FUNDAMENTALS OF WORKBOOK1.1.1

~Concept of workbook

A workbook is a file that contains one or more worksheets to help you organize data. You can create a new workbook from a blank workbook or a template.

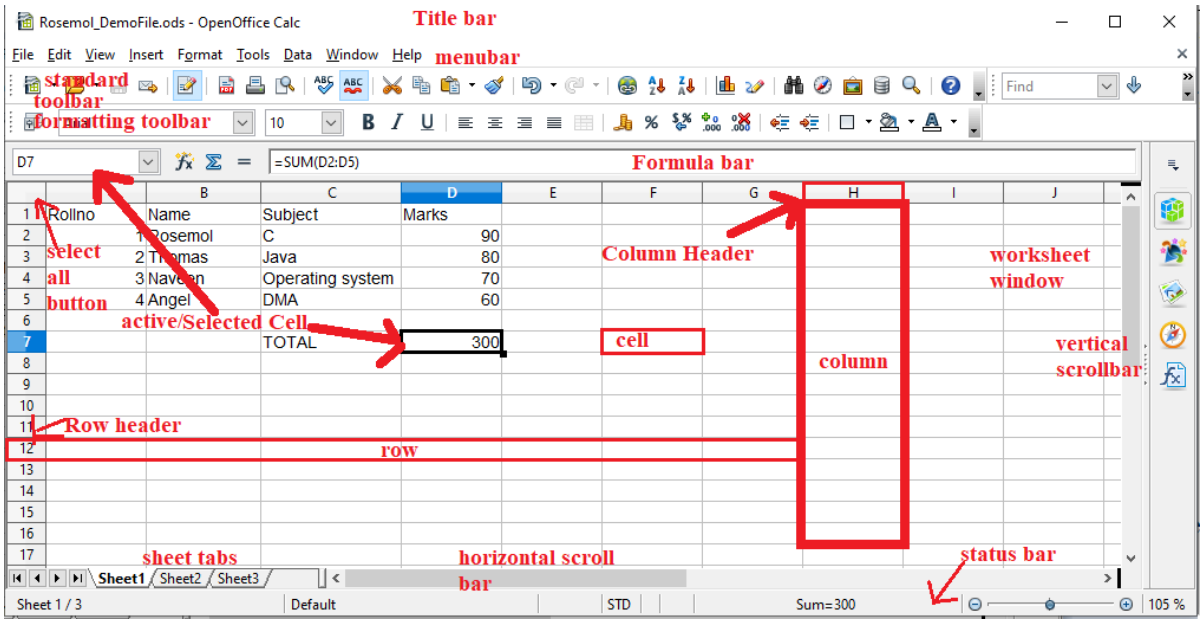
A spreadsheet or worksheet is a file made of rows and columns that help sort data, arrange data easily, and calculate numerical data. A spreadsheet software program has ability to calculate values using mathematical formulas and the data in cells.

A spreadsheet is a grid of boxes. Each box is called a cell. Each cell is located in a particular row and column in the grid. Each cell is available to store data.

The spreadsheet can contain 32000 rows and 255 columns.

More than one cell is called a range of cells. You can select a range of cells and move them, copy them, format them and so on as easily as a single cell.

Spreadsheets are used to store a table or group of tables .A table is a range of cells with related data. To store tables that are related you can stack the tables into sheets.



~Apache OpenOffice

Apache OpenOffice is the leading open-source office software suite for word processing, spreadsheets, presentations, graphics, databases and more. It is published by the non-profit Apache Software Foundation. OpenOffice works on all common computers. OpenOffice can work with ODF documents as well as documents from other common office software packages. It can be downloaded and used completely free of charge for any purpose.

~What are ODF documents?

