Similarly, Customer might be:

- **Customer Table: Customer_ID, Name, Region_ID**
- **Region Table: Region_ID, Region Name, Country_ID**
- **Country Table: Country_ID, Country Name**

Key Features:

- **Normalized: Reduces data redundancy and storage.**
- **More complex joins: Query performance can be slower due to multiple joins.**
- **Better data integrity: Good for transactional systems.**
- **Used when: You need to maintain very clean, organized data and are handling complex datasets.**

Comparison Table: Star Schema vs Snowflake Schema

Feature	Star Schema	Snowflake Schema

Structure	**Fact table + directly linked dimensions**	**Fact table + normalized dimension tables**
Data Redundancy	**Higher (due to denormalized dimensions)**	**Lower (due to normalization)**
Query Performance	**Faster**	**Slower (more joins)**
Simplicity	**Easy to understand and use**	**More complex to design and query**
Storage Efficiency	**Less efficient**	**More efficient**
Best Used In	**Reporting & BI tools like Power BI**	**Complex data warehouses with many relationships**
Joins Required	**Fewer**	**More**
Maintenance	**Easier**	**More effort**

Which One to Use in Power BI?

Star Schema is preferred for most Power BI projects.

- **Faster performance**
- **Easier DAX formulas**
- **Cleaner visual relationships**
- **Use Snowflake Schema only when:**
- **You have very complex datasets**
- **You need to maintain high normalization**
- **Data comes from normalized transactional systems and can't be flattened**

56.9 Best Practices for Data Modeling

- **Use meaningful column names and consistent data types.**
- **Prefer star schema layout for faster performance.**
- **Avoid many-to-many relationships if possible; use bridge tables if needed.**
- **Set cross-filter direction to single unless both directions are essential.**
- **Create calculated columns and measures in the data model for better report performance.**
- **Regularly review relationships in Model view to ensure accuracy.**

56.10 Troubleshooting Relationship Issues

- **No Relationship Detected:**
- **Check if columns have matching data types and values.**
- **Ambiguous Relationships:**
- **Multiple paths between tables cause errors; use inactive relationships or redesign model.**
- **Performance Problems:**
- **Reduce many-to-many relationships and avoid bi-directional filters unless needed.**

56.11 Summary

Data modeling and relationships are the foundation of effective Power BI reports. By organizing tables properly and defining clear relationships, you ensure accurate data analysis and smooth report performance. This chapter equips you with the knowledge to design solid data models from simple to complex scenarios.

Chapter 57: Star vs Snowflake Schema

57.1 Introduction

In Business Intelligence (BI) and data warehousing, how we organize data is crucial for fast reporting and effective analysis. To simplify how data is stored and accessed, data modelers use schemas—structured layouts that describe how tables relate to each other.

Two of the most widely used schemas are:

- Star Schema: A simple structure for quick data access.
- Snowflake Schema: A more complex structure focused on data efficiency.

Understanding these two models helps us make better decisions while designing dashboards or handling large datasets.

57.2 What is a Schema in Data Modeling?

A schema is like a blueprint for a database. It shows:

- What tables exist.
- What each table contains.
- How tables are connected to each other (relationships).
- How data flows between tables.

Schemas help database designers ensure:

- Data is structured correctly.
- Data can be queried easily.
- Reports run efficiently.

There are different types of schemas, but in BI, the Star Schema and Snowflake Schema are the most common.

57.3 What is a Star Schema?

A Star Schema gets its name from its shape: it looks like a star. At the center is a fact table, and it is surrounded by dimension tables that connect to it directly.

Example Scenario: Retail Sales

Let9s say a store tracks daily sales. In a star schema:

The Fact Table stores the sales transactions (amount, quantity, etc.).

The Dimension Tables provide information about:

- Customers
- Products
- Stores
- Dates

Fact Table (Sales):

Sale_ID Date_ID Customer_ID Product_ID Store_ID Quantity Total_Amount
Dimension Tables:
Customers Table
Customer_ID Name Gender Age Region
Products Table
Product_ID Product_Name Category Brand
Date Table
Date_ID Date Month Year
Stores Table
Store_ID Store_Name Location Size

Each dimension table directly links to the Sales table using an ID. That9s why it forms a star.

Advantages of Star Schema:

- Easy to understand
- Fast query performance
- Suitable for reporting tools like Power BI, Tableau
- Fewer joins needed (less complexity)

Disadvantages:

- Repeated data (e.g., region names repeated in the customer table)
- Larger storage requirement

57.4 What is a Snowflake Schema?

A Snowflake Schema is an extension of the Star Schema. It normalizes dimension tables into multiple related tables, breaking them into more granular tables.

The result is a structure that looks like a snowflake, with multiple layers of data.

Same Example: Retail Sales

The Fact Table remains the same.

But the Customer dimension is broken into:

- Customers → Regions → Countries

Customer Table:

Customer_ID	Name	Region_ID

Region Table:

Region_ID	Region_Name	Country_ID

Country Table:

Country_ID	Country_Name

Benefits of Snowflake Schema:

- **Eliminates data redundancy (e.g., region and country stored only once)**
- **Uses less disk space**
- **Easier to maintain consistent data**

Drawbacks:

- **More complex**
- **More joins needed to get data (e.g., customer → region → country)**
- **Slower queries compared to star schema**
- **Not as intuitive for beginners**

57.5 Differences Between Star and Snowflake Schema

Feature	Star Schema	Snowflake Schema
Structure	Simple, denormalized	Complex, normalized
Dimension Tables	One-level, wide tables	Multi-level, split into sub-tables
Redundancy	More (data repeated in dimensions)	Less (data is split and stored once)
Storage Space	More required	Less required
Query Performance	Faster (fewer joins)	Slower (more joins)
Maintenance	Easier to query	Harder to query and design
Use Case	Dashboards, OLAP cubes, adhoc analysis	Large enterprise databases needing data integrity

57.6 When to Use Star Schema

Choose Star Schema when:

- **You are building reports in Power BI or Tableau**

Speed is critical

- **Your audience includes business users (non-technical)**
- **The data model needs to be easy to understand**
- **Minor redundancy is acceptable**

57.7 When to Use Snowflake Schema

Choose Snowflake Schema when:

- **You want high data integrity (no repeated data)**

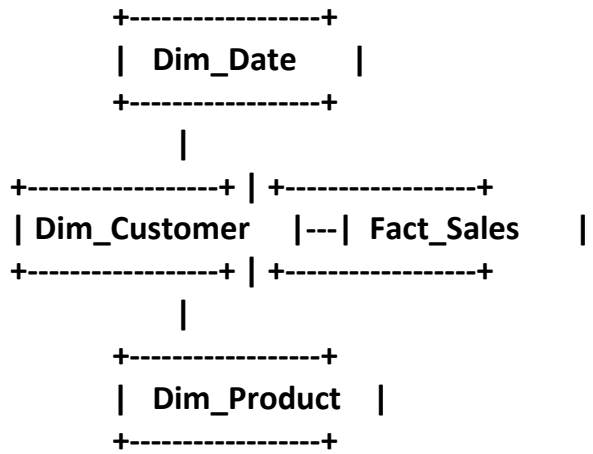
- You are handling large, complex datasets
- Storage space is limited
- Your organization has many dimensions and sub-dimensions
- You have skilled users who understand joins and SQL

57.8 Star Schema vs Snowflake Schema (Summary Table)

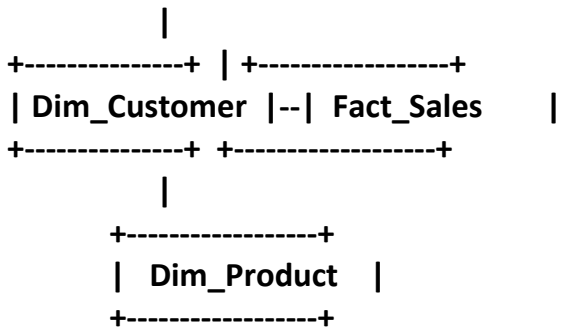
Factor	Star Schema	Snowflake Schema
Simplicity	High	Low
Data Redundancy	High	Low
Query Speed	High	Medium to Low
Storage Requirement	High	Low
Table Joins Needed	Few	Many
Best Use Case	BI Tools, Reports	Complex DB Systems, ER Models
Learning Curve	Easy	Moderate to Difficult

57.9 Visual Comparison

Star Schema (Text-Based Visual):



Snowflake Schema (Text-Based Visual):



57.10 Conclusion

Both Star and Snowflake Schemas are powerful tools for organizing and analyzing data. The right choice depends on your needs:

If you need speed, simplicity, and ease of reporting, go with Star Schema.

If your focus is on storage efficiency and data consistency in a large and complex system, go with Snowflake Schema.

A good data modeler often uses a hybrid approach, combining elements of both schemas based on project requirements.

Chapter 58: Calculated Columns vs Measures

58.1 Introduction

In Power BI, data modeling involves creating additional fields to analyze and visualize data. These fields can be added using DAX (Data Analysis Expressions) and are generally of two types:

- Calculated Columns
- Measures

Both are powerful but serve different purposes. This chapter helps you clearly understand when to use each.

58.2 What is a Calculated Column?

A Calculated Column is a column that is added to a table using a DAX expression. It performs row-wise calculations—meaning the formula runs for every row individually.

Example:

Total_Sales = Sales[Quantity] * Sales[Unit_Price]

This new column Total_Sales stores values for every row in the Sales table.

58.3 What is a Measure?

A Measure is a formula that returns a single value, calculated dynamically based on filters in the report. It doesn't add a column but gives summarized output based on user selection.

Example:

Total_Sales = SUM(Sales[Quantity] * Sales[Unit_Price])

This measure returns total sales based on current filters or slicers applied in the report.

58.4 Key Differences Between Calculated Columns and Measures

Feature	Calculated Column	Measure
Storage	Stored in memory	Not stored
Usage	Tables, filters, groupings	Visuals like cards, charts
Context	Row context	Filter context
Performance	Slower for large data	Faster and optimized
Example	Profit = Sales - Cost	Total Profit = SUM(Sales - Cost)

58.5 When to Use Calculated Columns

Use Calculated Columns when:

- You need to group, filter, or sort by a new value.

- **The logic is row-specific, like calculating age, full name, or profit per row.**

58.6 When to Use Measures

Use Measures when:

- **You want to calculate values that aggregate across rows.**
- **The result should change dynamically based on filters, slicers, and visuals.**
- **You want to optimize memory usage.**

58.7 Example Comparison: Sales Calculation

OrderID	Quantity	Unit Price
1	10	20
2	5	30

Calculated Column:
Total_Price = Sales[Quantity] * Sales[Unit_Price]
Measure:
Total_Sales = SUM(Sales[Quantity] * Sales[Unit_Price])
The calculated column adds a new field to the table; the measure displays total dynamically.

58.8 Performance Consideration

- **Calculated Columns consume memory.**
- **Measures are computed only when needed, making them faster and more memoryefficient.**
- **Best Practice: Prefer measures unless a column is strictly required for filtering or sorting.**

58.9 Visual Examples

- **Tables & Filters → Calculated Columns**
- **Cards, KPIs, Charts → Measures**

58.10 Combination Example

You can use both together:

1. **Column: Profit_Margin = (Sales - Cost) / Sales**
2. **Measure: Avg_Profit_Margin = AVERAGE(Sales[Profit_Margin])**

This gives you row-based logic and aggregated insights together.

58.11 Advanced Example with Dates

Let9s create a column to classify each sale as "Weekend" or "Weekday".
Calculated Column:

Day_Type = IF(WEEKDAY(Sales[Order_Date], 2) > 5, "Weekend", "Weekday")
Key Insight: Row-based classifications require Calculated Columns.

58.12 Advanced Measure Example: Year-over-Year Growth

To calculate year-over-year (YoY) growth:
YoY_Growth =
VAR CurrentYearSales = CALCULATE(SUM(Sales[Total_Sales]), YEAR(Sales[Date]) = 2024)
VAR LastYearSales = CALCULATE(SUM(Sales[Total_Sales]), YEAR(Sales[Date]) = 2023)
RETURN
DIVIDE(CurrentYearSales - LastYearSales, LastYearSales)
Key Insight: If logic needs to respond to filters, use a Measure.

58.13 Memory and Storage Comparison

Aspect	Calculated Column	Measure
Physical Storage	Stored in memory	Not stored
Memory Usage	High	Low
Scalability	Lower	Higher
Model Size	Increases	Minimal impact

Tip: Avoid unnecessary columns for better performance.

58.14 Can You Convert Between the Two?

You can9t convert directly, but logic can be rewritten.
Column:
Profit = Sales[Total_Sales] - Sales[Cost]
Measure:
Total_Profit = SUM(Sales[Total_Sales]) - SUM(Sales[Cost])

58.15 Practical Tips for Choosing Between Them

Situation	Use
Group or sort using new value	Calculated Column
Need totals that change with slicers	Measure

Row-level calculations	Calculated Column
Summary KPIs or ratios	Measure
Optimize performance	Measure
Filtering by that field	Calculated Column
Aggregation across rows	Measure

58.16 Using Both Together

Example:

1. **Create column:**

Profit_Margin = (Sales[Total_Sales] - Sales[Cost]) / Sales[Total_Sales]

2. **Create measure:**

Avg_Profit_Margin = AVERAGE(Sales[Profit_Margin])

58.17 Real-World Use Case: Sales Dashboard

Goal	Type	Formula
Total Sales	Measure	SUM(Sales[Total_Sales])
Product Categories	Column	IF(Product[Name]="iPhone","Mobile","Other")
Cost by Region	Measure	SUM(Sales[Cost])
Weekend Tag	Column	IF(WEEKDAY(Date[OrderDate],2)>5,"Weekend","Weekday")
Monthly Trend	Measure	SUM(Sales[Total_Sales]) by Month

58.18 Common Mistakes and How to Avoid Them

Mistake	Why It9s Bad	Fix
Using column instead of measure	Slows performance	Use measure if no row logic
Too many columns	Heavy model	Use variables or measures
Filtering with measure	Doesn9t work	Use columns in slicers
Repeating same logic	Hard to update	Centralize logic in DAX

58.19 Summary Table: Quick Comparison

Criteria	Calculated Column	Measure
Storage	In table	Recalculated
Context	Row	Filter
Memory	High	Low
Aggregation	No	Yes
Usage	Filters, slicers	Charts, cards
Responsiveness	Static	Dynamic
Performance	Slower	Faster

58.20 Conclusion

- Use Calculated Columns when you need row-level operations or want to filter/group visuals.
- Use Measures for dynamic, aggregated summaries that change based on filters.
- For best performance and flexibility, prefer Measures whenever possible.

Chapter 59: DAX Basics: SUM, AVERAGE, COUNT, DISTINCTCOUNT

59.1 What is DAX?

DAX stands for Data Analysis Expressions. It is a formula language used in Power BI, Excel Power Pivot, and SSAS to define custom calculations and expressions on your data models.

Think of DAX as Excel formulas for data models — but more powerful and dynamic.

59.2 Why Learn DAX Basics First?

Understanding basic DAX functions is essential because they:

- Help you build custom measures and calculated columns
- Allow you to analyze your data more effectively
- Are used in almost every Power BI report

59.3 SUM Function

Syntax: SUM(ColumnName)

Example:

Total_Sales = SUM(Sales[Amount])

This sums up all the values in the Amount column from the Sales table.

Use Cases:

- Total revenue
- Total quantity sold
- Total cost

Limitation: Works only on numeric columns.

59.4 AVERAGE Function

Syntax:

AVERAGE(ColumnName)

Example:

Average_Price = AVERAGE(Sales[Unit_Price])

This calculates the average price from the Unit_Price column.

Use Cases:

- Average revenue
- Average number of items per order

Limitation: Cannot be used for text or non-numeric data.

59.5 COUNT Function

Syntax:

COUNT(ColumnName)

Example:

Order_Count = COUNT(Sales[Order_ID])

This counts the number of rows in the Order_ID column that are not blank.

Use Cases:

- **Count of orders**
- **Count of filled fields**

Important: COUNT only includes non-blank values.

59.6 DISTINCTCOUNT Function

Syntax:

DISTINCTCOUNT(ColumnName)

Example:

Unique_Customers = DISTINCTCOUNT(Sales[Customer_ID])

This counts the number of unique customer IDs.

Use Cases:

- **Number of unique customers**
- **Unique product count**

Best Used When: You need to eliminate duplicates.

59.7 Practical Example Table

Assume this table:

Order_ID	Customer_ID	Amount	Unit_Price
1001	C001	500	50
1002	C002	1000	100
1003	C001	200	20

DAX Measures:

- **Total Amount = SUM(Sales[Amount]) → 1700**
- **Avg Price = AVERAGE(Sales[Unit_Price]) → 56.67**
- **Total Orders = COUNT(Sales[Order_ID]) → 3**
- **Unique Customers = DISTINCTCOUNT(Sales[Customer_ID]) → 2**

59.8.Visual Example in Power BI

Visual	DAX Measure	Explanation
Card	SUM(Sales[Amount])	Shows total revenue
Card	AVERAGE(Sales[Unit_Price])	Avg price per item
Table	COUNT(Sales[Order_ID])	Total number of orders
Table	DISTINCTCOUNT(Sales[Customer_ID])	Unique customer count

59.9.Tips for Beginners

- Use SUM for adding up numeric values.
- Use AVERAGE for understanding trends.
- Use COUNT when you want the number of entries.
- Use DISTINCTCOUNT when you care about uniqueness.

59.10.Common Mistakes

Mistake	Explanation
Using SUM on a text column	SUM only works with numbers
Using COUNT expecting unique count	COUNT includes duplicates
Using DISTINCTCOUNT on numbers expecting sum	It only counts unique values, not adds them

59.11.Deep Dive: SUM() Function

Total_Revenue = SUM(Sales[Total])

- Adds up values from the Total column
- Ignores blanks
- Works only on numeric values

Common Mistake: Trying to sum text or boolean columns.

59.12 Deep Dive: AVERAGE() Function

Average_Price = AVERAGE(Sales[Unit_Price])

- Helps in comparing average performance across products, customers, etc.
- Ignores blank rows

59.13 Deep Dive: COUNT() Function

Order_Count = COUNT(Sales[Customer_ID])

- Counts only non-blank entries
- Skips null values

Other variants:

- COUNTA() – Counts everything except blank
- COUNTBLANK() – Counts only blank rows
- COUNTROWS() – Counts all rows in the table

59.14 Deep Dive: DISTINCTCOUNT() Function

Unique_Customers = DISTINCTCOUNT(Sales[Customer_ID])

- Counts unique values only
- Useful for removing duplicates

59.15 Combining SUM with CALCULATE()

India_Sales = CALCULATE(SUM(Sales[Total]), Sales[Country] = "India")

- Uses CALCULATE() to apply filter
- Very powerful for segmented analysis

59.16.Creating KPIs with DAX

KPI	DAX Formula
Total Sales	SUM(Sales[Total])
Avg Sale/Customer	SUM(Sales[Total]) / DISTINCTCOUNT(Sales[Customer_ID])
Orders/Customer	COUNT(Sales[Order_ID]) / DISTINCTCOUNT(Sales[Customer_ID])

59.17.Filter Context vs Row Context

Context		Meaning
Row Context		Works on one row at a time
Filter Context		Works with visuals and slicers

- SUM, AVERAGE, etc., respect the filter context in visuals.

59.18.Summary Table

Function	Works On	Ignores Blanks	Removes Duplicates	Use Case
SUM	Numbers	Yes	No	Total sales
AVERAGE	Numbers	Yes	No	Avg price
COUNT	Any column	Yes	No	Count of filled rows
DISTINCTCOUNT	Any column	Yes	Yes	Unique customers

59.19.Pro Tips

- Avoid using COUNT on ID columns expecting uniqueness.
- Prefer DISTINCTCOUNT when analyzing unique entities.
- Use CALCULATE with SUM/AVERAGE for custom filtering.

59.20 Final Thoughts

DAX functions like SUM, AVERAGE, COUNT, and DISTINCTCOUNT are the building blocks of Power BI analysis. Mastering these enables you to:

- Build dynamic reports
- Create meaningful KPIs
- Answer business questions

Chapter 60: Logical and Date Functions in DAX

60.1 Introduction

DAX (Data Analysis Expressions) is the formula language used in Power BI, Excel Power Pivot, and Analysis Services. It includes a variety of functions to perform calculations and return information. Two of the most commonly used categories are Logical and Date functions. These functions are crucial when building dynamic, context-aware calculations that respond to filters, time, and user interactions.

60.2 Logical Functions Overview

Logical functions allow you to evaluate expressions and return different results depending on the outcome. They are the backbone of conditional logic in DAX.

Key Logical Functions:

Function	Description
IF	Returns one value if a condition is TRUE and another if FALSE
SWITCH	Evaluates an expression against a list of values and returns a result based on the first match
AND	Returns TRUE if all arguments are TRUE
OR	Returns TRUE if any argument is TRUE
NOT	Returns the opposite (negation) of a logical expression

60.3 IF Function

Syntax:

IF(<logical_test>, <value_if_true>, <value_if_false>)

Explanation: The IF function checks whether a condition is TRUE or FALSE and returns corresponding results.

Example:

Status = IF(Sales[Total_Sales] > 1000, "High", "Low")

This example checks if each sale is more than 1000. If so, it returns "High", else "Low".

Use Cases:

- Label rows for segmentation
- Show messages or classifications in visuals
- Customize visuals based on logic

Nested IF Example:

Category = IF(Sales[Total_Sales] > 1000, "High", IF(Sales[Total_Sales] > 500, "Medium", "Low"))

60.4 SWITCH Function

Syntax:

SWITCH(<expression>, value1, result1, value2, result2, ..., else_result)

Explanation: SWITCH is a cleaner alternative to multiple nested IFs. It checks the expression against values sequentially.

Example:

Rating = SWITCH(Sales[Score], 5, "Excellent", 4, "Good", 3, "Average", "Poor")

Use Cases:

- **Assigning grades, ratings, or categories**
- **Simplifying multiple IF conditions**
- **Enhance readability of complex logic**

60.5 AND, OR, NOT Functions

These are used to combine multiple logical conditions.

AND Example:

IF(Sales[Amount] > 500 && Sales[Region] = "East", "Valid", "Invalid") OR Example:

IF(Sales[Region] = "East" || Sales[Region] = "West", "Target", "Other")

NOT Example:

IF(NOT(Sales[Discount] > 0), "No Discount", "Discounted")

Use Cases:

- **Combine multiple filters in one condition**
- **Filter or categorize data dynamically**
- **Exclude specific logic flows**

60.6.Nested Logical Functions

You can nest logical functions inside each other for more complex conditions.

Example:

IF(Sales[Total_Sales] > 1000, IF(Sales[Region] = "West", "High-West", "High-Other"), "Low")

60.7.Date Functions Overview

Date functions are essential for time-based calculations, period comparisons, and calendarbased analytics.

Key Date Functions:

Function	Description
TODAY	Returns current system date (no time)
NOW	Returns current system date and time

YEAR	Returns year from a date
MONTH	Returns month from a date
DAY	Returns day from a date
WEEKDAY	Returns the day number of the week
DATEDIFF	Calculates difference between two dates in selected interval
EOMONTH	Returns the last day of the month
DATEADD	Adds or subtracts time from a date
SAMEPERIODLASTYEAR	Compares current period to the same period last year

60.8.TODAY and NOW Functions

TODAY Example:

IF(Sales[Order_Date] = TODAY(), "Today", "Earlier") NOW Example:

IF(Sales[Timestamp] < NOW(), "Past", "Future")

Use Cases:

- **Filter today9s transactions**
- **Calculate deadlines or age of data**
- **Create real-time dashboards**
- **Schedule reminders or alerts**

60.9 YEAR, MONTH, DAY Functions

Extract parts of a date to create filters or new columns.

Examples:

Year = YEAR(Sales[Order_Date])

Month = MONTH(Sales[Order_Date])

Day = DAY(Sales[Order_Date])

Use Cases:

- **Create Yearly, Monthly, Daily visuals**
- **Group data based on calendar units**
- **Filter by year or month for trend analysis**

60.10 WEEKDAY Function

Syntax:

WEEKDAY(<date>, <return_type>)

- **return_type 1: Sunday = 1, Saturday = 7 (default)**

- **return_type 2: Monday = 1, Sunday = 7 (European style)**

Example:

DayType = IF(WEEKDAY(Sales[Order_Date], 2) > 5, "Weekend", "Weekday")

Use Cases:

- **Filter business days vs weekends**
- **Track patterns by weekday**
- **Adjust marketing campaigns by weekday behavior**
- **Identify low-traffic or high-traffic days**

60.11 DATEDIFF Function

Syntax:

DATEDIFF(<start_date>, <end_date>, <interval>)

- **Interval options: SECOND, MINUTE, HOUR, DAY, MONTH, QUARTER, YEAR**

Example:

DeliveryDays = DATEDIFF(Sales[Order_Date], Sales[Delivery_Date], DAY)

Use Case:

- **Analyze processing or delivery times**
- **Track customer response time**
- **Calculate age or duration**
- **Compute employee tenure**

60.12 EOMONTH Function

Syntax:

EOMONTH(<start_date>, <months>)

Example:

MonthEnd = EOMONTH(Sales[Order_Date], 0)

Use Case:

- **Group records by month-end**
- **Perform monthly summaries**
- **Financial reporting alignment**
- **Predict billing or revenue cutoffs**

60.13 DATEADD Function

Syntax:

DATEADD(<dates>, <number_of_intervals>, <interval>)

Example:

SalesLastMonth = CALCULATE(SUM(Sales[Total_Sales]), DATEADD(Sales[Order_Date], -1, MONTH))

Use Case:

- **Period-over-period comparisons**
- **Forecasting future performance**

- Trend analysis across intervals
- Yearly or quarterly comparisons

60.14 SAMEPERIODLASTYEAR Function

Syntax:

SAMEPERIODLASTYEAR(<dates>)

Example:

SalesLY = CALCULATE(SUM(Sales[Total_Sales]), SAMEPERIODLASTYEAR(Sales[Order_Date]))

Use Case:

- Compare performance to the same time last year
- Measure growth rate
- Business planning and KPIs
- Seasonal trend analysis

60.15.Practical Scenario: Time Intelligence

Use these functions together to solve real-world business problems.

Goal	DAX Formula
Sales Last Year	CALCULATE(SUM(Sales[Total]), SAMEPERIODLASTYEAR(Sales[Order_Date]))
Sales This Month	CALCULATE(SUM(Sales[Total]), FILTER(Sales, MONTH(Sales[Order_Date]) = MONTH(TODAY())))
Sales on Weekend	CALCULATE(SUM(Sales[Total]), FILTER(Sales, WEEKDAY(Sales[Order_Date], 2) > 5))

60.16.Summary Table

Function	Category	Usage
IF, SWITCH	Logical	Decision-making logic
AND, OR, NOT	Logical	Combine multiple conditions
TODAY, NOW	Date	Get current date/time for filtering
YEAR, MONTH, DAY	Date	Break dates into parts
WEEKDAY	Date	Identify weekend/weekday
DATEDIFF	Date	Time difference between dates
EOMONTH	Date	End-of-month calculations
DATEADD	Date	Move back/forward in time
SAMEPERIODLASTYEAR	Date	Compare year-over-year

60.17.Conclusion

Logical and Date functions are powerful tools in DAX that allow analysts to:

- **Build complex decision trees**
- **Segment data based on time logic**
- **Compare data across timeframes like months and years**
- **Enhance reports with dynamic date-based filtering and analysis**

Chapter 61: Time Intelligence: YTD, MTD, and QTD

61.1 Introduction to Time Intelligence

Time Intelligence in DAX is the ability to analyze data across different time periods—like comparing monthly sales over years, accumulating yearly totals, or analyzing quarterly performance.

Power BI includes special DAX functions specifically designed for time-based analysis. These functions rely on a Calendar Table (Date Table) that includes a continuous range of dates and is marked as a Date Table in Power BI.

Time Intelligence enables:

- Year-to-date (YTD), month-to-date (MTD), and quarter-to-date (QTD) analysis
- Comparing current vs previous periods
- Creating cumulative totals
- Time-based filtering and trend analysis

61.2 Importance of a Proper Date Table

Time Intelligence functions like TOTALYTD, TOTALMTD, and SAMEPERIODLASTYEAR only work correctly with a dedicated Date Table. This table must:

Include a continuous range of dates
Be marked as the official date table
Be related to your data model

Example of a minimal Date Table:

DateTable = CALENDAR(DATE(2019,1,1), DATE(2030,12,31))

Then mark it as the Date Table in Power BI by using:

Model View → Select Table → Mark as Date Table → Choose Date Column

61.3.Key Time Periods

Period	Description
YTD (Year-To-Date)	Accumulates data from January 1st to the current date of the year
MTD (Month-To-Date)	Accumulates data from the 1st of the current month to today
QTD (Quarter-To-Date)	Accumulates data from the start of the quarter to the current date

61.4.TOTALYTD Function

Syntax:

TOTALYTD(<expression>, <dates>, [<filter>], [<year_end_date>])

expression: The calculation to perform (like SUM(Sales[Amount]))

- dates: The date column from the date table

- **filter (optional): Any additional filter**
- **year_end_date (optional): Use a non-December fiscal year-end**

Example:

Sales_YTD = TOTALYTD(SUM(Sales[Total_Sales]), 'Date'[Date])

Use Cases:

- **Cumulative yearly revenue tracking**
- **Progress toward yearly sales targets**
- **Comparing YTD values vs previous years**

61.5 TOTALMTD Function

Syntax:

TOTALMTD(<expression>, <dates>, [<filter>])

Example:

Sales_MTD = TOTALMTD(SUM(Sales[Total_Sales]), 'Date'[Date])

Use Cases:

- **Track sales from the start of the current month**
- **Understand monthly spending trends**
- **Drive short-term decisions and goals**

61.6 TOTALQTD Function

Syntax:

TOTALQTD(<expression>, <dates>, [<filter>])

Example:

Sales_QTD = TOTALQTD(SUM(Sales[Total_Sales]), 'Date'[Date])

Use Cases:

- **Evaluate quarterly business performance**
- **Align reporting with quarterly goals**
- **Track revenue, expenses, or KPIs within a quarter**

61.7 DATESYTD Function

Syntax:

DATESYTD(<dates>, [<year_end_date>])

Example:

CumulativeSales = CALCULATE(SUM(Sales[Total_Sales]), DATESYTD('Date'[Date]))

Explanation:

Returns a table of dates from the start of the year to the last visible date in the filter context. It9s more flexible than TOTALYTD.

Use Cases:

- **Use in advanced filters or combinations with CALCULATE**
- **Complex visualizations and customized filters**

61.8 Time Intelligence with CALCULATE

You can use basic DAX functions like YEAR(), MONTH(), or WEEKNUM() with CALCULATE to create your own period-based filters.

Example:

```
CurrentYearSales = CALCULATE(
  SUM(Sales[Total_Sales]),
  YEAR('Date'[Date]) = YEAR(TODAY())
PreviousYearSales = CALCULATE(
  SUM(Sales[Total_Sales]),
  YEAR('Date'[Date]) = YEAR(TODAY()) - 1
```

61.9 SAMEPERIODLASTYEAR

Syntax:

SAMEPERIODLASTYEAR(<dates>)

Example:

Sales_Last_Year = CALCULATE(SUM(Sales[Total_Sales]), SAMEPERIODLASTYEAR('Date'[Date]))

Use Cases:

- Compare current period to same period last year
- Highlight yearly growth trends
- Measure seasonal patterns

61.10 ParallelPeriod Function (Bonus)

Syntax:

PARALLELPERIOD(<dates>, <number_of_intervals>, <interval>)

Example:

Sales_Last_Quarter = CALCULATE(SUM(Sales[Total_Sales]), PARALLELPERIOD('Date'[Date], 1, QUARTER))

Use Case:

- Compare a quarter to the previous quarter
- Time shift values without relying on SAMEPERIOD functions

61.11Time Intelligence: Practical Use Cases

Goal	DAX Formula
YTD Revenue	TOTALYTD(SUM(Sales[Revenue]), 'Date'[Date])
MTD Orders	TOTALMTD(COUNT(Sales[Order_ID]), 'Date'[Date])
QTD Profit	TOTALQTD(SUM(Sales[Profit]), 'Date'[Date])

YTD vs Last Year YTD	TOTALYTD(SUM(Sales[Amount]), 'Date'[Date]) - TOTALYTD(Sales_Last_Year)
Same Period Last Year Sales	CALCULATE(SUM(Sales[Total_Sales]), SAMEPERIODLASTYEAR('Date'[Date]))

61.12.Best Practices for Time Intelligence

- Always use a dedicated Date Table
- Always mark it as Date Table in Power BI
- Use CALCULATE to modify filter context when needed
- Be cautious of missing dates—use continuous calendar
- Ensure relationships are correctly defined between fact and date tables

Summary Table

Function	Description
TOTALYTD	Calculates values from beginning of year to current date
TOTALMTD	Calculates values from beginning of month to current date
TOTALQTD	Calculates values from start of quarter to current date
DATESYTD	Returns date range for use in filters or CALCULATE
SAMEPERIODLASTYEAR	Compares current period to same period last year
PARALLELPERIOD	Shifts date context by defined intervals (months, years)

61.13.Conclusion

Time Intelligence functions in DAX provide powerful ways to analyze data over time. Whether you're tracking YTD sales or comparing current performance to last year, these functions help build dynamic, interactive, and insightful Power BI dashboards.

By mastering these tools and combining them with filters and visualizations, you'll unlock rich time-based insights to drive smarter business decisions.

Chapter 62: Advanced Data Cleaning in Power Query

62.1 Introduction

Power Query in Power BI is a powerful data transformation tool that enables users to clean, reshape, and prepare data before loading it into the data model. While basic cleaning tasks like removing columns or changing data types are common, advanced data cleaning helps tackle more complex data issues such as handling inconsistent formats, removing duplicates, splitting or merging columns, pivoting/unpivoting data, and managing nulls more efficiently.

62.2 Removing Duplicates with Advanced Options

Removing duplicates ensures that your data is accurate and not inflated by repeated entries.

Steps:

1. **Select the column(s) you want to check for duplicates.**
2. **Go to Home>Remove Rows>Remove Duplicates.**
3. **To remove rows based on multiple columns, select them together before applying.**

Advanced Tip:

Use Group By before removing duplicates if you need to preserve one record per group with summary logic like max, min, or average.

62.3 Filtering Rows Based on Custom Conditions

Power Query allows complex conditional filtering using custom criteria.

Example:

- **Remove rows where <Amount= is less than 0.**
- **Filter text containing certain keywords.**

Steps:

1. **Select the column.**
2. **Click the drop-down filter icon.**
3. **Choose Number Filters>Greater Than, or Text Filters>Contains, etc.**
4. **For advanced filters, use Add Column > Conditional Column to create a flag, and then filter based on it.**

62.4 Handling Nulls and Missing Data

Null values can break calculations or visualizations if not handled.

Options:

- **Replace Values: Replace null with a default value.**
- **Remove Rows: Eliminate rows where specific columns have null.**
- **Fill Down/Up: Fill missing values using the value above or below.**
- **Conditional Column: Create custom rules to replace or flag nulls.**

Example:

If [Region] is null, then "Unknown", else [Region]

62.5 Splitting Columns with Delimiters and Patterns

You can split a column into multiple parts using delimiters (like comma, space, hyphen) or based on length.

Steps:

1. **Select the column.**
2. **Go to Transform>Split Column.**
3. **Choose By Delimiter, By Number of Characters, or By Positions.**

Use Case:

- **Splitting "John Doe" into First Name and Last Name.**
- **Breaking date strings like "20240608" into year, month, day.**

62.6 Merging and Combining Columns

Merging columns is useful when you need a combined identifier or display-friendly value.

Steps:

1. **Select multiple columns (e.g., First Name and Last Name).**
2. **Go to Transform>Merge Columns.**
3. **Choose a separator like space, comma, or custom character.**

Use Case:

Merge "City" and "State" for full location: "Mumbai, MH".

62.7 Using Conditional Columns for Logic-Based Cleaning

Conditional columns allow you to apply logic directly in Power Query without DAX.

Steps:

1. **Go to Add Column>Conditional Column.**
2. **Set conditions like:**

If [Score] >= 90 then "Excellent"
Else If [Score] >= 70 then "Good"
Else "Needs Improvement"

Use Case:

Labeling records based on thresholds
Creating flags for inclusion/exclusion

62.8 Using Column from Examples

This feature learns patterns from your manual input and applies it automatically.

Steps:

1. **Go to Add Column>Column From Examples>From Selection.**
2. **Start typing a desired value based on existing data.**
3. **Power Query will predict and apply a formula.**

Use Case:

- **Extract domain from email addresses.**
- **Create abbreviated codes or initials.**

62.9 Pivoting and Unpivoting Columns

Power Query allows you to reshape data for better analysis.

Unpivot Columns

Turns multiple columns into rows.

Steps:

1. **Select the columns to unpivot.**
2. **Go to Transform>Unpivot Columns.**

Use Case:

Convert month-wise sales columns into one <Month= column with corresponding values.

Pivot Columns

Turns row data into columns.

Steps:

1. **Select the key column.**
2. **Go to Transform>Pivot Column.**
3. **Choose the values column and aggregation.**

Use Case:

Create a summary view with categories as columns.

62.10 Grouping Data for Aggregation

Grouping allows summarizing data in Power Query, similar to SQL GROUP BY.

Steps:

1. **Go to Home>Group By.**
2. **Select the grouping column and the aggregation function (sum, count, avg, etc.).**

Use Case:

Total sales by region.

Count of records by category.

62.11 Text Standardization: Trim, Clean, Upper, Lower, Proper

Power Query offers several text-cleaning tools.

Function	Description
Trim	Removes extra spaces
Clean	Removes non-printable characters
Upper	Converts text to UPPERCASE
Lower	Converts text to lowercase

Proper	**Capitalizes first letter of each word**

Steps:

1. **Select the column.**
2. **Go to Transform> choose the appropriate function.**

Use Case:

- **Normalize names and addresses.**
- **Clean inconsistent data imports.**

62.12 Removing Errors from Queries

Errors can occur due to invalid data types, division by zero, etc.

Steps:

- **Go to Home>Remove Rows>Remove Errors.**
- **Use Keep Errors to diagnose issues.**
- **Use Try&Otherwise in Advanced Editor to handle errors programmatically.**

Example:

try [Amount] / [Count] otherwise 0

62.13 Custom Column with M Code

For highly customized cleaning, use M language to write your own transformation logic.

Steps:

1. **Go to Add Column>Custom Column.**
2. **Write expressions using M code.**

Example:

if Text.StartsWith([City], "New") then "East" else "Other"

62.14 Practical Example: Cleaning Customer Data

Scenario:

You import customer data with inconsistent name formats, missing phone numbers, and duplicate entries.

Steps:

- **Trim and clean name fields.**
- **Split Full Name into First and Last Name.**
- **Replace null in Phone with "Not Available".**
- **Remove duplicate Customer IDs.**
- **Create a conditional column for high-value customers.**

62.15.Summary Table

Task	Power Query Feature
Remove duplicates	Remove Rows > Remove Duplicates
Filter by conditions	Filter Rows / Conditional Column
Handle nulls	Replace / Remove / Fill
Split/Merge columns	Transform > Split / Merge
Pivot/Unpivot	Transform > Pivot / Unpivot
Text cleanup	Trim, Clean, Case functions
Error handling	Remove Errors / Try-Otherwise
Custom logic	Conditional / Custom Column

62.16.Conclusion

Advanced data cleaning in Power Query is critical for reliable and high-quality analysis. By mastering these tools, you can automate complex transformations, reduce manual corrections, and create cleaner, more professional Power BI dashboards. Clean data ensures accurate insights, making your reports trustworthy and impactful.

In the next chapter, we9ll explore Data Transformation using Power Query M Language to take your data preparation skills even further.

Chapter 63. Column Splits, Merges, and Custom Columns

63.1 What Are Column Transformations in Power BI?

In Power BI, especially through Power Query Editor, we often need to reshape the data to make it suitable for analysis. Three very common and powerful operations for transforming your columns are:

- Splitting columns: Breaking one column into two or more separate columns.
- Merging columns: Combining two or more columns into one.
- Creating custom columns: Generating new columns based on logic, math, or text manipulation. These are vital when:
- Data is not in a clean, analysis-ready format.
- Important information is combined or scattered across fields.
- You want to enhance data with new insights or categorizations.

63.2 Splitting Columns

Splitting columns is necessary when one column contains multiple pieces of data joined together, such as:

- Full names ("John Smith")
- Addresses ("New York, NY")
- Product codes ("ABC-1234-X")

63.2.1 Split by Delimiter

A delimiter is a character that separates values in a string (e.g., comma ,, space " ", dash -, or custom symbols).

Steps:

1. Go to Power Query Editor.
2. Select the column you want to split.
3. From the Home or Transform tab, click Split Column → By Delimiter.
4. Choose a delimiter:

Common: Comma, Space, Semicolon, Colon, Tab

Custom: You can type your own (e.g., |, @)

5. Select how to split:
    - Left-most delimiter: splits at the first occurrence o Right-most delimiter: splits at the last occurrence
    - Each occurrence: splits multiple times, producing more columns

Example:

- Column: "John Smith"
- Split by space " "

Output:

- **First Column: John o Second Column: Smith**

Use Case: Separating first and last names.

63.2.2 Split by Number of Characters

Used when the data has fixed-width formatting.

Steps:

1. **Select the column.**
2. **Go to Split Column → By Number of Characters.**
3. **Specify how many characters to split after.**
4. **Choose whether to:**
   o Split once (one-time split) o Repeatedly split at that interval (for multiple equal chunks)

Example:

Column: "20240608" □ Split every 4 characters □ Result:

- **Column1: 2024 o Column2: 0608**

Use Case: Extracting Year and MonthDay from date string.

63.2.3 Split by Position

This is useful when you know the exact character positions where you want to split.

Steps:

1. **Select column.**
2. **Choose Split Column → By Positions.**
3. **Enter split positions, e.g., 2, 5.**

Example:

Input: "AB12345Z" □ Positions: 2 and Output:

- **Column1: AB o Column2: 123**

Column3: 45Z

63.3 Merging Columns

Merging columns allows you to combine multiple fields into a single column using a separator.