ODF is an ISO International Standard format for office documents, created in 2006. ODF files have the following file extensions:

- *.odt (word processor documents) OpenDocument Text
- *.ods (spreadsheet documents) OpenDocument spreadsheet
- *.odp (presentation documents) OpenDocument presentation

The advantage of ODF is that it is not tied to any one office application suite. It is an open standard that any company can implement in their software. OpenOffice uses ODF format as its default document format. Most other word processors, of recent vintage, also have the ability to import and export ODF

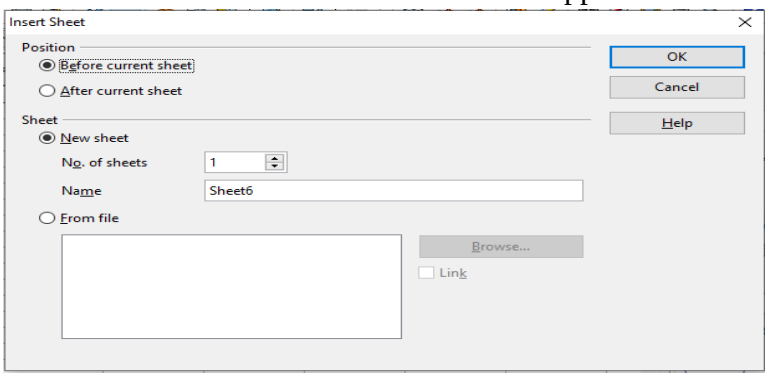
~What is Calc?

Calc is the spreadsheet component of OpenOffice.org (OOo). You can enter data (usually numerical) in a spreadsheet and then manipulate this data to produce certain results.

~Adding worksheet

Calc provides three sheets in every new spreadsheet document.

Following are the steps to add a sheet :
1. Click a sheet name that is located either before or after where you want to add new sheets.
2. Choose Insert->sheet to open insert sheet dialog box.
OR
1. Right click the sheet name
2. Choose insert from the shortcut menu that appears



3. Mark the box to position the new sheet either before or after the active sheet.
4. Indicate how many sheets you want to add.
5. Provide the name of the sheet.
6. Click on OK.

~Cell Address

A cell reference, or cell address, is an alphanumeric value used to identify a specific cell in a spreadsheet. Each cell reference contains one or more letters followed

by a number. The letter or letters identify the column and the number represents the row.

In a standard spreadsheet, the first column is A, the second column is B, the third column is C, etc. These letters are typically displayed in the column headers at the top of the spreadsheet. If there are more than 26 columns, the 26th column is labelled Z, followed by AA for column 27, AB for column 28, AC for column 29, etc. Column 55 is labelled BA. Rows simply increment numerically from top to bottom starting with "1" for the first row.

Examples of cell references are listed below:

1. First column, sixth row: A6
2. sixth column, twentieth row: G20
3. Sixty-first column, three hundred forty-second row: BI342
4. One thousand column, two thousandth row: ALL2000

Cell references are helpful in two ways:

1) They provide an easy way to locate a specific value within a spreadsheet

2) They are used in creating formulas.

~ Formula Bar

The Formula Bar is where data or formulas you enter into a worksheet appear for the active cell. The Formula Bar can also be used to edit data or formula in the active cell. The active cell displays the results of its formula while we see the formula itself in the Formula Bar.You can also hide the Formula Bar entirely by going to the View menu and uncheck the Formula Bar option.

~ Column

A column is a vertical series of cells in a spreadsheet. Columns run vertically downward across the worksheet. A column is identified by a column header that is on the top of the column, from where the column originates. Column headers (column letter) are named as A, B, C, D, E, F, G, and H.

~ Rows

Rows run horizontally across the worksheet. A row is identified by the number that is on left side of the row, from where the row originates.

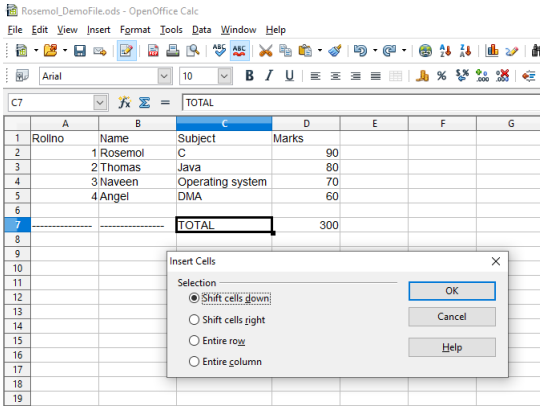
~ Cells

Cells are small boxes in the worksheet where we enter data. A cell is the intersection of a row and column. It is identified by row number and column header such as A1, A2.