Common Use Cases:

- **Combine First and Last Name**
- **Create full addresses (City + State)**

- **Build composite keys (e.g., Region-Year)**

Steps:

1. **Select the columns to merge (hold Ctrl for multiple).**
2. **Click Transform → Merge Columns.**
3. **Choose a separator:**
   - **Space, Comma, Colon, Custom**
4. **Enter a name for the new column.**

Example:

First Name	Last Name
John	Smith

Merge with space " " → Result: "John Smith"

63.4 Creating Custom Columns

A custom column is a new column derived by applying logic to one or more existing columns. You can use:

- **Basic arithmetic**
- **Text functions**
- **Logical comparisons**
- **Nested conditions (if-else)**

63.4.1 Creating with GUI (Custom Column Window)

Steps:

1. **Go to Add Column → Custom Column.**
2. **Give your new column a name.**
3. **Use the formula box to define logic using M syntax.**

Example 1: Calculate Profit

[Revenue] - [Cost] Example 2: Classify Sales

if [Sales] > 10000 then "High" else "Low"

63.4.2 Using Conditional Column (No Code Needed)

This tool is for non-coders. It provides a user-friendly interface to build if-else logic.

Steps:

1. **Go to Add Column → Conditional Column.**
2. **Choose column, condition (e.g., equals, greater than), and result values.**

Example:

- **If [Status] = "Open" → "Pending"**
- **If [Status] = "Closed" → "Complete"**

- **Else → "Unknown"**

This results in a new column with categories based on the value of Status.

63.4.3 Writing M Code Manually (Advanced)

M is the Power Query Formula Language. It allows complex custom column creation.

Example 1: Add 10% Tax

[Amount] * 1.1

Example 2: Combine Columns With Formatting

[City] & ", " & [State]

Example 3: Multi-level Category if [Score] >= 90 then "Excellent" else if [Score] >= 75 then "Good" else "Average"

You can also use text and date functions like:

Text.Upper([Name])

Date.Year([OrderDate])

63.5 Real-World Scenario: Cleaning Email Data

You have a column: email_address

email_address
john.smith@gmail.com
alice@yahoo.com

Goal: Extract:

- **Username: part before @**
- **Domain: part after @**

Steps:

1. **Split column by delimiter @.**
2. **Result:**
   - **Username: john.smith o Domain: gmail.com**

This is extremely useful in customer analysis, grouping users by email provider.

63.6.Summary Table

Task	Power Query Feature
Break name into first/last	Split by Delimiter
Separate code by characters	Split by Number of Characters

Join name and city	**Merge Columns**
Calculate total cost	**Custom Column**
Add logic without coding	**Conditional Column**
Advanced math/text logic	**M Code**

63.7.Tips for Working with Columns in Power Query

- **Always rename new columns after a split or merge for clarity.**
- **Use Preview Window to verify transformations before applying.**
- **Use Error handling (e.g., try...otherwise) in custom columns when data may be missing or invalid.**
- **Keep column names short and clear for easy use in reports.**

63.8 Conclusion

Mastering column transformations such as splits, merges, and custom column creation is essential for preparing data in Power BI. These techniques help convert messy or unstructured data into a format ready for rich visual analysis.

In the next chapter, we9ll go deeper into Power Query M Language: Syntax and Common Functions, to unlock even more powerful transformations.

Chapter 64: Report Design: Layouts, Bookmarks, Buttons

64.1 Introduction to Report Design in Power BI

Power BI isn9t just about loading data and showing charts—presentation and user experience matter greatly. Report design focuses on:

- Clarity: Making visuals easy to read and interpret.
- Usability: Allowing users to interact with the data efficiently.
- Storytelling: Guiding users through key insights.

You achieve this using three major features:

1. Layouts – How visuals are arranged and presented.
2. Bookmarks – To save specific views and visual states.
3. Buttons – To allow interactivity, navigation, and actions.

64.2 Layouts in Power BI

64.2.1 Why Layouts Matter

A good layout ensures that your audience:

- Immediately sees the most important insights
- Does not feel overwhelmed or confused
- Can easily navigate the report

Power BI reports are visual presentations. Just like a slide deck or webpage, they benefit from:

- Alignment
- Grouping
- Consistent color usage
- Proper spacing

64.2.2 Key Layout Concepts

a) Canvas Settings

- You can change the page size (16:9, letter, custom)
- Use a background color or image for branding
- Gridlines and Snap-to-grid help align visuals

b) Sections & Grouping

Organize your report into logical areas:

- Header: Contains title, logos, date filters
- Left Panel: Slicers (e.g., by Year, Region)
- Main Visual Area: Charts, tables, KPIs
- Footer (optional): Explanatory notes or navigation

c) Visual Consistency

Maintain uniform:

- **Fonts**
- **Font sizes**
- **Visual colors (use a fixed color theme)**
- **Title formats for each visual**

64.2.3 Tips for Layout Design

Element	Tips
White Space	Leave empty space to improve readability. Don9t overcrowd visuals.
Grouping	Use shapes or background boxes to visually group related charts.
Navigation Area	For complex reports, create a side menu or tab-style layout.
Icons & Branding	Use company logos or small icons (insert → image or shapes).

64.3 Bookmarks in Power BI

64.3.1 What Are Bookmarks?

A bookmark in Power BI captures a specific state of the report, such as:

- **Which visuals are visible**
- **What filters and slicers are applied**
- **What page you9re on**
- **Drill level or sort order**

You can return to this state later using the Bookmarks Pane or Buttons.

64.3.2 Real-World Bookmark Scenarios

1. **Switch Between Views**
   Example: Switch from bar chart view to table view of the same data.
2. **Show/Hide Slicers or Filters**
   Example: Add a <More Filters= button to expand hidden slicers.
3. **Step-by-Step Storytelling**

Example: Walk users through a business story by showing one insight at time.

4. **Custom Drillthrough**
   Simulate navigating to a detailed view and then returning back.

64.3.3 How to Create a Bookmark – Step-by-Step

1. **Design the desired view (set filters, hide/show visuals)**

2. **Go to View > Bookmarks Pane**
3. **Click Add Bookmark**
4. **Rename it clearly (e.g., <Show Table View=)**
5. **Right-click the bookmark:**
    - **Choose if you want to include Data, Display, or Current Page**

64.3.4 Using the Selection Pane with Bookmarks

- **Open the Selection Pane**
- **Rename and show/hide visuals as needed For each bookmark:**
    - **Save a different combination of visibility o Example: Bookmark 1 shows a line chart, Bookmark 2 shows a bar chart**

64.3.5 Organizing Bookmarks

- **Groups: Bundle bookmarks into logical folders (e.g., Navigation, Filters, Views)**
- **Ordering: Drag and drop to reorder the sequence**
- **Play Bookmarks: Present like a slideshow (View > Bookmark Navigator)**

64.4 Buttons in Power BI

64.4.1 What Are Buttons?

Buttons are clickable objects that trigger actions like:

- **Navigating to another page**
- **Switching between bookmarks**
- **Resetting slicers**
- **Opening external links**

They allow you to build your own navigation system—like a mini web app.

64.4.2 Types of Buttons

Button Type	Use
Back	Used on drillthrough pages to return
Bookmark	Linked to a bookmark
Page Navigation	Go to another report page
Q&A	Open a text box for natural language queries
Reset	Reset all slicers or views

Blank	**Fully customizable button**

64.4.3 How to Add and Configure a Button

1. **Go to Insert > Buttons**
2. **Choose a button type or <Blank=**
3. **Resize and place on the canvas**
4. **Format:**

o Button Text o Background and font color o Rounded corners or shadow o Hover effects

64.4.4 Assigning an Action

1. **Select the button**
2. **In the Visualizations Pane, turn Action = ON**
3. **Choose the Action Type: o Bookmark**
   - **Page Navigation**
   - **Q&A**
   - **Web URL**
4. **Select the Target (bookmark name, page name, or URL)**

64.4.5 Button States

Each button supports three visual states:

State	Purpose
Default	**Normal appearance**
On Hover	**Style when mouse is over it**
On Press	**Style when clicked**

You can use different fonts, colors, and effects to show interactivity clearly.

64.5 Advanced Example: Toggle View Using Bookmarks and Buttons

Scenario:

Toggle between a Bar Chart and Data Table on a single page.

Step-by-Step:

1. **Place both visuals on top of each other.**
2. **Hide the table using the Selection Pane.**
3. **Save a bookmark called <Chart View=**
4. **Now hide the chart, show the table, and save another bookmark as <Table View=**
5. **Add two buttons:**

- **<Show Table= (linked to the <Table View= bookmark)**
- **<Show Chart= (linked to the <Chart View= bookmark)**

6. **You9ve created a toggle experience on one page.**

64.6 Use of Selection and Layer Pane

Selection Pane:

- **Show/hide specific visuals**
- **Rename visuals for easier management with bookmarks**

Layer Order:

- **Bring visuals forward or send to back**
- **Useful when layering shapes, buttons, and charts**

64.7.Best Design Practices

Tip	Why It Matters
Limit visuals per page	Avoid clutter, improve performance
Use consistent themes	Keeps report professional
Use readable fonts	Arial, Segoe UI, or Calibri are best
Use tooltips and titles	Help guide users
Optimize for mobile	Consider responsiveness if mobile usage is expected
Use navigation buttons	Make large reports easier to use

64.8.Summary Table

Feature	Description	Purpose
Layout	Arrangement of visuals and text on the report page	Improves readability and storytelling
Bookmarks	Save and recall visual states	Enable interactivity and userspecific views
Buttons	Create navigation and actions	Makes reports dynamic and userfriendly

Chapter 65: Drillthrough and Conditional Formatting in Power BI

65.1 Introduction

Power BI offers features that make dashboards not just beautiful, but also intelligent and interactive. Two such key features are:

- **Drillthrough – A navigation tool that lets users explore more details about a data point by switching to another report page that filters based on that selection.**
- **Conditional Formatting – Automatically changes colors, fonts, icons, or even images in visuals based on data values, drawing attention to key insights and anomalies.**
- **Both features greatly improve user experience, dashboard clarity, and insight discovery.**

65.2 Drillthrough in Power BI

65.2.1 What is Drillthrough?

Drillthrough allows users to right-click on a data item (such as a customer, product, or region) and open a dedicated page filtered just for that item. It's a way to move from summary-level data to granular insights without crowding a single page with all details.

Think of it as zooming into a specific part of the data story.

65.2.2 Real-World Analogy

Imagine an online store9s dashboard. On the homepage, you see sales by all regions. You want to know why sales in 'South India' are dropping. With drillthrough, you right-click on 'South India' and open a page that shows:

- **Monthly trends**
- **Top-selling products**
- **Customer complaints**
- **Delivery delays**
- **&all automatically filtered for South India.**

65.2.3 How Drillthrough Works – Step-by-Step Guide

Step 1: Create a Drillthrough Page

1. **Add a new page in Power BI.**
2. **In the Visualizations pane, look for Drillthrough filters section (bottom of Fields panel).**
3. **Drag a field (like Customer, Product, or Region) into it.**

This tells Power BI: <Only show this page when someone drills into this field.=

Step 2: Add Visuals

- **Add cards, charts, tables, etc., that help analyze the selection.**
- **You can add slicers or filters on top of the drillthrough.**

Step 3: Add a Back Button

1. **Insert → Button → Back**
2. **In the visual settings pane, enable the action and set it to Back.**

This lets users return to the summary page smoothly.

65.2.4 Important Notes

- **The drillthrough page is hidden from navigation unless a user triggers it.**
- **You can only drill from a visual that contains the same field used in the drillthrough filter.**

65.2.5 Drillthrough with Multiple Fields

You can use multiple fields in the drillthrough filter, such as:

- **Product Category**
- **Salesperson**
- **Year**

In this case, Power BI filters the page by all fields passed in during the drillthrough.

65.2.6 Cross-Report Drillthrough (Advanced)

Drillthrough can happen across two different reports in Power BI Service. To do this:

1. **Enable the setting in File → Options → Report Settings → <Allow Cross-report drillthrough=.**
2. **Both reports must be in the same workspace.**
3. **The field names and data types must match exactly.**

65.2.7 Drillthrough vs Drill Down vs Tooltip

Feature	Purpose	Where it Happens
Drillthrough	Navigate to a new page filtered on value	New Report Page
Drill Down	Explore hierarchy levels in same visual	Within Same Visual (e.g., Year → Month)
Tooltip Page	Show small popup with more details	Hover Over Visual

65.3 Conditional Formatting in Power BI

65.3.1 What is Conditional Formatting?

Conditional formatting automates the visual appearance of data points based on rules or logic. It helps quickly identify:

- **Outliers**
- **Target achievements**
- **Negative values**
- **High risk zones**
- **Performance issues**

For example:

- **Green for good performance**
- **Yellow for warning**
- **Red for critical status**

65.3.2 Supported Elements for Conditional Formatting

You can apply conditional formatting to:

Type	Supported Visuals
Background color	Table, Matrix, Cards
Font color	Table, Matrix
Data bars	Table, Matrix
Icons	Table, Matrix
Web URL	Table, Matrix, Custom Tooltips
Image URL	Table, Matrix

65.3.3 Ways to Apply Conditional Formatting

There are three main methods:

1. **By Color Scale (Automatic): o Low → Red o Mid → Yellow o High → Green**
2. **By Rules (Manual Thresholds): o < 50 = Red o 50 to 75 = Yellow**
   - **75 = Green**
3. **By Field Value (using DAX expression):**
   - **A custom column (text value like <Green=) controls the color**

65.3.4 Step-by-Step: Apply Conditional Formatting

Let9s apply Background Color to a Table:

1. **Add a Table visual.**
2. **Go to Format → Cell Elements → Background color.**
3. **Turn it ON for a specific field.**
4. **Click Advanced Controls.**

5. **Choose formatting logic:**

o **Color scale o Rules o Field value**

6. **Apply → Done!**

65.3.5 Using Icons for Conditional Formatting

Icons add intuitive signals:

- **Green up arrow = growth**
- **Red down arrow = loss**
- **Warning triangle = below threshold**

You can customize icon style, size, alignment, and condition.

65.3.6 Using Data Bars

Data bars show the magnitude of values in bar form inside the cell.

Useful for comparing values in a list format.

65.3.7 Conditional Formatting via DAX Example

```
SalesColor =
SWITCH(
  TRUE(),
  [Sales] < 10000, "Red",
  [Sales] < 50000, "Yellow",
  "Green"
```

Then use this field for format by field value.

65.3.8 Conditional Formatting on Measures

Yes, you can apply formatting to DAX measures too.

Just ensure the measure returns a numerical or text value suitable for formatting.

65.4 Use Case Examples

Example 1: Highlight Low Inventory

- **Inventory < 20 = Red**
- **Inventory 20-50 = Yellow**
- **Inventory > 50 = Green**

Example 2: Sales Target Performance

- **Achieved ≥ 100% → Green checkmark icon**
- **Achieved 80-99% → Yellow warning icon**
- **Achieved < 80% → Red cross icon**

Example 3: Drillthrough on Product Performance

- **Matrix shows product summary**
- **Right-click → Drillthrough → <Product Detail Page=**
- **See monthly sales, top regions, returns, and marketing campaigns**

65.5.Best Practices

Drillthrough Best Practices	Conditional Formatting Best Practices
Always add a Back button	Use consistent color themes
Keep drillthrough pages simple	Don't overload visuals with too many icons
Use bookmarks with drillthrough for dynamic pages	Combine formatting with tooltips if using only color
Add titles indicating what the page shows	Always test for accessibility (color blindness)

65.6.Summary Table

Feature	Purpose	Output
Drillthrough	Navigate from summary to detail	New filtered report page
Conditional Formatting	Highlight data using visuals automatically	Colored backgrounds, icons, bars

Chapter 66: Power BI Service: Publishing, Sharing, Scheduling

Power BI is made up of two major components:

1. **Power BI Desktop – Used to build reports and dashboards.**
2. **Power BI Service (app.powerbi.com) – A cloud-based platform where you publish, share, collaborate, and automate your Power BI content.**

This chapter covers how to publish reports, share them with others, and schedule data refreshes in the Power BI Service.

66.1 What is Power BI Service?

Power BI Service is a cloud platform provided by Microsoft for:

- **Hosting reports and dashboards**
- **Sharing insights with teams or organization**
- **Scheduling data refreshes**
- **Managing datasets and workspaces**
- **Collaborating in real-time**

You can access it at:

澳https://app.powerbi.com

66.2.Key Components of Power BI Service

Component	Description
Dashboard	A canvas made from visuals pinned from multiple reports
Report	Interactive visualizations and pages built in Power BI Desktop
Workspace	A shared environment for teams to collaborate
Dataset	The data model and queries used in reports
App	A packaged version of reports/dashboards shared with users

66.3.Publishing Reports to Power BI Service

66.3.1 Step-by-Step Guide

1. **In Power BI Desktop, open the report you created.**
2. **Click File → Publish → To Power BI.**
3. **Sign in with your Microsoft account (Office 365 / Azure AD).**
4. **Select a workspace (e.g., My Workspace or a team workspace).**
5. **Wait for upload confirmation → Open in Power BI Service.**

Your report and dataset are now available on the cloud.

66.4 Creating and Using Workspaces

66.4.1 What is a Workspace?

A workspace is a collaborative environment where:

- You publish datasets, reports, dashboards
- Multiple team members can access or manage content
- You define roles (Admin, Member, Viewer)

66.4.2 Workspace Roles

Role	Permissions
Admin	Full control
Member	Edit, publish, refresh
Contributor	Add/edit content
Viewer	View-only access

Best practice: Avoid using "My Workspace" for sharing with others. Use a dedicated workspace instead.

66.5 Sharing Reports and Dashboards

You can share your reports with:

- Individuals
- Teams
- Entire organization
- Through apps

66.5.1 Methods of Sharing

Method Description
Direct Share Share report directly to users via email or link
Apps Bundle dashboards and reports into an app and distribute
Embed in Website Embed visuals into internal websites (requires permissions)
Method Description
Teams/Outlook Share directly in Microsoft Teams/Outlook

66.5.2 How to Share a Report

1. Open the report in Power BI Service.
2. Click the Share button (top-right).
3. Enter email addresses of recipients.
4. Choose permissions (view/edit/reshare).

5. **Add optional message → Click Send.**

Note: Users must have a Pro license or the content must be in a Premium workspace for free users to access it.

66.5.3 Best Practices for Sharing

- **Use apps to deploy to larger audiences**
- **Always check data security (RLS) before sharing**
- **Include page navigation and tooltips for better usability**
- **Add descriptive titles and tooltips in visuals**

66.6 Scheduling Data Refresh

66.6.1 What is Data Refresh?

When you publish a report, Power BI does not automatically refresh your local Excel or database connection.

You must schedule it in the Power BI Service.

66.6.2 Steps to Schedule Refresh

1. **Go to Power BI Service → Datasets tab in your workspace.**
2. **Click the ellipsis (...) next to your dataset → Settings.**
3. **Under Scheduled refresh, turn it ON.**
4. **Configure:**
    - **Time zone**
    - **Refresh frequency (daily, weekly, hourly)**
    - **Time slots**
5. **Add notification rules (optional).**
6. **Click Apply.**

Note: You need a gateway if the data source is on-premises.

66.6.3 Types of Data Sources and Refresh Needs

Data Source	Gateway Needed?	Refresh Type
Excel (OneDrive)	❌	Auto-refresh
SQL Server (local)	✅	Manual/scheduled
SharePoint Online	❌	Auto-refresh

Web Data	❌	**Schedule via service**
APIs	❌	**Manual/custom refresh**

66.6.4 Refresh Failure Notifications

You can enable email alerts if:

- **A refresh fails**
- **A dataset does not refresh on time**

Useful for data owners and admins.

66.6.5 Data Gateway (for On-Premises Sources)

If your data is stored locally (e.g., Excel on your PC, SQL Server on LAN), you must install and configure the Power BI Gateway on a server that runs 24/7.

Steps:

1. **Download gateway from Power BI Service**
2. **Install and link it to your account**
3. **Set credentials and permissions**
4. **Configure in the dataset's settings**

66.7 Security Considerations

- **Use Row-Level Security (RLS) to restrict data by user roles.**
- **Avoid publishing sensitive data to public workspaces.**
- **Use Azure Active Directory Groups for sharing in bulk.**
- **Always audit user access and report usage regularly.**

66.8.Key Differences Between Desktop and Service

Feature	Power BI Desktop	Power BI Service
Report Creation	✅	❌
Data Modeling	✅	❌
Sharing	❌	✅
Scheduling Refresh	❌	✅

Workspaces & Collaboration	❌	✅
Real-time Dashboards	❌	✅

66.9.Summary

Task	Where it Happens
Build Reports	**Power BI Desktop**
Publish & Share	**Power BI Service**
Schedule Refresh	**Power BI Service**
Manage Access & Security	**Power BI Service**

66.10 Real-Life Use Case

An analyst builds a report in Power BI Desktop showing:

- **Sales by product**
- **Customer satisfaction**
- **Regional growth**

They publish it to Sales Workspace → schedule a daily refresh → share with managers via Power BI App → users view it on mobile or browser and act on it instantly.

Chapter 67: Mobile App & Dashboards

67.1 Introduction

Power BI Mobile App is a powerful tool that enables users to access dashboards and reports from anywhere, using smartphones or tablets. It supports iOS, Android, and Windows devices. With the mobile app, users can stay connected to business data and monitor key metrics on the go.

67.2 Key Features of Power BI Mobile App

Feature	Description
Live Dashboards	Access dashboards with real-time data, just like on desktop
Interactive Reports	Tap visuals, drill down into data, apply filters
Touch Optimized	Designed for fingers – not mouse clicks
QR Code Scanning	Scan QR codes to view specific reports instantly
Alerts & Notifications	Set alerts on KPIs and receive push notifications
Offline Access	Download reports to view without internet (limited functionality)
Dark Mode	Enhanced readability for low-light environments
Biometric Security	Supports Face ID, Fingerprint Lock for secure access

67.3 How to Get the App

- iOS (iPhone/iPad): App Store → Search <Power BI= → Download
- Android: Google Play Store → Search <Power BI= → Download
- Windows: Microsoft Store → Install Power BI app (available for Surface, etc.)

67.4 Logging In and Navigation

1. Open the app and sign in using your Microsoft 365 or Power BI Pro account.
2. You will see:

o Home: Recently accessed reports/dashboards. o Favorites: Bookmarked dashboards. o Workspaces: Your workspace and shared team workspaces. o Apps: Published apps from your organization. o Alerts: All data alerts you've configured.

67.5 Using Dashboards in Mobile

- Dashboards appear as tiles (cards) on the screen.
- Each tile can be: o A chart o KPI card o Image o Web content
- Tap any tile → Opens the underlying report or detail view.
- Swipe to move between tiles.
- Tap to zoom or interact.

67.6.Mobile vs Desktop Layout

Aspect	Desktop	Mobile
Navigation	Tabs, mouse	Swipe, tap
Layout	Wide (landscape)	Narrow (portrait)
Actions	Hover, right-click	Tap, press
Interaction	High detail, complex	Summary, focus metrics
Performance	Depends on PC	Optimized for mobile memory/battery

Best Practice: Design mobile layouts separately in Power BI Desktop using the "Mobile Layout" tab.

67.7.Mobile Layout Design in Power BI Desktop

1. Open your report in Power BI Desktop.
2. Go to View > Mobile Layout.
3. Drag and drop visuals into a vertical canvas.
4. Remove clutter; only include essential KPIs or summary charts.
5. Save and publish. Mobile users will automatically see the optimized layout.

67.8 QR Code Integration

Power BI can generate QR codes for specific dashboards or tiles.

Steps:

1. Go to any dashboard on Power BI Service.
2. Click ... (More options) > Generate QR Code.
3. Print or share the QR Code.
4. Users can scan it using the Power BI Mobile App.

Use Cases:

- Attach QR codes to machines to monitor their metrics.

- **Place on meeting room doors to show booking status.**
- **Use in shops to monitor footfall or POS data.**

67.9 Alerts and Notifications

You can set alerts on KPI tiles (cards showing numbers).

Steps:

1. **Tap a tile > More options > Add alert rule.**
2. **Set conditions (e.g., <If Sales > 100000, alert me=).**
3. **Receive push notifications or emails.**

Example: If product stock drops below 100 units, alert the store manager instantly.

67.10 Working with Bookmarks and Buttons

- **Bookmarks save visual states (e.g., filters, focus modes).**
- **Buttons let users navigate, reset filters, or toggle visuals.**

In Mobile:

- **Use large, clearly labeled buttons.**
- **Keep bookmark functionality simple (2–3 states).**
- **Test bookmarks for mobile responsiveness.**

67.11 Data Refresh and Live Connectivity

- **The mobile app reflects the latest refreshed data from Power BI Service.**
- **Live reports (like connected to SQL/Azure) update in real-time.**
- **Offline support allows viewing downloaded versions (read-only).**

67.12.Mobile App Use Cases by Role

Role	Use Case
Salesperson	View monthly performance on the road
Store Manager	Scan QR codes on machines to check health metrics
CEO/Executive	Monitor company KPIs before meetings
Logistics Officer	Get alert when deliveries are delayed
Finance Analyst	Quickly compare budget vs. actuals from mobile

67.13.Tips for Building Mobile Dashboards

1. **Minimize visuals: Focus on KPIs and charts that answer questions.**
2. **Use card visuals: Best for mobile.**
3. **Avoid scrolling across: Stick to vertical layouts.**
4. **Use tooltips carefully: No hover on mobile.**
5. **Color contrast: High contrast for readability on screens.**

6. **Test frequently on your phone before publishing.**

67.14 Summary

Power BI Mobile App empowers decision-makers to stay informed anywhere, anytime. It ensures real-time insights, rich interactivity, and mobile-optimized layouts. With proper design and features like alerts, QR codes, and touch navigation, users can interact with data as naturally as reading a text message.

Chapter 68. Integration with Excel, SharePoint, Power Automate

68.1 Integration with Excel

Excel and Power BI are both core tools in Microsoft9s data ecosystem. Their deep integration supports both importing data into Power BI and analyzing Power BI datasets in Excel.

68.1.1 Loading Excel Data into Power BI

Methods:

a) From Local File:

- **Home tab → Get Data → Excel Workbook**
- **Select the file and sheets or tables.**
- **Load directly or transform using Power Query.**

b) From OneDrive for Business:

- **Benefits:**
  - **Auto-refresh every hour.**
  - **Seamless cloud sync.**
- **Add from Power BI Service → Get Data → Files → OneDrive - Business.**

c) From SharePoint Online:

- **Treat SharePoint as a cloud drive.**
- **File format:**
  https://site.sharepoint.com/sites/Team/Shared%20Documents/report.xlsx

Tip: Always convert Excel ranges into tables (Ctrl + T) before loading—Power BI treats tables as structured, refreshable data.

68.1.2 Limitations to Be Aware Of

- **Cannot load password-protected Excel files.**
- **Cannot connect to macros or external linked ranges.**
- **Large Excel files can slow performance.**
- **Avoid using merged cells or empty header rows.**

68.1.3 Analyze Power BI Dataset in Excel (Live Connection)

This feature allows Excel to act like a front-end for Power BI datasets.

Steps:

1. **In Power BI Service, open a published dataset.**
2. **Click <Analyze in Excel=.**

3. **Download .odc file.**
4. **Open it in Excel – build PivotTables and use Power Pivot.**

Best for:

- **Finance teams who prefer Excel**
- **Quick ad-hoc analysis using published Power BI data**
- **Consistent data governance (because data comes from centralized Power BI dataset)**

68.2 Integration with SharePoint

SharePoint is Microsoft9s document management and collaboration platform. Power BI can pull data from SharePoint lists/files and embed reports into SharePoint pages.

68.2.1 Connecting to SharePoint Lists

Example Use Case: A project management team logs issues in a SharePoint list. You can report on issues in Power BI.

Steps:

1. **Power BI Desktop → Get Data → SharePoint Online List**
2. **Enter site URL (not the full list URL).**
3. **Choose the list (e.g., "Project Issues").**
4. **Use Power Query to clean list columns (remove system columns like GUID, Attachments, etc.)**
5. **Load data.**

Power Tip: Watch for lookup columns – they create nested records. Use Expand to flatten them.

68.2.2 Connecting to Excel Files Stored in SharePoint

Instead of OneDrive, you may use SharePoint libraries.

Steps:

1. **Get the file link ending in .xlsx.**
2. **Go to Power BI → Get Data → Web.**
3. **Paste the file path and authenticate.**
4. **Select table(s) or range(s).**

Tip: Use Organizational Account authentication (not Anonymous or Windows).

68.2.3 Embed Power BI Report in SharePoint Page

Perfect for dashboards within company portals or team sites.

Steps:

1. **Publish the report to Power BI Service.**
2. **Copy report link (from web address bar or share pane).**
3. **Go to SharePoint Online.**
4. **Edit page → Add Power BI web part → Paste report URL.**
5. **Save and publish the page.**

Use Case:

- **HR Dashboards embedded in HR SharePoint portal**
- **Project dashboards visible in PMO SharePoint pages**

68.3 Integration with Power Automate

Power Automate connects Power BI to workflow automation.

68.3.1 Adding Power Automate Visual in Power BI

You can trigger flows from inside a report based on selection.

Steps:

1. **Power BI Desktop → Visualizations → Add Power Automate for Power BI visual.**
2. **Drag data fields into the visual.**
3. **Click Create New Flow.**

You9ll be taken to Power Automate designer.

Flow Example:

- **Trigger: When user selects a customer with overdue balance.**
- **Action: Send email to finance team + post on Microsoft Teams.**

68.3.2 Alerts + Automated Actions

You can trigger flows based on Power BI alerts.

Steps:

1. **In Power BI Service, pin a KPI card to a dashboard.**
2. **Set an alert (e.g., "when revenue drops below 1M").**
3. **In Power Automate:**

o Trigger: Power BI data alert o Action: Email, Teams message, SharePoint item creation, etc.

68.3.3 Real-world Use Cases

Scenario	Trigger	Power Automate Action
Customer debt overdue	Visual selected in Power BI	Send collection email

Inventory < threshold	KPI alert	Auto-email supplier
Approval needed	Button clicked in report	Create approval flow
New entry in SharePoint list	Automatically reflected in report	No manual refresh required (with scheduled refresh)

68.3.4 Common Automation Templates

- "Send email when a Power BI alert is triggered"
- "Start approval flow when record selected in Power BI"
- "Create Teams channel message when data condition met"

68.4.Best Practices for Integration

Tool	Best Practice
Excel	Use structured tables, avoid merged cells
SharePoint	Use file links ending in .xlsx or list API
Power Automate	Name flows clearly, add conditions to avoid false triggers
Power BI	Store reports and data in same workspace for refresh efficiency

68.5.Summary

Power BI9s integration with Excel, SharePoint, and Power Automate forms a powerful enterprise ecosystem:

- Excel: Ideal for analysts who want to explore trusted data sources or import legacy sheets.
- SharePoint: Provides centralized access and auto-refreshed data in reports.
- Power Automate: Turns insights into real-time, automated action.

With these integrations, organizations can build collaborative, intelligent, and automated data solutions that respond instantly to business needs.

Chapter 69: AI Visuals and Power BI APIs

69.1 AI Visuals in Power BI

AI visuals are special visuals that use machine learning and statistical techniques to provide deeper insights, automatically detect patterns, and help make predictions or explanations. You don9t need to be a data scientist to use them—Power BI makes it easy.

69.1.1 Key AI Visuals in Power BI

a) Q&A Visual (Natural Language Querying)

What it does: Lets users ask questions in plain English like <total sales last month= or <top 5 products by profit=.

Steps to use:

1. **Insert Q&A Visual from the Visualizations pane.**
2. **Type a natural language question.**
3. **Power BI shows the answer in chart/table form.**

Tip: Model your data well (proper names, relationships) to make Q&A smarter.

b) Decomposition Tree

What it does: Breaks down a metric (like Sales) step-by-step by dimensions (Region → Product → Salesperson).

Features:

- **Users can expand/collapse parts interactively.**
- **Can show high value, low value, or all values at each branch.**
- **Uses AI to suggest the most impactful breakdown.**

Steps to use:

1. **Insert Decomposition Tree.**
2. **Drag metric into Analyze.**
3. **Drag dimensions into Explain by.**
4. **Click branches to explore.**

Use case: Find out why profit dropped in a specific region and which product was most responsible.

c) Key Influencers Visual

What it does: Explains what factors influence a particular outcome (like customer churn or high sales).

Features:

- **Shows what fields have most influence on a selected field.**

- **Works like a mini machine learning model.**
- **Two modes: Top Influencers and Contrast (comparison).**

Steps:

1. **Insert Key Influencers visual.**
2. **Add a metric to Analyze (e.g., Churn = Yes/No).**
3. **Add multiple fields to Explain by (e.g., Age, Region, Plan Type).**
4. **Visual shows charts of influential factors.**

Use case: <What drives customer churn?=

d) Smart Narrative

What it does: Automatically generates text summaries (natural language) from your visuals.

Features:

- **Auto updates based on filters/slicers.**
- **Fully customizable (you can edit the generated text).**
- **Shows insights like totals, trends, comparisons.**

Steps:

1. **Insert Smart Narrative visual.**
2. **Click a visual → right-click → Add to Smart Narrative.**
3. **Edit the text if needed.**

Use case: Auto-generate summaries for managers.

e) Anomaly Detection (in Line Charts)

What it does: Detects outliers and unusual data points automatically.

Steps:

1. **Insert a Line Chart with time series data.**
2. **Enable <Anomaly detection= in the analytics pane.**
3. **Customize sensitivity.**

Use case: Detect sudden spike in traffic or drop in revenue.

69.1.2 Benefits of AI Visuals

Feature	Benefit
Natural Language Q&A	Self-service reporting
Key Influencers	Automated analysis of drivers
Decomposition Tree	Easy drilldown and explanation
Smart Narrative	Automatically generated summaries

Anomaly Detection	**Early detection of problems**

69.2 Power BI APIs

Power BI provides REST APIs and JavaScript APIs that allow developers and advanced users to automate, integrate, and embed Power BI into other apps and services.

69.2.1 Power BI REST API

REST APIs are web-based APIs that allow programmatic control of Power BI objects.

69.2.1.1 What Can You Do With REST APIs?

Task	Example
Get dataset or report info	List all datasets in a workspace
Refresh datasets	Trigger a scheduled refresh programmatically
Upload PBIX files	Publish reports from your code
Manage users	Add/remove users from a workspace
Export reports	Export Power BI report to PDF or PowerPoint

69.2.1.2 Example: Refresh Dataset using REST API

POST
https://api.powerbi.com/v1.0/myorg/groups/{groupId}/datasets/{datasetId}/refreshes
Authorization: Bearer {access_token}
This can be scheduled using Python, Power Automate, or Azure Functions.

69.2.1.3 Authentication

To use Power BI REST APIs, you need:

- **Azure AD app registration**
- **OAuth2 token with appropriate scopes (e.g., Dataset.ReadWrite.All)**
  Admin or contributor access to the workspace

69.2.2 Power BI JavaScript API (Power BI Embedded)

This allows embedding Power BI reports and dashboards into custom applications, portals, or web apps.

69.2.2.1 Use Cases for Embedding

- ISVs (independent software vendors) embedding reports into client apps
- Internal apps with custom filtering experience
- Secure embedding with row-level security

69.2.2.2 Example: Embed Report in Web App

```
<div id="reportContainer"></div>
<script> var report =
powerbi.embed(document.getElementById("reportContainer"), {
  type: 'report',  id: 'reportId',
embedUrl: 'https://app.powerbi.com/reportEmbed?reportId=reportId',
accessToken: 'your_access_token',
  settings: {
filterPaneEnabled: false,
navContentPaneEnabled: true
  }
 });
</script>
```

69.2.2.3 Secure Embedding Options

- User owns data: User logs in using their credentials (e.g., Power BI Pro license required)
- App owns data: Backend service authenticates with Azure AD and gets embed token for any user (best for external users)

69.2.3 Power BI + Azure AI

You can go beyond built-in AI visuals by integrating Azure Machine Learning and Cognitive Services:

- Call Azure ML models from Power Query or Dataflows
- Use Language Detection, Sentiment Analysis, and Image Tagging in Power BI

69.2.4 Automation & Scripting Examples

Task	Tool	Description
Schedule dataset refresh	PowerShell, Python	Automate refreshes every hour
Deploy reports	REST API + DevOps	CI/CD for Power BI
Embed dashboard in app	JS API	Custom user interface
Export report as PDF	REST API	Auto-generate and email reports

69.3 Best Practices for AI Visuals & APIs

Area	Best Practice
AI Visuals	Clean, well-labeled data improves model accuracy
APIs	Use Service Principals instead of user tokens in production
Q&A	Use synonyms and metadata to improve question results
Embedding	Use Row-Level Security for personalized experiences
Dataset Refresh API	Monitor refresh status with retry logic

69.4 Summary

Power BI9s AI Visuals help non-technical users discover insights using natural language and intelligent analysis. The Power BI APIs enable developers to integrate, embed, and automate Power BI beyond its UI.

Together, they transform Power BI from a reporting tool into a smart, interactive, and programmable analytics platform.

Chapter 70: Introduction to Tableau and Workspace Overview

70.1 What is Tableau?

Tableau is a leading Business Intelligence (BI) and data visualization tool that enables users to convert raw data into interactive, visually appealing dashboards and reports with ease. It allows users to explore data, identify trends, and make informed decisions—all without needing to write complex code.

Key Characteristics:

- **Visual drag-and-drop interface**
- **Wide range of chart types**
- **Real-time data analysis**
- **Interactive dashboards**
- **Can connect to various data sources**
- **Highly customizable visuals**

70.2 Why Use Tableau?

- **Intuitive Interface: Drag-and-drop simplicity allows both technical and non-technical users to create insights quickly.**
- **Interactive Dashboards: End-users can filter, drill down, and explore data dynamically.**
- **Speed & Performance: Tableau is optimized for in-memory processing, especially when using extracts.**
- **Data Source Compatibility: Supports connections with Excel, SQL, Google Sheets, Cloud services, and Big Data tools.**
- **Advanced Analytics: Integrates with Python, R, and supports forecasting and clustering.**

70.3.Tableau Product Family

Product	Description
Tableau Desktop	Core application to build and publish dashboards
Tableau Public	Free version for publicly available dashboards
Tableau Online	Cloud-based version to share dashboards over the internet
Tableau Server	On-premises sharing and collaboration tool
Tableau Prep	Tool for data cleaning and transformation

Tableau Reader	Free desktop tool to view Tableau Packaged Workbooks (.twbx) offline
Tableau Mobile	View dashboards on smartphones and tablets

70.4.Tableau File Types

Extension	Description
.twb	Tableau Workbook – contains instructions and structure of the report
.twbx	Packaged Workbook – includes data, images, and visualizations in one file
.tde / .hyper	Tableau Data Extracts – optimized files for faster performance
.tds	Tableau Data Source file – stores metadata like calculated fields
.tdsx	Packaged Data Source – data source + extract combined

70.5.Tableau Workspace Overview

When you launch Tableau Desktop and open a new worksheet, here is what you9ll see:

70.5.1 Start Page

- Left Pane: Connect to data (Excel, CSV, Server, etc.)
- Right Pane: Recent workbooks and templates

70.5.2 Data Source Page

- View and manage tables
- Join and union data sources
- Preview rows and rename columns
- Change data types

70.5.3 Worksheet Interface

Element	Description
Data Pane	Left section showing all dimensions, measures, parameters
Shelves	Top area to drag fields for Rows, Columns, Filters, Pages

Marks Card	**Control visual properties (color, size, shape, tooltip)**
Canvas	**Middle space showing the chart/visual**
Show Me	**Top-right panel suggesting chart types**
Tabs	**Bottom of the screen: navigate between Sheets, Dashboards, Stories**

70.5.4 Marks Card Options

- **Color: Assign color based on category or measure**
- **Size: Adjust mark size dynamically**
- **Label: Add data labels**
- **Tooltip: Customize what9s shown when you hover over data**
- **Detail: Add additional context without affecting visual layout**
- **Shape: Use different mark shapes**

70.6 Tableau Workflow Overview

1. **Connect to Data o Select source (Excel, SQL, Google Sheets, etc.)**
2. **Prepare Data o Rename fields, adjust data types, create joins or unions**
3. **Create Worksheets o Drag fields into rows/columns, apply filters, create visuals**
4. **Build Dashboards**

o Combine multiple sheets, add filters, images, interactivity

5. **Add Actions o Filter, highlight, or URL actions for user engagement**
6. **Publish or Export o Publish to Tableau Server, Online, or export to PDF/Image**

70.7 Best Practices for Beginners

- **Use Extracts for better performance with large data**
- **Choose the right chart type for the data story**
- **Avoid overuse of color and cluttered visuals**
- **Use parameters for user-driven filters**
- **Always preview your dashboard on Tableau Public or Server**

70.8.Similarities and Differences with Power BI

Similarities

Feature	Tableau	Power BI
Purpose	Business Intelligence & Visualization	Business Intelligence & Visualization
No-code Drag & Drop UI	Yes	Yes
Interactive Dashboards	Yes	Yes
Custom Calculations	Yes (Calculated Fields)	Yes (DAX, Power Query M)
Scheduled Refresh	Yes (with Tableau Server/Online)	Yes (with Power BI Service)
Mobile App	Yes	Yes
Connect to Data	Yes – wide variety	Yes – especially Excel, SQL

Differences

Feature	Tableau	Power BI
Ease of Use	Slightly more complex for beginners	More user-friendly for Excel users
Data Modeling	Less flexible	More powerful with relationships and DAX
Price	Higher licensing costs	More affordable (especially with Microsoft 365)
Customization	More flexible visual customization	Slightly more rigid visuals
Community/Public Access	Tableau Public for free dashboards	Limited public sharing in free version

Scripting Support	**Python, R integration**	**R and Python supported, but more limited**
Performance	**Faster with large data (especially with extracts)**	**Performance depends on model and service**
File Types	**.twb, .twbx**	**.pbix**
Storage	**Cloud or on-prem (Server/Online)**	**Primarily cloud-based**

70.9.Summary

In this chapter, you have learned:

- **What Tableau is and what it's used for**
- **The key features, product types, and file types in Tableau**
- **Detailed layout and workspace overview**
- **How to navigate Tableau's interface to create visuals**
- **Workflow from data connection to publishing**

Chapter 71: Connecting to Data Sources (Excel, CSV, Google Sheets)

71.1 Why Data Connection is Important in Tableau

Tableau is a visual analytics platform, but before visualizing anything, it needs to know where your data is and how it's structured.

Connecting to the right data source allows Tableau to:

- Understand your fields and data types
- Let you clean, shape, and model your data
- Refresh and stay in sync with live sources
- Build visualizations based on real-time or extracted data

71.2 Supported Data Sources Overview

Tableau can connect to:

- Files: Excel, CSV, JSON, PDF, Spatial files, Statistical files (e.g., SAS, SPSS, R)
- Databases: SQL Server, MySQL, PostgreSQL, Oracle, Snowflake, Redshift, etc.
- Clouds: Google Sheets, Google BigQuery, Amazon Redshift, Salesforce, etc.
- Web Connectors: Connect to web data using APIs or Tableau9s Web Data Connector (WDC)

In this chapter, we9ll focus on file-based and cloud spreadsheet sources:

- Microsoft Excel
- CSV (Text) Files
- Google Sheets

71.3 Connecting to Excel in Tableau

Steps to Connect:

1. Launch Tableau Desktop
2. On the Start Screen, under <Connect=, click Microsoft Excel
3. Browse and select your Excel file (.xlsx, .xls)
4. Tableau displays all available worksheets (tabs) from the file
5. Drag one or more worksheets into the data canvas (middle panel)

Key Options in Excel Data:

- Use Data Interpreter (cleans messy headers, hidden rows, merged cells)
- Join multiple sheets (if they share a common key)
- Union sheets (if the structure is the same across sheets)

Best Practices for Excel Files:

- Ensure your first row contains column headers
- Avoid merged cells

- **Do not insert blank rows in data**
- **Clean up calculated formulas (use static values if possible)**
- **Keep each table in a separate sheet if you plan to join or union them**

71.4 Connecting to CSV Files

CSV stands for Comma-Separated Values. These are plain-text files where each line is a record, and values are separated by commas.

Steps to Connect:

1. **Open Tableau and click Text File under <Connect=**
2. **Select your .csv file**
3. **Tableau opens the file in the Data Source tab**
4. **Review and clean field names, data types, and sample data**

Things to Check:

- **Delimiter issues: If it's not comma-separated (e.g., pipe |, tab \t), you may need to adjust or convert it**
- **Missing headers: Rename fields in Tableau if the first row is not headers**
- **Encoding: Use UTF-8 encoding to prevent issues with special characters (like ₹, €, or Chinese characters)**

Best Practices for CSV Files:

- **Always use one table per file**
- **Avoid putting notes or extra information at the top**
- **Don9t include formatting – CSV is plain text only**
- **Prefer using a .csv extension, not .txt unless it's a tab-separated format**

71.5 Connecting to Google Sheets

Google Sheets are cloud-based spreadsheets by Google. Tableau connects directly to your Google account and reads the data from Sheets.

Steps to Connect:

1. **On Tableau9s Start page, go to Connect > To a Server > Google Sheets**
2. **A browser opens – sign in with your Google account**
3. **Grant Tableau access to your sheets**
4. **Once authorized, Tableau shows a list of your spreadsheets**
5. **Select a spreadsheet, and choose the sheets you want to load**

Notes on Google Sheets:

- **Tableau requires Internet access to connect to Google Sheets**
- **It supports both Live and Extract connections**
- **If the Google Sheet is updated, Tableau will reflect those changes when refreshed (in live mode)**

Best Practices for Google Sheets:

- **Store your data in the first worksheet**
- **Keep headers in the first row**
- **Avoid formulas that use dynamic web content like GOOGLEFINANCE() or IMPORTHTML(), as Tableau may not interpret them correctly**

71.6.Understanding the Data Source Tab

Once you connect to your data (Excel/CSV/Google Sheets), Tableau brings you to the Data Source Tab.

Here9s what you can do:

Option	Description
Rename Fields	Double-click column headers to rename
Change Data Types	Click the data type icon (e.g., ABC, #) to change it
Filter Rows	Use <Add Filter= to limit rows
Join Sheets	Combine two or more sheets by matching a common column
Union	Stack sheets that have the same columns
Preview Data	See sample records before jumping to visualization
Use Data Interpreter	Clean up bad structure in Excel/CSV

71.7.Use of Data Interpreter (Excel/CSV)

What It Does:

- **Cleans up multi-row headers**
- **Skips extra notes above data**
- **Removes unnecessary blank rows**
- **Handles merged cells**

How to Use:

- **When connecting to Excel or CSV, you9ll see a checkbox <Use Data Interpreter=**
- **Click it to clean the data**

- **Optionally, download a cleaning report that explains what was removed or fixed**

71.8 Live vs Extract Connection

Mode	Description	Pros	Cons
Live	Tableau pulls data directly from source	Real-time data updates	Slower, especially with large data
Extract	Tableau creates a .hyper file (snapshot)	Faster, can work offline	Needs manual refresh

Switching Between Them:

- **Go to the Data pane**
- **Right-click your data source → Select Use Extract or Use Live**

71.9 Previewing and Cleaning Data

After loading your data, you can:

- **Create calculated fields in the data source tab**
- **Use Filters to exclude rows**
- **Hide unnecessary fields**
- **Change field names and types**

This prepares the data for effective use in Sheets (visualization interface).

71.10 Summary

In this chapter, you9ve learned:

- **How to connect Tableau to Excel, CSV, and Google Sheets**
- **Key differences between these file types**
- **How to use Data Interpreter to clean structured data**
- **The difference between Live and Extract modes**
- **Best practices to keep your data clean and Tableau-ready**

Chapter 72: Dimensions vs Measures

72.1 Real-Life Analogy to Understand Easily

Think of your data as an Excel table where you9re tracking sales across different cities:

City	Category	Sales	Profit
Mumbai	Furniture	500	50
Delhi	Office	700	80
Chennai	Tech	900	100

- "City" and "Category" describe what or where.
- "Sales" and "Profit" are how much you sold or earned.

In Tableau:

- "City", "Category" → Dimensions
- "Sales", "Profit" → Measures

72.2 Deep Dive into Dimensions

Dimensions = Descriptive Fields

They categorize your data and are often text, dates, or IDs.

Purposes:

- Slice or divide your data (e.g., show sales by City)
- Create groupings, filters, labels
- Generate headers in charts

Types of Dimension Fields:

- Text: Customer Name, City, Region
- Date/Time: Order Date, Ship Date
- IDs (non-numeric logic): Order ID, Product ID

Example in Tableau:

You want to see total sales by region:

- Drag <Region= (dimension) to Columns
- Drag <Sales= (measure) to Rows

Tableau automatically generates a bar chart where each bar represents a region.

72.3 Deep Dive into Measures

Measures = Quantitative Fields

They are numerical, can be summed, averaged, counted, etc.
Purposes:

- **Perform calculations (e.g., SUM, AVG, MAX, COUNT)**
- **Used as values in graphs, KPIs**
- **Generate axes in charts (bar lengths, line heights)**

Examples of Measures:

- **Sales, Profit, Quantity, Revenue, Discount**
- **Calculated metrics: Profit Margin, Average Sales per Day**

Example in Tableau:
You want to calculate average profit by category:

- **<Profit= → Measure**
- **<Category= → Dimension**

Use aggregation: AVG([Profit])

72.4 Tableau Behavior: What Happens Visually?

Blue vs Green Pills

Pill Color	Type	Meaning
Blue	Discrete	Creates categories
Green	Continuous	Creates numeric axes

- **Blue (Dimensions): Used to split the view into separate parts**
- **Green (Measures): Used to plot values or KPIs**

72.5 Dimensions vs Measures in a Chart Example

Scenario: Analyzing Monthly Sales in Delhi

Order Date	City	Sales
Jan-2024	Delhi	3000
Feb-2024	Delhi	3500
Mar-2024	Delhi	4000

- **"Order Date" → Dimension**
- **"Sales" → Measure**

Create Line Chart:

- **Drag Order Date to Columns**
- **Drag Sales to Rows**

You9ll get a line showing sales growth over time.

72.6 Converting Fields: Why & How

Sometimes, fields are misclassified. Example:

- <Product ID= = 101, 102, 103 (Numbers)
- Tableau may consider it a Measure, but it9s actually categorical

Convert to Dimension:

Right-click → Convert to Dimension

Similarly, convert a numeric field to Measure when needed.

72.7 Advanced Use: Calculations with Dimensions + Measures

You can create Calculated Fields:

Example 1:

Sales per Customer

SUM([Sales]) / COUNTD([Customer Name])

Here:

- [Sales] = Measure
- [Customer Name] = Dimension used inside COUNTD

Example 2:

Profit Ratio

SUM([Profit]) / SUM([Sales])

These calculations create new Measures for analysis.

72.8 Impact on Filter, Sorting, and Grouping

Filters:

- Filter by Dimension → Filter entire categories (e.g., only <Furniture= category)
- Filter by Measure → Use ranges (e.g., only show sales > 10,000)

Sorting:

- Sort by Dimension: Alphabetical order
- Sort by Measure: Numeric values (high to low)

Grouping:

Group Dimensions (e.g., group <Mumbai= and <Delhi= as <North India=) You can9t group Measures; but you can bin them (e.g., Profit bins)

72.9 Practice Scenario

Let9s say you have this dataset:

Product	Category	Sales	Profit

A	**Tech**	**1000**	**200**
B	**Office**	**1500**	**300**
C	**Furniture**	**2000**	**400**

Create a bar chart showing profit by product:

1. **Drag <Product= → Columns (Dimension)**
2. **Drag <Profit= → Rows (Measure)**

You9ll get 3 bars, one for each product9s profit.

72.10 Summary Table: Full Comparison

Feature	Dimensions	Measures
Type	Categorical (Qualitative)	Quantitative (Numeric)
Aggregation	Not aggregated	Can be aggregated
Chart Role	Defines structure/grouping	Defines data values
Tableau Color	Blue (Discrete)	Green (Continuous)
Examples	Customer, Region, Date	Sales, Profit, Quantity
Chart Behavior	Creates labels or segments	Creates bars, lines, axes
Convertibility	Can convert to/from Measures	Can convert to/from Dimensions

Chapter 73: Basic Charts 4 Bar, Line, Pie, Text in Tableau

This chapter will help you master the foundational chart types in Tableau — the ones used most often in dashboards, business reports, and storytelling.

73.1 What Are Basic Charts?

Basic charts help you visually communicate data for quick understanding. Tableau makes these charts very easy to create with drag-and-drop actions.

We9ll cover:

- **Bar Charts**
- **Line Charts**
- **Pie Charts**
- **Text Tables (Cross Tabs)**

73.2 Bar Chart (Most Common Visual)

Purpose:

Used to compare values across different categories — like sales by product, profit by region, etc.

Example:

Product	Sales
A	1000
B	1500
C	1200

Bar Chart will show three vertical bars: one for each product, with heights proportional to sales.

How to Create in Tableau:

1. **Drag a Dimension (e.g., Product) → Columns**
2. **Drag a Measure (e.g., Sales) → Rows 3. Tableau will automatically show a bar chart.**

4. Optional:

- **Drag Category to color (for grouped bars) o Right-click on axis → sort descending**

Tips:

- **Use stacked bars for parts of a whole.**
- **Use horizontal bars if category names are long.**

73.3 Line Chart

Purpose:

Used to show trends over time, like monthly revenue, stock prices, website traffic, etc.

Example:

Month	Sales
Jan	2000
Feb	3000
Mar	2500

Line Chart will connect these points with a line to show how sales change over time.

How to Create in Tableau:

1. **Drag Date/Time Dimension (e.g., Order Date) → Columns**
2. **Drag a Measure (e.g., Sales) → Rows**
3. **Click on "Show Me" → select Line Chart**
4. **Tableau draws a line across time intervals.**

Tips:

- **Break lines by Category/Region for multiple lines.**
- **Format the date (Month, Year, Quarter) based on granularity.**

73.4 Pie Chart

Purpose:

Used to show parts of a whole. For example, market share of different brands or sales distribution by product category.

Note: Use pie charts only for small datasets (3–6 slices max). More slices = confusing.

Example:

Category	Sales
Tech	4000
Office	3000
Furniture	3000

Pie slices will be proportional to sales. Tech gets 40%, others get 30%.

How to Create in Tableau:

1. **Drag a Dimension (e.g., Category) → Color**
2. **Drag a Measure (e.g., Sales) → Angle**

3. **Go to Marks card → select <Pie=**
4. **Drag Measure to Label for showing % or values**

Tips:

- **Click "Label" → "Show Percent of Total"**
- **Use tooltips for detailed values**
- **Don't use pie charts for too many categories**

73.5 Text Table (Cross Tab)

Purpose:

Used to show raw data in tabular format, similar to Excel.

Example:

Category	Sales	Profit
Office	4000	200
Furniture	3500	150

How to Create in Tableau:

1. **Drag Dimensions (e.g., Category) to Rows**
2. **Drag Measures (e.g., Sales, Profit) to Text**
3. **Change Mark type → "Text"**
4. **This creates a grid of numbers**

Tips:

- **Use color coding (highlight higher/lower values)**
- **Add Grand Totals from the "Analytics" pane**
- **Keep it compact — best for summary tables**

73.6.Side-by-Side Comparison

Chart Type	Best For	Data Type Needed	Key Visual Feature
Bar Chart	Comparison	Category + Numeric	Bars of different lengths
Line Chart	Trend Over Time	Date + Numeric	Connected lines over time
Pie Chart	Proportions	Categorical	Circular wedges
Text Table	Raw Data View	All	Text/Number grid

73.7.Formatting Tips for All Basic Charts

- **Labels: Always show value labels or percentages if needed**
- **Colors: Use consistent color schemes; avoid too many colors**
- **Tooltips: Customize tooltips for more info on hover**
- **Sorting: Sort bars from highest to lowest to improve readability**
- **Titles & Legends: Use clear chart titles and legends**

73.8 Example Use Cases in Dashboards

Use Case	Recommended Chart
Monthly Revenue Trend	Line Chart
Sales by Product	Bar Chart
Profit Contribution by Region	Pie Chart
Detailed Customer Report	Text Table

Chapter 74: Sorting, Filtering, and Grouping in Tableau

This chapter focuses on essential data manipulation features in Tableau that help users organize, explore, and make sense of data.

We9ll learn how to:

- **Sort data (manually or automatically)**
- **Filter data (with interactive or static filters)**
- **Group values (combine or cluster similar items)**

These features improve dashboard readability and performance.

74.1 Sorting in Tableau

What is Sorting?

Sorting is the process of arranging data in a particular order — ascending or descending — based on field values.

Example:

Product	Sales
A	300
B	100
C	500

If sorted descending by sales → C, A, B.

Types of Sorting in Tableau:

A. Manual Sorting

- **Drag categories into your desired order.**
- **Useful when you need a custom-defined sequence (e.g., days of the week: Mon, Tue, ...).**

B. Automatic Sorting

- **Based on field values.**
- **Steps:**
  1. **Click on the axis label or header.**
  2. **Choose sort ascending or descending.**

C. Sort Dialog Box

- **Right-click the field → Sort...**
- **Sort by:**
  - **Data source order o Alphabetical**

- Field (e.g., Sales) o Choose ascending or descending

74.2 Filtering in Tableau

What is Filtering?

Filtering means restricting data in a view — showing only the data that meets certain conditions.

Why Filter?

- Focus on specific regions, time frames, products, etc.
- Improve performance by reducing volume.
- Create interactive dashboards using filter controls.

Types of Filters in Tableau:

A. Dimension Filters

- Filters on categorical fields (e.g., Country, Category).
- Choose members to include/exclude.
- Example: Show only Sales for <India= and <USA=.

B. Measure Filters

- Filters on numeric fields (e.g., Sales > 5000).
- Apply ranges, minimum, maximum, etc.

C. Date Filters

- Filter data based on time period.
- Options include: o Relative Date (e.g., last 7 days) o Range of Dates o Specific Years, Quarters, or Months

D. Top N Filters

- Show only Top N or Bottom N values.
- Steps:
  1. Right-click a dimension → Filter
  2. Go to Top Tab
  3. Choose <By field= → e.g., Top 5 by Sales

E. Context Filters

- Act as a filter on top of other filters.
- Useful when filters are dependent.
- Example:
  - Context Filter: Region = "West"
  - Secondary Filter: Top 3 Products in West only

How to Apply a Filter:

1. **Drag a field to the Filters Shelf**
2. **Choose values to include or exclude**
3. **Optionally, drag the field to Filter card on dashboard for interactivity**

Filter Options in Dashboards:

- **Show filters as:**
  - **Dropdown**
  - **Single/Multiple Value List**
  - **Slider (for dates/measures)**

You can apply filters to:
- **One sheet**
- **Selected sheets**
- **All using this data source**

74.3 Grouping in Tableau

What is Grouping?

Grouping is the process of combining multiple dimension values into a single group or category.

Example:

State	Group
Delhi	North India
Punjab	North India
Kerala	South India
Tamil Nadu	South India

Now you can analyze by region instead of individual states.

Ways to Group in Tableau:

A. Manual Grouping

1. **Ctrl + Select values (e.g., select "Delhi" and "Punjab")**
2. **Right-click → Group**
3. **Tableau creates a new group field**

B. Edit Group

- **Right-click on the group field → Edit Group**
- **Rename, add/remove members, or create <Other= category**

C. Automatic Grouping with Bins (Numerical Grouping)

- **For numerical fields like Age or Sales**
- **Create equal-sized intervals (e.g., age groups: 0–20, 21–40...)**

Steps:

- Right-click field → Create → Bins

- Set bin size When to Use Groups:
  - Simplify visualizations (combine small categories)
  - Clean messy data (e.g., grouping <APAC= and <Asia-Pacific=)
  - Roll up details for better performance

74.4 Combined Example

Let9s say you're analyzing Sales by Category across different countries:

- Sort the countries by highest sales (descending)
- Filter to show only 3 categories: Furniture, Office, Technology
- Group smaller countries into "Others"

This provides a clean, focused view for business users.

74.5 Tips and Best Practices

Feature	Best Practice
Sorting	Sort by highest value for readability
Filtering	Use relative dates for dashboards (e.g., Last 30 days)
Grouping	Use to reduce clutter, improve performance
Dashboard Filters	Apply to multiple sheets for consistency
Top N	Use to highlight best/worst performers

Key Takeaways:

- Sorting arranges your data for easier interpretation.
- Filtering removes unwanted noise and improves focus.
- Grouping helps simplify your data and improve performance.
- These tools work best together — use them to create clean, efficient dashboards.

Chapter 75: Publishing Dashboards in Tableau

75.1 What Does Publishing Mean in Tableau?

Publishing a dashboard means sharing your visualizations with others by uploading them to the Tableau Server, Tableau Online, or Tableau Public, so your audience can view and interact with them without needing Tableau Desktop.

Publishing enables:

- **Collaboration across teams**
- **Centralized data access**
- **Scheduled data refresh**
- **Interactive usage via browser or mobile**

75.2.Platforms for Publishing

Platform	Description
Tableau Public	Free platform for public sharing (not private or secure)
Tableau Server	Enterprise solution installed on-premises
Tableau Online	Cloud-based, secure, managed by Tableau

75.3.Preparing Dashboard for Publishing

Before you publish:

1. Clean and Optimize Dashboard

- **Remove unused fields and sheets**
- **Minimize dashboard size and complexity**
- **Set default filters and views**

2. Add Titles and Descriptions

- **Make sure every chart has a clear title**
- **Use Tooltips for better context**

3. Test Interactions

- **Test filters, actions, and tooltips**
- **Check mobile responsiveness (if needed)**

75.4 Steps to Publish (on Each Platform)

A. To Tableau Public (Free)

All data becomes public!

Steps:

1. **Create your dashboard in Tableau Desktop**
2. **Click File → Save to Tableau Public**
3. **Sign in to your Tableau Public account**

4. **Give your workbook a name**
5. **Your dashboard is published at:**

https://public.tableau.com/views/<your_dashboard>

B. To Tableau Server or Tableau Online Requires login and permission to a project folder.

Steps:

1. **Click Server → Publish Workbook**
2. **Select either: o Tableau Server**
   - **o Tableau Online**

3. Log in with your credentials 4. Choose the destination Project Folder

5. Set the following:

- **Workbook name**
- **Sheets to include**
- **Data source publishing method:**
  - **Embed in workbook**
  - **Publish separately (recommended for large data)**

6. **Configure Permissions (viewer, editor, etc.)**
7. **Click Publish**

75.5.Data Source Publishing Options

When publishing, you must choose how your data source is handled:

Option	Use When
Embedded in Workbook	Small, static datasets
Published Data Source	Large or shared datasets
Live Connection	Real-time updates required
Extract	For performance and scheduling

75.6.Permissions and Security

After publishing, control who can:

- **View dashboards**
- **Download data/workbooks**
- **Edit or comment**

Tableau supports role-based access like Viewer, Explorer, Creator.

Permissions can be set at:

- **Workbook level**
- **Project level**
- **Data source level**

75.7 Scheduling Refreshes (Only in Server/Online)

You can schedule automatic data updates for:

- **Extracted data sources**
- **Workbooks**

Steps:

1. **Go to Tableau Server/Online**
2. **Select workbook → Schedules**
3. **Choose refresh frequency:**
   - **Daily, Hourly, Weekly**

4. Ensure credentials are embedded for data access

75.8.Embedding and Sharing Dashboards

Once published, dashboards can be:

Option Purpose
URL Share Direct link via browser
Embed in Webpage Use iframe to integrate into your website
Email Snapshot Send a static image or PDF
Subscribe Users Automated emails on schedule

75.9.Best Practices for Publishing

Tip	Description
Use Extracts	Improves performance and enables scheduling
Optimize Filters	Minimize filters for faster loading
Mobile Friendly	Test views on smaller screens
Describe KPIs	Add instructions or legends for clarity
Tag Workbooks	Helps in searching and organizing on the Server

Summary

- **Publishing enables your dashboards to reach end users via browser/mobile.**
- **You can publish to Tableau Public, Server, or Online.**
- **Always prepare dashboards (optimize, test) before publishing.**
- **Permissions and scheduling play a vital role in managing access and updates.**
- **Post-publishing, use sharing options (URL, email, embed) to distribute dashboards efficiently.**

Chapter 76: Data Interpreter, Pivoting, Joins & Unions in Tableau

76.1 Data Interpreter

What is Data Interpreter?

Data Interpreter is a cleaning tool inside Tableau that helps you prepare Excel or Google Sheets data for analysis by removing unwanted formatting such as:

- Merged cells
- Blank rows/columns
- Headers or footers
- Notes embedded in the file
- Multiple header rows

Think of it like a smart helper that figures out where your real table starts and cleans everything else.

Why is it Needed?

Excel files created manually often contain report-like layouts with titles, descriptions, or summaries — not just clean tabular data. Tableau has trouble interpreting these. Data Interpreter:

- Automatically detects the true header row
- Removes irrelevant formatting
- Cleans the data for visualization

Supported Formats

File Type	Data Interpreter Available?
Excel (.xlsx)	✅ Yes
Google Sheets	✅ Yes
CSV	❌ No
SQL, JSON, etc.	❌ No

How to Use It?

1. Open Tableau → Connect to Excel file or Google Sheets
2. Click <Use Data Interpreter= in the Data Source pane

3. **Tableau will:**
   o Clean the file o Show the cleaned data o Allow you to <Review results=
4. **You can click Review Results to open the Excel file and view what Tableau changed Example**

Original Excel:

Company Sales Report Q1

Region | Jan | Feb | Mar
North | 200 | 300 | 400
South | 250 | 320 | 450

After using Data Interpreter:

Region	Jan	Feb	Mar
North	200	300	400
South	250	320	450

76.2 Pivoting in Tableau

What is Pivoting?

Pivoting is used to convert multiple columns into rows, allowing Tableau to treat similar data across time or category as a single variable.

Used to normalize wide tables into long (tidy) format.

When to Use It?

Use pivot when your data has columns that represent the same type of data — for example, months as columns:

Region Jan Feb Mar

This is not ideal for Tableau, because you can9t apply a date filter or trendline across columns.

After pivoting:

Region	Month	Sales
North	Jan	200

North	Feb	300

How to Pivot

1. **Go to Data Source tab**
2. **Select all the columns you want to pivot (e.g., Jan, Feb, Mar)**
3. **Right-click → Click <Pivot=**
4. **Tableau creates:**
    - **Pivot Field Names → rename it to "Month"**
    - **Pivot Field Values → rename it to "Sales"**

Benefits:

- **Easier time-based analysis (line charts, trend analysis)**
- **Can apply filters like Month = "Feb"**
- **Better formatting for Tableau's built-in date features**

76.3 Joins in Tableau

What is a Join?

Joining in Tableau means combining rows from two tables based on common fields (keys). Similar to SQL joins.

Types of Joins:

Type	Description
Inner Join	Keeps only matching rows from both tables
Left Join	All records from the left table + matched from right
Right Join	All records from right + matched from left
Full Outer	All records from both tables; matches where possible

How to Do It:

1. **Drag the second table from the sidebar onto the canvas beside the first**
2. **Choose the join type from dropdown (inner, left, etc.)**
3. **Choose the fields to match (e.g., Product ID in both tables)**

Example:

Customer Table:

Cust_ID	Name
101	Alice

102	Bob

Order Table:

Order_ID	Cust_ID	Amount
O001	101	$200
O002	103	$250

- **Inner Join on Cust_ID → only record for 101 (Alice)**
- **Left Join → includes both Alice and Bob**
- **Right Join → includes Alice and customer 103**

Join Precautions:

- **Ensure the fields have the same format (both are strings or both are integers)**
- **Mismatched or missing keys may result in nulls**
- **Can lead to data duplication if one side has repeated keys**

76.4 Unions in Tableau

What is a Union?

Union means stacking multiple tables with the same column structure vertically.

It's like gluing more rows at the bottom of a dataset.

When to Use:

- **You have monthly or regional files with the same format o Jan_Sales.xlsx o Feb_Sales.xlsx**
- **You have multiple sheets in Excel, each with same columns**

How to Create Union

A. Manual Union (Drag & Drop):

1. **Drag the first sheet to the canvas**
2. **Drag the second sheet below the first**
3. **Tableau automatically forms a union B. Wildcard Union (Fastest for many files):**

1. **Click "Add" → Union**
2. **Use pattern:**
3. **Sales_*.xlsx**
4. **Tableau combines all files that match this pattern**

Result:

If Jan and Feb files have:

Product	Sales
A	100
B	200

Unioned Output:

Product	Sales	Table Name
A	100	Jan_Sales
B	200	Feb_Sales

- **Tableau adds Table Name to track source file/sheet**

Summary: When to Use What

Feature	Purpose	Use Case
Data Interpreter	Clean Excel files with messy headers	Reports with titles/footnotes
Pivot	Convert columns to rows	Monthly sales data in columns
Join	Combine fields from different tables	Customer & Orders
Union	Stack similar tables vertically	Monthly or regional sales files

Best Practices

- **Always review output of pivot, join, or union in Tableau9s data view**
- **Rename columns after pivoting or union to make them meaningful**
- **When using Joins, ensure keys match in data type and content**
- **When using Union, check the Table Name column to trace source files**

Chapter 77: Groups, Sets, Hierarchies, and Calculated Fields in Tableau

77.1 Groups

What Are Groups?

A Group in Tableau is a user-defined category that combines multiple dimension values into one label. Think of it like creating your own classification.

- **It does not change the original data.**
- **It creates a new dimension field with grouped values.**

Use Cases

- **Clean and consolidate inconsistent data.**
- **Simplify dimensions in charts (e.g., 100 categories into 5 groups).**
- **Manually create categories for business logic (e.g., <East Zone= = Delhi, Kolkata).**

Creating Groups

Method 1: From the Data Pane

1. **Right-click a dimension (e.g., Customer Segment)**
2. **Choose Create > Group**
3. **In the window:**
   - **Select values (e.g., <Corporate=, <Small Business=)**
   - **Click Group**
   - **Rename to <B2B Customers=**

Method 2: From the View (Visual)

- **Select marks (e.g., bars or pie slices)**
- **Right-click >Group**

Resulting Field

- **Tableau creates a new field called GroupName (Group)**
- **This field behaves like any other dimension and can be used in rows, columns, filters, or colors.**

Notes

- **Groups are static – if a new category comes into data, it won9t be automatically grouped unless manually updated.**
- **You can ungroup values later or rename the grouped label.**

77.2 Sets

What Are Sets?

A Set is a subset of data that is either:

- **Fixed (static): Manually selected members.**

- **Dynamic: Defined by rules or conditions.**

Purpose

- **Highlight specific records (e.g., Top 10 Customers)**
- **Compare "In Set" vs "Out of Set"**
- **Use in visual interactions (via Set Actions in dashboards)**

Types of Sets

Type Description

Fixed Set You select members manually

Dynamic Set Defined by conditions (e.g., Sales > 50K)

Creating Sets

1. **Right-click a field (e.g., Customer Name)**
2. **Choose Create > Set**
3. **Choose:**
   - **Manual selection (Fixed Set) o Condition tab (e.g., SUM(Sales) > 5000)**
   - **Top tab for Top N logic (e.g., Top 10 by Profit)**

Using Sets

- **Drag to filters shelf to limit data**
- **Place on Color shelf to contrast "In Set" vs "Out of Set"**
- **Combine with Set Actions (click to change views)**

Dynamic Behavior

- **Unlike groups, Dynamic Sets auto-update if the data changes.**

Set Actions in Dashboards

- **Interactive features where clicking on a region or chart updates other visuals using sets.**
- **Examples: Selecting one product category to filter charts using a set.**

77.3 Hierarchies

What Are Hierarchies?

Hierarchies allow you to define parent-child relationships in dimensions. They enable drill-down and roll-up capabilities in visuals.

Use Case

- **Analyze sales by Country → State → City**
- **Drill down into time: Year → Quarter → Month**
- **Navigate product breakdowns: Category → Sub-Category → Product**

Creating Hierarchies

1. **Drag one dimension on top of another in the Data Pane**

2. **Give a name to the hierarchy (e.g., Location)**
3. **Add more fields as needed and reorder if required**

Using Hierarchies in Views

- **When you use the top-level field (e.g., Country), Tableau shows "+" symbol**
- **Click the "+" to expand and see the next level**
- **Works well in bar charts, maps, tables, and tree maps**

Behavior

- **Maintains a structured path for drilling down**
- **Works across filters and actions**
- **Keeps dashboard clean by reducing initial complexity**

77.4 Calculated Fields

What Are Calculated Fields?

These are custom fields that you create using:

- **Mathematical operations**
- **Text functions**
- **Logical comparisons**
- **Date calculations**
- **Aggregates or row-level expressions**

Purpose

- **Derive new values**
- **Simplify repeated logic**
- **Perform custom comparisons**
- **Create flags or indicators**
- **Handle missing data or clean values**

Types of Calculations

Type	Description	Example
Row-level	Runs for each row of data	[Profit] / [Sales]
Aggregate	Works on aggregated data	AVG([Sales]), MAX([Order Date])
Boolean Logic	Conditional values	IF [Sales] > 10000 THEN 'High'
String	Text manipulation	LEFT([Product Name], 5)
Date	Date-based calculations	DATEDIFF('month', [Order Date], [Ship Date])

Table Calculations	Uses visual context	RUNNING_SUM(SUM([Sales]))

How to Create a Calculated Field

1. **Right-click in the Data Pane >Create > Calculated Field**
2. **Give it a name (e.g., Profit Margin)**
3. **Type your formula:**

[Profit] / [Sales]

4. **Click OK. If valid, Tableau adds it to your data pane.**

Examples

1. Profit Margin Calculation

[Profit] / [Sales]

2. Category Classification

IF [Sales] > 10000 THEN "Excellent"
ELSEIF [Sales] > 5000 THEN "Good"
ELSE "Low"

3. Customer Full Name (Concatenation)

[First Name] + " " + [Last Name]

4. Days to Ship

DATEDIFF('day', [Order Date], [Ship Date])

5. Discount Flag

IF [Discount] > 0 THEN "Discounted" ELSE "Full Price"

Summary: Groups vs Sets vs Hierarchies vs Calculated Fields

Feature	Purpose	Dynamic?	Example Use
Group	Combine labels manually	❌	Combine regions
Set	Create subsets of dimension values	✅	Top 10 products
Hierarchy	Create drill-down paths	✅	Country > City
Calculated Field	Create new fields using formulas	✅	Profit Margin, Date Differences

Chapter 78: Advanced Charts in Tableau 3 Treemap, Heatmap, Bullet Graphs

78.1 Treemap 3 Deep Dive

What is a Treemap?

A Treemap visualizes hierarchical data as a set of nested rectangles. The size and color of each rectangle represent different quantitative values.

When to Use:

- **To show parts of a whole (like pie charts but more space-efficient)**
- **To explore hierarchical data (e.g., Region → Country → City)**
- **To analyze two levels of comparison: value and another metric via color**

How to Create (Step-by-Step):

1. **Drag a categorical field (Dimension) like Product Category to Rows**
2. **Drag a measure like Sales to Size**
3. **Choose <Treemap= from the Show Me panel**
4. **Drag Profit to Color for additional encoding**
5. **Drag fields to Label (e.g., Category, Sales)**

Optional Enhancements:

- **Add Sub-Category to Detail to drill down within rectangles**
- **Use Calculated Fields to create KPIs for size or color**
- **Add Quick Filters to slice the Treemap dynamically**

Best Practices:

- **Avoid too many categories—Treemaps lose clarity if rectangles get too small.**
- **Use contrasting colors for better distinction.**
- **Combine with tooltips for context-rich interaction.**

78.2 Heatmap 3 Deep Dive

What is a Heatmap?

A Heatmap uses color density in grid cells to represent the intensity of a numeric value. It9s especially useful for identifying trends or outliers in categorical vs categorical comparisons.

When to Use:

- **Comparing two dimensions (e.g., Region vs Month)**
- **Showing distribution of high/low values**
- **Highlighting patterns in large datasets**

How to Create (Step-by-Step):

1. **Drag Region to Rows**
2. **Drag Month (or Order Date → Month) to Columns**
3. **Drag a measure like Profit to Color**

4. **Change Marks Type to Square or Circle**
5. **Drag the same measure (Profit) to Label**

Optional Enhancements:

- **Use Diverging Color Palettes for highlighting positive vs negative trends**
- **Add Tooltips for exact values**
- **Sort rows/columns to emphasize trends**

Advanced Customization:

- **Use Z-score normalization to show relative intensity (e.g., by row average)**
- **Use calculated bins to categorize performance levels**
- **Add Reference Lines for averages**

Best Practices:

- **Use color with purpose (e.g., green to red scale for performance)**
- **Ensure color contrast is clear for accessibility**
- **Avoid over-labeling—tooltips are often better than static labels**

78.3 Bullet Graph 3 Deep Dive

What is a Bullet Graph?

A Bullet Graph is a variation of the bar chart enhanced with:

- **Target lines**
- **Performance bands (qualitative ranges)**
- **Actual values (as a bar)**

Created by Stephen Few, bullet graphs are useful in KPI dashboards.

When to Use:

- **Showing actual vs target performance**
- **Tracking progress over time or across departments**
- **Replacing gauge charts or progress bars in a compact way**

How to Create (Step-by-Step):

1. **Create a calculated field for target if it9s not available (e.g., [Target])**
2. **Drag Region (or other dimension) to Rows**
3. **Drag Sales (or actual measure) to Columns**
4. **Drag Target to Detail**
5. **Go to Show Me → Choose Bullet Graph**
6. **Tableau automatically creates reference lines and bands**

Elements:

- **Bars = actual value**
- **Thin lines = target**
- **Background bands = performance zones Performance Bands via**

Reference Bands:

- **Right-click on the axis → Add Reference Line**

- Choose Band instead of Line
- Set ranges like:
  - 0–70% (Poor) o 70–90% (Average)
  - 90–100% (Excellent)

Optional Customizations:

- Add Conditional Coloring for performance (e.g., green for above target)
- Show labels only for underperforming regions
- Combine multiple KPIs vertically using a Dashboard layout

Best Practices:

- Use sparingly to avoid visual overload
- Maintain consistent axis ranges across comparisons
- Always label clearly or use intuitive tooltips

78.4 Comparative Summary

Chart Type	Use Case	Encoding Mechanism	Ideal For
Treemap	Visualizing part-to-whole + hierarchy	Size and Color	Showing large sets in compact space
Heatmap	Matrix of comparisons	Color gradient	Time vs category comparisons
Bullet Graph	KPI vs Target comparisons	Bars, Lines, Backgrounds	Business scorecards and executive views

Real-World Use Cases

Industry	Scenario	Recommended Chart
Retail	Sales by Category/Sub-Category	Treemap
Finance	Profit margins by Region and Month	Heatmap
Marketing	Campaigns performance vs target	Bullet Graph
Operations	Inventory level performance tracking	Bullet Graph
Education	Student performance across subjects	Heatmap

Chapter 79: Using Parameters and Dual-Axis Charts in Tableau

79.1 What are Parameters in Tableau?

Definition:

A parameter in Tableau is a dynamic input control that allows users to change the value of a variable used in calculations, filters, reference lines, or display controls. Parameters are not tied to data fields directly—they are independent values that you can reference in calculated fields, filters, sheets, or dashboards.

Use Cases:

- **Switch between measures (e.g., Sales vs Profit)**
- **Change chart views dynamically**
- **Set thresholds (e.g., Top N values)**
- **Create custom sort options**
- **Drive conditional logic**

How to Create a Parameter:

1. **Right-click in the Data Pane → Create Parameter**
2. **Set the following:**
    - **Name: e.g., "Select Measure" o Data type: Integer, Float, Date, String, Boolean**
    - **Allowable values:**
        - **Fixed list (you type options)**
        - **Range**
        - **All**
3. **Click OK**
4. **Right-click the parameter → Show Parameter**

Example: Switch Between Sales and Profit

1. **Create Parameter**
   Name: Select Measure
   Data type: String
   Allowable values: List
   - **o Sales o Profit**
2. **Create a Calculated Field Name: Chosen Metric**
3. **IF [Select Measure] = "Sales" THEN [Sales]**
4. **ELSEIF [Select Measure] = "Profit" THEN [Profit]**
5. **END**

6. **Use Chosen Metric in your chart instead of a direct field.**
7. **Now, when users select an option from the Parameter control, the chart updates.**

79.2 Advanced Parameter Examples

Use Case	Example
Top N Filter	Create parameter for <N= and filter [Index] <= [N]
Custom Reference Line	Use parameter to place a dynamic reference line (e.g., target sales)
Sorting Logic	Create a parameter with values <Sales=, <Profit=, <Quantity= and use a calculated field to change sort measure
Date Control	Let users pick a year or range of years to focus analysis

79.3 What is a Dual-Axis Chart?

Definition:

A Dual-Axis Chart (or Combo Chart) allows you to display two different measures with two independent axes—usually left (primary) and right (secondary)—in the same chart.

This is useful when the measures differ in units or scale, such as comparing Sales (in dollars) vs Profit Ratio (in %).

How to Create a Dual-Axis Chart:

1. **Drag Order Date to Columns**
2. **Drag Sales to Rows**
3. **Drag Profit to Rows as well, placing it next to Sales**
4. **Tableau will create two separate charts**
5. **Right-click the second axis → Dual Axis**
6. **Right-click on the axis again → Synchronize Axis (optional)**
7. **Change Marks type independently:**
   - **o Line for one o Bar for another**

Use Cases of Dual-Axis Charts:

Use Case	Example
Compare Revenue and Profit Margin	Sales (bar) vs Profit Ratio (line)

Track Trends Over Time	**Units Sold vs Sales Amount**
Show Growth & Baseline	**Current Year Sales vs Last Year Sales**
Performance Metrics	**Target (line) vs Actual (bar)**

79.4 Customization Tips

Axis Synchronization:

If both axes use the same units (e.g., two currency fields), always synchronize axes to avoid misinterpretation.

Color & Marks:

- **Use contrasting colors and mark types (e.g., bar vs line) for clarity.**
- **Customize tooltips for each axis separately.**

Dual-Axis Map:

You can even layer two maps (e.g., one showing filled regions, another showing symbols/points) using dual-axis.

79.5 Combined Example 3 Using Parameter with Dual-Axis

Goal: Allow the user to choose which KPI (Sales, Profit, Quantity) to compare with a fixed field (e.g., Discount)

1. **Create Parameter: Select KPI o List: Sales, Profit, Quantity**
2. **Calculated Field: Dynamic KPI**
3. **IF [Select KPI] = "Sales" THEN [Sales]**
4. **ELSEIF [Select KPI] = "Profit" THEN [Profit]**
5. **ELSE [Quantity]**
6. **END**
7. **Dual-Axis Setup:**
   - **o Axis 1: Dynamic KPI (Line) o Axis 2: Discount (Bar) o Dual-Axis enabled, marks customized**

8. Show Parameter and dashboard becomes interactive.

79.6 Best Practices

Topic	Recommendation
Parameters	Limit to user-friendly options; use calculated fields for flexibility
Dual-Axis	Avoid clutter—use only when axes are logically distinct
Synchronization	Use carefully; unsynchronized axes can mislead
Dashboard Use	Parameters + dual-axis can make dashboards highly dynamic and compact
Accessibility	Label both axes clearly and avoid excessive overlapping

Chapter 80: Dashboard Design and Interactivity in Tableau

80.1 What is a Dashboard in Tableau?

A dashboard in Tableau is a collection of multiple visualizations (views) combined into a single screen. It enables users to analyze, compare, and interact with different data elements in one place.

Dashboards can include:

- **Charts**
- **Filters**
- **Maps**
- **KPIs**
- **Images**
- **Web content**
- **Parameters**
- **Buttons and navigation**

80.2 Creating a Dashboard in Tableau

Steps to Create a Dashboard:

1. **Click on the <New Dashboard= button at the bottom tab bar.**
2. **The dashboard workspace will appear.**
3. **Use the left pane to drag:**
    - **Sheets (charts/tables/maps)**
    - **Objects (text, images, web, buttons, blank spaces)**

4. Adjust layout using:

- **Floating or Tiled mode o**
  Resize sheets manually o
  Drag to rearrange

80.3 Tiled vs Floating Layouts

Type	Description
Tiled	Elements snap into a grid. Easy to align and resize. Recommended for structured dashboards.
Floating	Elements can be placed anywhere, overlapping allowed. Offers more design freedom. Ideal for custom layouts or layered design.

You can mix both for flexibility.

80.4 Dashboard Sizing

Choose a size that matches your target screen:

Fixed Size (e.g., 1366×768) – Best for consistent layouts

Automatic – Adapts to screen, but can misalign elements

 Range – Minimum and maximum bounds for responsiveness

80.5 Adding Interactivity

Interactivity turns a static dashboard into a dynamic analysis tool. Below are key techniques:

80.5.1 Filters

- Add filters from any worksheet
- Choose whether filters apply to:
  - This worksheet o Selected worksheets
  - All using this data source

Example: Filter sales by region or category dynamically

80.5.2 Highlight Actions

- Used to emphasize data points across charts
- When a user hovers or clicks on a category in one chart, related data in other charts is highlighted

Example: Hovering over a segment in a pie chart highlights matching bars in a bar chart

80.5.3 Filter Actions

- Makes one visualization act as a filter for another
- Go to Dashboard > Actions > Add Action > Filter

Example: Clicking on a region in a map filters the bar chart below to show data for that region only

80.5.4 URL Actions

- Launch a website or report when a mark is clicked
- Useful for external links, product pages, or dynamic searches

Example: Clicking on a customer name opens their profile page

80.5.5 Parameter Actions

Change parameter values using a user action

Use for dynamic metric selection, switching dimensions, etc.

Example: Click on a bar to change the parameter value and switch what metric is shown in another chart

80.5.6 Set Actions

- Add or remove data from a dynamic set when interacting with a view

- **Great for on-the-fly segmentation**

Example: Click on top-performing products to see how they behave across regions

80.6 Navigation Buttons and Sheet Swapping

Buttons and navigation features can make dashboards behave like applications.

Navigation Buttons:

- **Add a Button Object**
- **Link to another dashboard, worksheet, or story**
- **Customize text/icon and style**

Sheet Swapping:

- **Use a parameter to switch between different views**
- **All views are placed on top of each other with a filter applied to show one at a time Example: Switch between a bar chart and a map using a dropdown**

80.7.Best Practices for Dashboard Design

Area	Best Practice
Layout	Align charts logically (left to right or top to bottom), keep important charts larger
Consistency	Use consistent color schemes, fonts, and formatting
Tooltips	Keep concise, but informative; customize if necessary
Color	Use intuitive color scales (e.g., red for loss, green for gain)
Performance	Use extract instead of live connections when possible, minimize quick filters
White Space	Use spacing and separators to group related elements

Titles & Legends	Label everything clearly and logically
KPIs	Put key metrics at the top for quick access
Responsive Design	Use fixed sizes for shared dashboards or mobile-specific versions

80.8.Mobile Dashboard Design

- Tableau allows designing for Phone, Tablet, and Desktop views
- Click <Device Preview= in dashboard settings
- Tailor layout for each device
- Resize, hide, or rearrange sheets and objects per device

80.9.Dashboard Performance Optimization

Strategy	Description
Minimize Worksheets	Only use what is necessary to reduce load time
Use Extracts	Replace live connections with data extracts where possible
Reduce Quick Filters	Each filter adds query load
Avoid High Granularity	Aggregated data loads faster
Use INDEX() and RANK() wisely	Complex table calculations can slow performance
Clean Up Tooltips	Avoid excessive fields in tooltips

80.10.Putting It All Together: A Dashboard Flow Example

Goal: Regional Sales Performance Dashboard Top KPI Area: o Total Sales o Profit o Discount Rate Middle Section:

- Map of regions o Bar chart of sales by category
  - Bottom Section:
- Line chart showing trend over time

Interactivity:

- Map Click: filters all charts to selected region
- Category Bar Hover: highlights in trend line
- Parameter Selector: allows users to choose what metric is shown in the line chart Navigation Button: goes to detailed product dashboard

Chapter 81: Story Points and Mobile View Design in Tableau

81.1 What Are Story Points in Tableau?

Story Points in Tableau are a sequence of visualizations that work together to tell a datadriven narrative. It allows you to guide your audience through a step-by-step analysis, much like slides in a presentation.

81.1.1 Key Components of a Story

Each Story Point is made of:

- A Dashboard or Worksheet (the main content)
- A Caption (text at the top to describe the insight)
- Story Navigation Panel (for jumping between points)

81.1.2 Why Use Story Points?

- To explain decisions or findings in a structured way.
- To highlight key patterns or trends across time or segments.
- To provide context with visual evidence.
- To let users explore insights in an ordered format.

81.2 Creating a Story in Tableau

Step-by-Step Guide:

1. Click on "New Story" at the bottom tab bar.
2. Tableau opens a new canvas with: o Blank story point o Caption area (editable)
   o Side panel with available dashboards and sheets
3. Drag a sheet or dashboard onto the blank space.
4. Edit the caption at the top to describe this insight or point.
5. Click <Add a New Story Point= to create the next step in the narrative.
6. Repeat to build a full story flow.

81.2.1 Editing and Customizing Story Points

- Duplicate Points: To create slight variations
- Update Views: Change filters/parameters per point
- Resize: Choose story size (Desktop, Tablet, Phone)
- Hide/Show Navigation: Toggle the story index

81.3.Best Practices for Story Points

Area	Best Practice
Focus	Each point should address one clear insight
Order	Start with big picture, then drill down
Captions	Use action words (e.g., "Sales Declined in Q4")
Brevity	Don9t overload each point with too many visuals
Interaction	Consider using filters and highlights to tell dynamic stories
Device Size	Choose size based on presentation screen or audience device

81.4.Mobile View Design in Tableau

What is Mobile Design?

Mobile design in Tableau allows you to customize dashboards and stories for viewing on phones and tablets. Mobile users need a different layout due to smaller screens.

81.4.1 Accessing Device Designer

1. **Go to your dashboard.**
2. **Click on <Device Preview= from the dashboard menu.**

3. **Tableau displays: o Desktop o Tablet o Phone (portrait or landscape)**

81.4.2 Customizing for Mobile

You can create different versions of your dashboard for each device:

Device	Use Case
Desktop	Full dashboards with detailed visuals
Tablet	Medium-complex dashboards, easier navigation
Phone	Summarized KPIs and simplified visuals

Tip: Tableau auto-generates a mobile layout, but you should customize it.

81.4.3 Mobile Layout Adjustments

- **Hide non-essential charts**

- **Reposition charts vertically for scrolling**
- **Use larger fonts/buttons for touch screens**
- **Ensure filters/parameters are easily accessible**
- **Avoid floating objects unless necessary**

81.5.Best Practices for Mobile Dashboards

Area	Tip
Layout	Use vertical stacking instead of side-by-side
Font Size	At least 14px for readability
KPIs	Put top metrics at the top
Buttons	Use icons and navigation buttons to save space
Performance	Keep it light – fewer charts = faster loads
Interactivity	Use dropdowns over radio buttons or checkboxes

81.6.Common Use Case: Executives on the Go

Example:

- **Create one dashboard for desktop with full detail**
- **Create a mobile version with:**
  - **Total Revenue**
  - **Profit Trend**
  - **Alerts/flags (e.g., regions below target)**
  - **A button to call/email the sales team**

Use mobile layout to make it touch-friendly, readable, and actionable from a phone.

81.7 Summary

Feature	Story Points	Mobile View
Purpose	Telling a narrative	Device-specific user experience
Structure	Sequential slides	One dashboard in various formats
Audience	Presentations, analysts	Executives, sales teams, field workers
Interaction	Less dynamic, more guided	High touch-based interaction
Customization	Per slide/point	Per device type

Chapter 82: Forecasting, Trend Lines, Reference Bands in Tableau

82.1 Introduction

Forecasting, trend lines, and reference bands are powerful analytical tools in Tableau that help users:

- Predict future values based on historical data
  Understand underlying patterns and trends
- Visually guide viewers by showing limits or benchmarks

These features enhance data storytelling, especially for business planning, performance analysis, and executive reporting.

82.2 Forecasting in Tableau

82.2.1 What Is Forecasting?

Forecasting in Tableau uses time series data to project future values based on past trends and seasonality.
It uses exponential smoothing models internally.

82.2.2 How to Add a Forecast

1. Drag a Date field to the Columns shelf (e.g., Month or Year).
2. Drag a Measure like Sales or Profit to the Rows shelf.
3. Go to the Analytics Pane (right sidebar).
4. Drag and drop Forecast into the view.

82.2.3 Customizing Forecast

Right-click on the forecast line → choose Forecast → Forecast Options:

Option	Description
Forecast Length	How far into the future to forecast
Forecast Model	Automatic or Custom (trend, seasonality, etc.)
Season Length	Set custom periodic cycles (e.g., 12 for months)
Ignore Last	Ignores recent incomplete periods (e.g., current month)
Prediction Interval	Shows confidence range (e.g., 95%)

82.2.4 Understanding Forecast Output

- Solid Line: Historical data
- Dashed Line: Forecasted values

Shaded Area: Confidence range (e.g., 95% prediction interval)

Use Case: Sales forecasting for the next 6 months using historical monthly data.

82.3 Trend Lines in Tableau

82.3.1 What Is a Trend Line?

A trend line shows the underlying pattern or direction of the data—linear, exponential, or polynomial trends.

Useful for identifying correlation between dimensions and measures.

82.3.2 How to Add a Trend Line

1. Create a scatter plot or a line chart (e.g., Profit vs Sales).
2. Open the Analytics Pane.
3. Drag Trend Line and drop it on the view.

82.3.3 Types of Trend Lines

Type	Use
Linear	Straight-line trend
Logarithmic	For values that grow quickly and then level off
Exponential	For continuously increasing values
Polynomial	For curved or changing trends

82.3.4 Customizing Trend Lines

Right-click the trend line → Edit Trend Lines:

Show R-squared Value: Indicates how well data fits the model

- Show P-value: Indicates statistical significance
- Force Intercept: Fix the trend line to start at a specific point

82.3.5 R-Squared and P-Value

Metric	Meaning

R^2 (R-squared)	Value between 0–1. Higher means better fit
P-value	Significance of trend. Lower than 0.05 = statistically significant

82.4 Reference Lines and Bands

82.4.1 What Are Reference Lines/Bands?

These are visual guides added to charts to indicate:

- Target values
- Averages
- Benchmarks
- Thresholds (upper/lower limits)

They do not change data, only improve readability and context.

82.4.2 Types of Reference Guides

Type	Description
Reference Line	Single value across axis
Reference Band	Shaded region between two values
Reference Distribution	Percentiles (e.g., 25%, 75%) shown on chart

82.4.3 How to Add Reference Lines

1. Right-click the axis or open Analytics Pane
2. Drag Reference Line, Band, or Distribution
3. Drop it on the desired axis (Table, Pane, or Cell level)

82.4.4 Customizing Reference Lines

- Scope: Table, Pane, Cell (entire view, row group, or cell)
- Label: Value, custom text, none
- Line Style: Solid, dashed, dotted
- Tooltip: Show explanation when hovered

82.4.5 Example Use Cases

Visual Type	Reference Use
Sales Bar Chart	Add a line for Sales Target

Profit Trend	Show average profit as reference line
Scatter Plot	Use bands to define performance zones

82.5 Combining Forecast, Trend, and Reference

Use Case: Sales Performance Dashboard

- Add a forecast to show next 3 months of projected sales
- Overlay a trend line to show the upward/downward slope
- Include reference line for monthly sales target
- Add reference band between 10% and 20% profit margin

This creates a visually enriched, data-driven dashboard to inform action.

82.6.Best Practices

Tip	Reason
Validate Historical Data	Forecasting relies heavily on accurate past data
Don9t Mix Too Many Lines	Avoid visual clutter
Use Reference Lines Sparingly	Highlight key targets, not every metric
Label Clearly	Let users know what each line/band means
Combine with Tooltips	Add descriptions for user context

82.7.Summary Table

Feature	Purpose	Tool Used
Forecast	Predict future values	Exponential smoothing
Trend Line	Show pattern or relationship	Linear/log models
Reference Line	Benchmark performance	Manual or calculated
Reference Band	Define safe zones	Min/Max ranges

Chapter 83: Level of Detail (LOD) Calculations in Tableau

83.1 What Are LOD Calculations?

LOD (Level of Detail) calculations in Tableau allow you to control the granularity at which aggregations are performed independently of the view. This means you can calculate values at a different level of detail than what's displayed.

For example, you can:

- Show average sales per customer, even if the view is at the region level.
- Fix a value at a specific dimension level.
- Compare individual row values to overall averages.

83.2 Why Use LOD Calculations?

Traditional aggregations (like SUM([Sales])) are affected by the visual structure of the worksheet. LODs provide a way to break free from the visual context and create accurate, flexible calculations.

Use LOD when you need to:

- Aggregate data at a different level than the current view
- Compare part-to-whole
- Normalize metrics
- Build advanced KPIs like % of total across a fixed level

83.3.Syntax of LOD Expressions

LOD expressions follow this format:

{ <LOD Type> [Dimension1], [Dimension2], ... : AGG([Measure]) }

There are three types of LOD expressions:

Type	Purpose
FIXED	Aggregates at the specified dimension(s)
INCLUDE	Adds extra dimensions to the view's context
EXCLUDE	Removes dimensions from the view's context

83.4.FIXED LOD Expression

Syntax:

{ FIXED [Dimension] : AGG([Measure]) }

What It Does:

Calculates the value at the exact dimension(s) listed, regardless of what9s shown in the view.

Example:

{ FIXED [Customer Name] : SUM([Sales]) }

This calculates total sales per customer, no matter what dimensions are in the current view.

Use Case:

To compare individual order sales with the customer9s total sales.

83.5.INCLUDE LOD Expression

Syntax:

{ INCLUDE [Dimension] : AGG([Measure]) }

What It Does:

Adds extra dimension(s) to the aggregation level in addition to what's in the view.

Example:

{ INCLUDE [Product Name] : AVG([Sales]) }

Even if your view is at the Category level, this gives the average sales per product under each category.

Use Case:

To drill down and get finer granularity without changing the visual layout.

83.6.EXCLUDE LOD Expression

Syntax:

{ EXCLUDE [Dimension] : AGG([Measure]) }

What It Does:

Removes one or more dimensions from the current visual grouping before performing aggregation.

Example:

{ EXCLUDE [Region] : SUM([Sales]) }

This removes the Region dimension from the aggregation, effectively computing a higherlevel total (e.g., national total) even if the view shows breakdowns by region.

Use Case:

To calculate a grand total or an average ignoring finer groupings.

83.7 Practical Examples

83.7.1 % of Total Sales by Customer

SUM([Sales]) /

{ FIXED : SUM([Sales]) }

This gives each customer9s sales as a percentage of total sales across the dataset.

83.7.2 Rank of Product in Category

RANK({ FIXED [Category], [Product Name] : SUM([Sales]) })
This ranks each product within its category based on total sales.

83.7.3 Difference from Customer9s Average Sales

SUM([Sales]) -
{ FIXED [Customer Name] : AVG([Sales]) }
This compares each order to that customer9s average sales, useful for performance tracking.

83.8 LOD vs Table Calculations

Feature	LOD Calculation	Table Calculation
Based on	Underlying data level	Visual layout
Scope	Fixed, flexible	Tied to the view
Use Case	Complex KPIs, % of total	Quick ratios, moving avg
Performance	Efficient with extracts	Can be slower with large data

83.9 Nested LOD Calculations

You can nest LOD expressions for complex logic:

```
{ FIXED [Category] :
   AVG(
      { FIXED [Category], [Sub-Category] : SUM([Sales]) }
```

This calculates the average sub-category sales per category.

83.10 Tips and Best Practices

Tip	Why
Use FIXED for predictable aggregations	Works outside of the view context
INCLUDE is great for drill-down	Maintains visual hierarchy

Avoid over-nesting	**Can affect performance**
Use meaningful naming	**Helps in debugging**
Use LOD for KPIs	**Great for year-over-year or % of total logic**

83.11 Summary Table

LOD Type	Behavior	Example Use
FIXED	**Ignores view, uses listed dimensions**	**Customer-level metrics**
INCLUDE	**Adds dimensions temporarily**	**Average per product**
EXCLUDE	**Removes dimensions from view**	**Regional vs. national comparisons**

Chapter 84: Geo Maps in Tableau 3 Symbol Maps, Filled Maps, and Custom Geocoding

84.1 Introduction to Geo Maps in Tableau

Tableau provides powerful geographic visualization capabilities that allow you to map data spatially. Geo maps are ideal when your data includes location-based fields like country, state, city, postal code, or coordinates (latitude/longitude).

Geo maps help to:

- Understand regional trends
- Visualize spatial patterns
- Perform comparative analysis across locations

84.2 Tableau's Built-In Geocoding

Tableau automatically recognizes geographic roles in your data, such as:

- Country
- State/Province
- City
- Postal Code
- Latitude / Longitude

When Tableau detects these, it assigns them a Geographic Role, enabling automatic plotting on a map.

You can right-click on any geographic field and select "Geographic Role" to verify or change it.

84.3 Creating Your First Map in Tableau

Step-by-Step:

1. Connect to your dataset.
2. Drag a geographic dimension (like State) to the Rows or view area.
3. Tableau will automatically generate a map.
4. Drag a measure (like Sales) to Size, Color, or Label shelf to show metrics.

84.4 Symbol Maps

What Are They?

Symbol Maps display circles or markers at geographic points (cities, zip codes, etc.).

How to Create:

1. Drag a geographic field like City to the view.
2. Drag Sales to Size or Color.
3. Tableau places symbol marks (by default, circles) on the corresponding location.

4. **More sales = larger or darker circles.**

Use Case:

To compare location-specific performance, such as sales per city or number of customers per branch.

Customization Options:

- **Change shapes of symbols**
- **Adjust color scales**
- **Add labels to markers**
- **Use tooltips for interactivity**

84.5 Filled Maps (Choropleth Maps)

What Are They?

Filled Maps color the entire region (like state or country) based on a measure value.

How to Create:

1. **Drag a geographic field like State to the view.**
2. **Tableau builds a map with colored areas.**
3. **Drag Profit or Sales to Color.**

Use Case:

To compare aggregated values across broader geographic areas, like comparing total profit by state.

Customization Options:

- **Adjust color gradient (e.g., red for low, green for high)**
- **Add labels like total sales**
- **Combine with filters or dashboards for more insight**

84.6 Custom Geocoding in Tableau

Why Use Custom Geocoding?

Sometimes your data includes locations Tableau doesn9t recognize, like:

- **Store IDs with coordinates**
- **Custom territories**
- **Delivery zones**

Options for Custom Geocoding:

1. Using Latitude & Longitude

If you have specific latitude and longitude values:

- **Import them into Tableau as fields**
- **Drag Latitude to Rows and Longitude to Columns**
- **Tableau will map exact locations using symbols**

2. Creating a Custom Geocoding File (Advanced)

- **Use CSV files structured by Tableau9s geocoding format.**

- **Place the custom geocoding folder in Documents > My Tableau Repository > Local Data.**
- **Reload Tableau and the new geographies will be available.**

3. Grouping Locations into Custom Territories

- **Right-click on geographic field >Create > Group**
- **Assign locations to your own regions like North Zone, South Zone**
- **These grouped territories can now be mapped. 4. Using Shapefiles or Spatial Files**

You can import:

- **ESRI Shapefiles**
- **KML/KMZ files**
- **GeoJSON files**
- **Tableau can natively read these to generate complex spatial visualizations.**

To import: go to "Connect" > "Spatial file" on Tableau start screen.

84.7.Advanced Map Features

Feature	Description
Map Layers	Add data layers such as streets, terrain, and population density
Dual-Axis Maps	Combine two maps on top of each other (e.g., symbol + filled map)
Map Legends	Display color or size legends dynamically
Zoom & Pan	Users can interactively zoom in or out of regions
Background Map Options	Switch from light, dark, to satellite backgrounds
Filter Actions	Clickable maps that filter dashboards based on location

84.8.Example Use Cases

Sales by Region:

- **Use a filled map to color-code states by sales**
- **Add tooltips to display exact amounts**

Store-Level Performance:

- **Use symbol maps with size representing sales per store**
- **Use color to represent profit margin Delivery Analysis:**
- **Import spatial file of delivery zones**

- Color zones by average delivery time

International Business:

- Map global sales with custom country groupings
- Use parameters to switch between year-to-year comparisons

84.9.Best Practices for Map Design

Tip	Why It Matters
Don9t overuse colors	Too many gradients can confuse users
Add interactivity	Tooltips, filters, and highlights improve user experience
Use dual-axis when necessary	To combine different data layers (e.g., customer vs. store)
Simplify maps	Focus only on needed geographic detail
Always label	Include legends, titles, or data labels for clarity

84.10.Common Errors and Fixes

Error	Solution
Blank Map	Make sure geographic role is assigned properly
Duplicates	Use aggregation (like COUNTD) to control data
Incorrect Locations	Check if Tableau recognizes the location (switch geographic role if needed)
Missing Coordinates	Provide Lat/Long values manually

84.11.Summary

Map Type	Best For	Example
Symbol Map	Point-based comparisons	Sales per City
Filled Map	Region-based aggregation	Profit by State
Custom Geocoding	Custom areas or coordinates	Store Zones, Warehouses
Spatial Files	Complex shapes or boundaries	Delivery Areas, Zip Codes

Chapter 85: Cluster Analysis in Tableau

85.1 What is Cluster Analysis?

Cluster Analysis is a data segmentation technique that groups data points (like customers, products, or regions) into clusters so that items in the same group are more similar to each other than to those in other groups.

In Tableau, clustering helps identify hidden patterns by segmenting data based on selected variables. It is widely used for:

- Customer segmentation
- Sales territory planning
- Product performance grouping
- Market analysis

85.2 Understanding the Concept of Clustering

Cluster analysis answers questions like:

- Which products sell similarly?
- Which customer groups behave alike?
- Which regions perform in similar ways?

It uses an unsupervised machine learning technique, generally based on K-Means Clustering, where:

- "K" = the number of clusters
- It minimizes the distance between points within a cluster and maximizes the distance between clusters.

85.3 When to Use Clustering in Tableau

You should use clustering when:

- You want to identify patterns in multi-dimensional data.
- You need to segment your data automatically.
- You want to group entities (like customers or stores) with similar behavior or metrics.

85.4 How to Perform Cluster Analysis in Tableau

Step-by-Step Guide:

Step 1: Open a Worksheet

- Load your dataset into Tableau and navigate to a worksheet. Step 2: Drag Dimensions/Measures to Rows and Columns

Example:

- Drag Sales to Columns
- Drag Profit to Rows

Step 3: Drag 8Cluster9f rom Analytics Pane

- **Go to the Analytics Pane (next to the Data Pane).**
- **Drag Cluster into the view.**
- **Tableau will automatically suggest a number of clusters based on internal calculations.**

Step 4: Adjust Clustering Variables

- **In the Clustering dialog box, you can add or remove variables (dimensions/measures) that Tableau uses to group the data.**
- **Click on "More Options" > Edit Cluster" to adjust which fields are used for clustering.**

Step 5: Analyze Clusters

- **Tableau color-codes each cluster.**
- **Hover to see which data points fall in which cluster.**

85.5 Example Use Case

Customer Segmentation

Suppose you have customer data with:

- **Sales**
- **Profit**
- **Order Frequency**

Steps:

1. **Use these fields in the clustering dialog.**
2. **Tableau may form 3 clusters like:**
    - **Cluster 1: High Sales, High Profit (Premium customers)**
    - **Cluster 2: Medium Sales, Medium Profit**
    - **Cluster 3: Low Sales, Low Profit (Low-value customers)**

This helps your marketing team target clusters differently.

85.6.Key Features of Tableau Clustering

Feature	Description
Drag-and-Drop Clustering	Easy to apply clustering with drag-drop
Editable Variables	Select which fields to include in the clustering algorithm
Dynamic Updates	Automatically updates as filters/slicers are used
Cluster Field	Becomes a new dimension field, usable in other sheets

85.7.Adding Clusters to Other Visualizations

Once the clusters are created:

1. **Right-click on the cluster field.**
2. **Choose "Group" or "Create Set" to use clusters in:**
   - **Filters**
   - **Dashboards**
   - **Calculated fields**
   - **Tooltips**

85.8.Cluster Analysis Limitations in Tableau

Limitation	Notes
Only works with continuous variables	You can9t cluster using text dimensions like region or product name directly
Number of clusters	You can9t manually choose the exact K — Tableau suggests a good K
No advanced algorithms	Only K-Means, no hierarchical or DBSCAN methods
No scoring metrics shown	Tableau doesn9t show cluster performance metrics like silhouette score, WCSS, etc.

85.9.Tips and Best Practices

Tip	Benefit
Normalize data	Scale features (especially if on different ranges) to avoid biased clustering
Remove outliers	Outliers may distort cluster centers
Start with 2–4 clusters	Too many clusters can complicate insights
Visualize in scatter plots	Helps to better understand cluster distribution
Use tooltips	Add descriptive details like totals and averages inside each cluster

85.10.Real-World Use Cases

Use Case	Description
Retail Segmentation	Group customers by purchase size and frequency
Marketing Campaigns	Target segments with different promotions
Product Management	Group similar products based on return rates and reviews
Banking	Segment accounts based on balance, tenure, and transactions
Logistics	Cluster warehouses based on volume and distance from delivery points

85.11.Using Clustering with Filters and Dashboards

Clusters are responsive to filters:

- If you filter by region, clustering will recalculate for that subset.
- This makes clusters more context-aware.

You can place clusters in:

- Legends (to color charts)
- Filters (to isolate a group)
- Tooltips (to show details about a cluster)

85.12 Exporting Cluster Fields

After creating a cluster:

- Right-click on it >Convert to Group
- Save it as a new field
- Use in calculated fields or data blending

85.13.Summary

Aspect	Details
Purpose	Group similar data points
Based On	Measures like Sales, Profit, Frequency
Technique	K-Means (auto-determined)

Result	**A new "Cluster" field**
Used In	**Segmentation, targeting, performance grouping**

85.14.Final Thought

Clustering is a powerful feature in Tableau that requires no coding or complex modeling. It allows business users to apply machine learning-level segmentation in just a few clicks. When used properly, cluster analysis can unlock powerful insights and support targeted decision-making across marketing, sales, operations, and beyond.

Chapter 86: Tableau Prep for Data Cleaning

86.1 What is Tableau Prep?

Tableau Prep is a powerful data preparation tool designed by Tableau to help users clean, combine, and shape data before analyzing it in Tableau Desktop or Tableau Cloud.

It provides an interactive, visual, and intuitive interface where users can:

- **Clean messy data**
- **Join and union multiple data sources**
- **Apply calculations and filters**
- **Pivot and reshape data**
- **Create reusable data pipelines**

The main product is called Tableau Prep Builder, which is typically installed on your desktop, and its outputs can be scheduled and run using Tableau Prep Conductor (available in Tableau Server/Tableau Cloud).

86.2 Why Use Tableau Prep?

Data is often dirty, inconsistent, or unstructured, making it difficult to analyze. Tableau Prep simplifies the process of:

- **Removing unnecessary columns**
- **Fixing nulls and formatting issues**
- **Renaming fields**
- **Grouping similar values**
- **Ensuring consistent data structure across sources**

It is ideal for non-technical users who want to clean and structure data visually instead of using SQL or scripts.

86.3.Tableau Prep Interface Overview

Component	Description
Flow Pane	A visual flow of all cleaning, joining, and shaping steps
Input Step	Where data sources are brought into Tableau Prep
Profile Pane	Visual profile of fields with automatic data profiling
Data Grid	Shows actual records in tabular form

Cleaning Step	**Place where most data transformations happen**
Output Step	**Where cleaned data is exported as .hyper, .tde, or .csv**

86.4.Common Data Cleaning Tasks in Tableau Prep

1. Removing Nulls

- **Tableau Prep visually highlights null values.**
- **You can filter them out or replace them using:**
  - **"Remove Nulls"**
  - **"Fill Nulls with Constant"**
  - **"Use LOD Calculation"**

2. Renaming Fields

- **Rename columns directly in the profile pane.**
- **Helps create clear and meaningful field names for later analysis.**

3. Changing Data Types

- **Easily convert between Text, Date, Number, etc.**
- **Ensures compatibility during joins and calculations.**

4. Grouping and Replacing

- **Automatically or manually group similar values (e.g., "NY", "New York", "N.Y." → "New York").**
- **Options:**
  - **Manual Group**
  - **Group and Replace Using Spelling**
  - **Group and Replace Using Pronunciation**

5. Filtering

Remove unwanted rows based on condition:

- **Values greater than X**
- **Records that match a string**
- **Dates within a range**

6. Splitting Fields

- **Split fields based on delimiters (e.g., split "Full Name" into "First Name" and "Last Name").**
- **Custom split logic is also supported.**

7. Creating Calculated Fields

- **Create new columns using formulas.**
- **Example:**
- **IF [Sales] > 1000 THEN "High" ELSE "Low" END**

- **You can use string, number, date, logic, and type conversion functions.**

86.5 Joins and Unions in Tableau Prep

Joins

- **Combine data from two tables based on a key field.**
- **Supports Inner, Left, Right, and Outer joins.**
- **Visual interface shows overlap in fields and join results.**

Unions

- **Stack datasets with the same column structure vertically.**
- **Good for combining monthly Excel files or multiple exports.**

86.6 Pivoting Data

Pivot Rows to Columns

- **For example, turn:**
- **Category | Month | Sales**
- **Furniture | Jan | 100**

Into:

Category | Jan | Feb | Mar

Pivot Columns to Rows

- **Flatten wide data into long format (ideal for Tableau visualizations).**

86.7 Data Profiling in Tableau Prep

Profile Pane shows:

- **Value distribution of each field**
- **Nulls, outliers, field types, and common values**
- **Enables filtering and cleaning directly from visual summaries**

This helps users quickly spot and fix data issues before they cause problems in dashboards.

86.8.Flow Steps in Tableau Prep

Step Type	Description
Input	Load one or more data sources
Clean	Apply data transformations
Join	Combine tables based on key
Union	Append tables
Pivot	Reshape data

Script	**Run R or Python scripts (for advanced logic)**
Output	**Export cleaned data to Tableau Extract (.hyper), CSV, or Tableau Server**

86.9.Output Options

You can export your cleaned data in the following formats:

- **.hyper (Tableau Extract)**
- **.csv**
- **Publish directly to Tableau Server or Tableau Cloud**

Also, flows can be scheduled using Tableau Prep Conductor (part of Tableau Server/Data Management Add-On).

86.10 Automation with Tableau Prep Conductor

- **Prep Conductor automates flow execution.**
- **Enables scheduling and monitoring flows.**
- **Useful for refreshing data pipelines daily or hourly.**

Example:

- **Clean Sales data every night at 12 AM → Automatically updates Tableau dashboards by morning.**

86.11.Real-World Use Cases

Industry	Use Case
Retail	**Clean product sales data from 20 store files**
Finance	**Merge expense reports and format values**
Healthcare	**Remove null patient data and standardize diagnoses**
Education	**Pivot and clean test scores from different formats**
Logistics	**Union monthly shipment logs and clean missing entries**

86.12.Best Practices

Tip	Why
Use clear naming conventions	Helps when building complex flows
Break steps into chunks	Easier to debug and maintain
Document flows	Add annotations in each step
Keep flows reusable	Design with future updates in mind
Monitor performance	Especially with large files or joins

86.13.Tableau Prep vs Power Query (Power BI)

Feature	Tableau Prep	Power Query
Interface	Visual and flow-based	Excel-like steps pane
Output	Tableau Hyper, CSV	Excel, Power BI Datasets
Performance	Excellent with Hyper Engine	Good, especially for Microsoft stack
Reusability	Flows can be reused and scheduled	Queries can be refreshed
Integration	Best with Tableau ecosystem	Best with Microsoft ecosystem

86.14.Summary

Component	Description
Tool Name	Tableau Prep Builder
Purpose	Clean, combine, and prepare data visually
Key Actions	Clean, Join, Union, Pivot, Calculate, Filter
Outputs	Hyper files, CSV, Server/Cloud

Benefits	**Visual prep, no-code logic, powerful automation**

86.15.Final Thoughts

Tableau Prep is a must-have tool for anyone working with dirty, complex, or unstructured data in Tableau. It empowers analysts, even non-technical users, to build robust, clean, and dynamic datasets ready for analysis.

Instead of spending hours writing SQL or manipulating Excel files, Tableau Prep allows you to build, clean, and automate data pipelines with a few intuitive clicks—making the data prep process faster, visual, and reusable.

Chapter 87: Exporting Reports and Publishing in Tableau

87.1 Introduction

After designing dashboards and visualizations in Tableau, the final step is often to export or publish your work so others can view and interact with it. Tableau provides multiple ways to share your insights, ranging from exporting to static formats (like PDF or images) to interactive online dashboards via Tableau Server or Tableau Public.

87.2 Exporting Options in Tableau Desktop

1. Export as Image

- Exports the current dashboard or worksheet as a static image.
- Format: .png
- Use Case: For inserting visuals into reports, PowerPoint, or emails.

How to:

- Go to: File > Export > Image...
- Choose the resolution and where to save.

2. Export as PDF

- Exports dashboard or workbook into PDF format for printing or offline viewing.
- Can include all dashboards or just selected views.

How to:

- File > Export as PDF
- Options to configure:
  - Layout (Portrait or Landscape) o Paper size (A4, Letter, etc.) o Include/Exclude tabs, title, etc.

3. Export Data

- Exports underlying or summarized data used in a worksheet.

How to:

- Right-click on a chart > View Data
- Then click Export All or Export Summary
- Format: .csv

Use Case: When the user wants access to raw data for further analysis.

4. Copy and Paste

- You can right-click a chart or worksheet and copy as image or crosstab and paste into:

- Excel o Word o Email o PowerPoint

5. Export to PowerPoint

- In newer versions of Tableau, there's direct export to PowerPoint.
- Dashboards are added as images in the presentation.

Steps:

- File > Export as PowerPoint
- Each dashboard becomes one slide.

87.3 Saving Tableau Files

File Type	Description
.twb (Tableau Workbook)	Contains visualization definitions but not the data itself
.twbx (Packaged Workbook)	Includes workbook + data + images. Best for sharing
.hyper (Extract File)	Extracted data for performance
.tds (Data Source)	Reusable data source definition
.tdsx (Packaged Data Source)	Data source + data in one file

For sharing with others offline, use .twbx format.

87.4 Publishing Reports Online

You can publish your Tableau work to one of three main destinations:

1. Tableau Public (Free Platform)

- Best for public sharing
- Anyone on the internet can view your dashboard
- No data privacy control

How to Publish:

- Go to Server > Tableau Public > Save to Tableau Public
- Requires a free Tableau Public account

Use Cases:

- Portfolios
- Blogs
- Classroom examples

2. Tableau Server

- On-premise solution for organizations
- Dashboards are hosted internally Provides:
  - User roles and permissions
  - Scheduled refreshes
  - Data governance

Publishing Steps:

- Server > Publish Workbook
- Choose server URL, project folder, and permissions

Data connections can also be published and scheduled for refresh.

3. Tableau Cloud (formerly Tableau Online)

- Hosted by Tableau (SaaS)
- Ideal for companies without in-house servers
- Provides all the features of Tableau Server but fully cloud-managed

Advantages:

- No infrastructure setup
- Easy to access anywhere
- Secure sharing

Publishing Process:

- Server > Tableau Cloud > Publish Workbook
- Login required
- Set authentication, project, schedule

87.5 Setting Permissions and Access Controls

When publishing to Tableau Server or Cloud, you can control who can see or interact with the dashboard.

Permission Type	Description
Viewer	Can only view and interact
Editor	Can edit workbooks
Owner	Full control, including permissions
None	No access at all

Permissions can be set at workbook, project, or user group levels.

87.6 Scheduling Data Refreshes

If you publish a workbook with live or extract data connections, you can schedule automatic refreshes.

Option	Description

Live Connection	Data updates automatically from source
Extract Refresh	Data is refreshed at set intervals (daily, weekly, hourly)

You can:

- Choose refresh times
- Set email alerts on failure
- Run refresh flows created in Tableau Prep

87.7 Embedding Dashboards

Dashboards published to Tableau Cloud/Server can be embedded into:

- Company intranets
- Websites
- SharePoint portals
- Apps

Embedding uses HTML iframe code, with optional single sign-on (SSO) support.

87.8 Exporting Insights via Subscriptions

Users can subscribe themselves or others to dashboards.

- Sends dashboard snapshots via email (PDF or Image)
- Can schedule daily, weekly, or on refresh
- Supports adding custom messages

Setup:

- Click Subscribe button
- Choose delivery format and schedule

87.9 Summary

Action	Description
Export Image	Static image of chart or dashboard
Export PDF	Printable view
Export Data	Underlying/summarized data
Export PowerPoint	Dashboard snapshots for presentation
Tableau Public	Free public hosting
Tableau Server	Enterprise-level private hosting

Tableau Cloud	**Fully hosted cloud service**
Permissions	**Control access levels for users**
Schedule Refresh	**Automate data updates**
Subscribe Users	**Email dashboards to users regularly**

87.10 Best Practices

- **Use Packaged Workbooks (.twbx) when sending to colleagues**
- **Always check filters and security before publishing sensitive data**
- **Add descriptions and titles in dashboards for clarity**
- **Avoid hard-coding paths; use relative paths or published data sources**
- **Test permissions after publishing to make sure intended users can access it**

Chapter 88: Final Tableau Project

88.1 Introduction

The Final Tableau Project is the culmination of your learning journey with Tableau. This hands-on project brings together all the core and advanced concepts you've studied — from connecting data sources to creating advanced dashboards — and tests your ability to solve real-world business problems using data visualization and analytics.

This chapter will walk you through the complete lifecycle of a Tableau project: from defining objectives to sharing insights with stakeholders.

88.2 Project Goal

The goal is to analyze a real-world dataset and create a fully interactive dashboard that enables users to:

- **Understand key business metrics**
- **Explore trends and comparisons**
- **Drill down into detailed insights**
- **Interact with data using filters, parameters, and controls**

You will also publish and share the dashboard as a professional report.

88.3 Dataset Selection

You can choose any dataset that represents a business case. Some suggestions:

Dataset Type	Description
Sales Data	Products, revenue, region, profit, quantity
HR Analytics	Employee demographics, performance, attrition
Retail Store	Inventory, orders, sales, customer feedback
Education	Student performance, attendance, courses
Healthcare	Patient visits, departments, costs, diagnostics

Recommendation: Use a multi-sheet Excel or CSV file to simulate real-world scenarios with multiple tables (e.g., Products, Orders, Customers).

88.4 Step-by-Step Project Workflow

Step 1: Define the Business Questions

Before working with data, define what you're trying to solve:

- What are the top-performing products?
- Which regions have declining sales?
- How does seasonality affect profits?
- What are the trends in customer behavior?

Step 2: Connect to Data in Tableau

- Go to Start > Connect to Data
- Choose Excel, CSV, or Database
- Use Joins or Unions if data is in multiple sheets
- Use Data Interpreter for cleaning headers
- Rename columns for readability

Step 3: Clean and Transform Data

Using Data Pane + Data Source Tab:

- Change data types (e.g., Date, String, Number)
- Split columns (e.g., Full Name → First, Last)
- Create calculated fields (e.g., Profit Ratio = Profit / Sales)
- Group values (e.g., Group subcategories)
- Use filters to exclude irrelevant rows

Optional: Use Tableau Prep for deeper cleaning

Step 4: Build Individual Visualizations

Create the following types of charts:

Visualization	Purpose
Bar/Line Chart	Compare sales over time
Pie/Tree Map	Distribution of categories
Map	Sales by region or country
KPI Cards	Summary metrics (Total Sales, Profit Margin)
Scatter Plot	Correlation between variables
Bullet/Box Plots	Targets and distributions
Heat Map	Intensity analysis (e.g., sales by day/hour)

Step 5: Add Interactivity

Make your dashboard user-friendly and dynamic:

- Filters (dropdown, multi-select)
- Parameters (select region, choose metric)
- Sorting and highlight actions

- **Drill Down using hierarchies (e.g., Year > Quarter > Month)**
- **Dashboard Actions (Filter, URL, Highlight)**

Step 6: Design the Dashboard Layout

Use these guidelines:

- **Use Grid or Containers for layout**
- **Place KPI metrics at the top**
- **Charts below, grouped by insight**
- **Use legends, labels, titles, and tooltips**
- **Use buttons and navigation (like bookmarks)**
- **Add color-coded indicators (green for increase, red for decrease)**

Step 7: Add Story or Presentation

If desired, use Story Points to present your analysis:

- **Create a narrative (slide-based flow)**
- **Highlight key findings**
- **Show comparisons or before/after analysis**

Step 8: Review and Test

Before sharing:

- **Test all filters and actions**
- **Check mobile/responsive view**
- **Ensure consistency in color palette and font**
- **Add titles, captions, and source notes**

Step 9: Publish and Share Options:

Method	Description
Tableau Public	Free public access
Tableau Server	Enterprise sharing
Tableau Cloud	Hosted Tableau sharing
Export	Image, PDF, PowerPoint

Also consider:

- **Scheduling refreshes**
- **Setting permissions**
- **Embedding dashboards**

88.5 Sample Project Brief

Let9s take an example to make this process more concrete:

Dataset: Superstore Sales

Objective:

- **Identify top and bottom-performing categories**
- **Analyze regional performance**
- **Provide monthly and yearly sales trends**
- **Enable regional managers to filter by category**

Dashboards to Include:

1. **Executive Summary o Total Sales, Profit, Quantity o KPI Cards**
2. **Sales Trends o Line charts (by month/year) o Profit trends with dual axis**
3. **Category Performance o Tree Map of Sales by Subcategory**
   o Bar chart of profit margin
4. **Geo Dashboard o Filled map of sales by state o Region-based filters**
5. **Interactivity o Filters: Region, Segment, Category o Parameters to switch between metrics (Sales vs Profit) o Drill Down from Year > Quarter > Month**

88.6 Evaluation Criteria

Criteria	Weight
Clarity of Visuals	25%
Accuracy of Data	20%
Interactivity & UX	20%
Business Insights	25%
Design & Presentation	10%

88.7 Final Tips

- **Choose a consistent color scheme**
- **Always provide context (titles, legends, axis labels)**
- **Highlight actionable insights, not just data**
- **Optimize for performance (reduce calculations, limit data size)**
- **Ensure your dashboard answers the key business questions**

88.8 Summary

The Final Tableau Project is your opportunity to combine all skills — from data prep to visual storytelling. It's a practical demonstration of your ability to solve problems using Tableau.

A well-designed dashboard should not only present data — it should drive decisions.