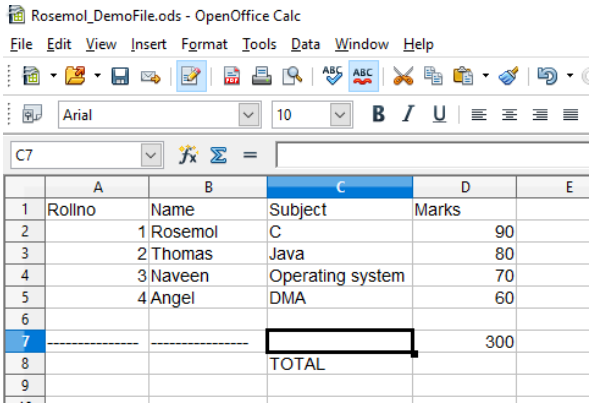
~ Insert Cell

1. Select a cell

2. Choose Insert->Cells… or right click on cell and select insert… from context menu or pop up menu.
3. Insert Cells dialog box appears as shown in the figure

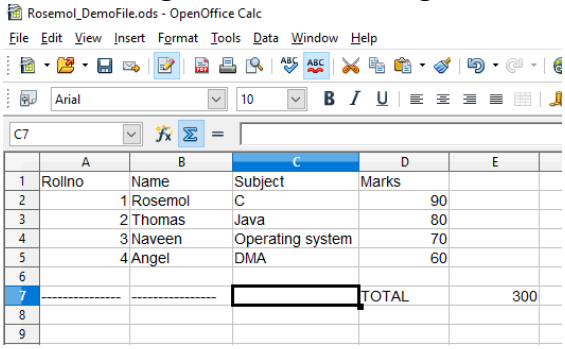


4. Shift cells down will move cell down



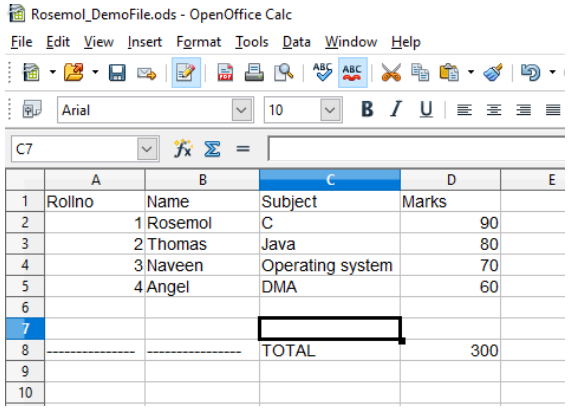
	A	B	C	D	E
1	Rollno	Name	Subject	Marks	
2	1	Rosemol	C	90	
3	2	Thomas	Java	80	
4	3	Naveen	Operating system	70	
5	4	Angel	DMA	60	
6					
7	----------	----------		300	
8			TOTAL		
9					

4. Shift cells right will move cell right



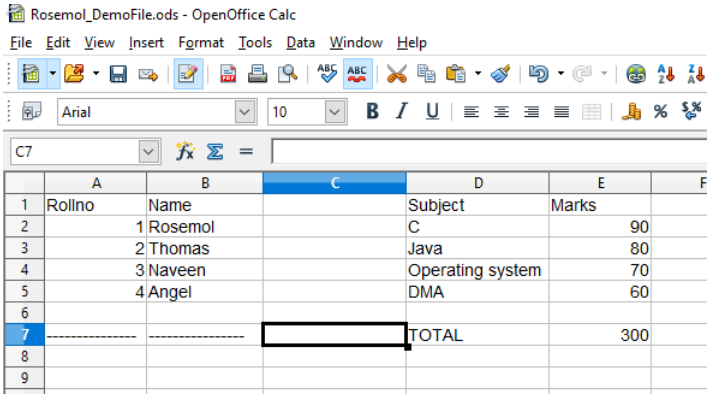
	A	B	C	D	E
1	Rollno	Name	Subject	Marks	
2	1	Rosemol	C	90	
3	2	Thomas	Java	80	
4	3	Naveen	Operating system	70	
5	4	Angel	DMA	60	
6					
7	----------	----------		TOTAL	300
8					
9					

5. Entire row will add a new row



	A	B	C	D	E
1	Rollno	Name	Subject	Marks	
2	1	Rosemol	C	90	
3	2	Thomas	Java	80	
4	3	Naveen	Operating system	70	
5	4	Angel	DMA	60	
6					
7					
8	----------	----------	TOTAL	300	
9					
10					

6. Entire column will add a new column

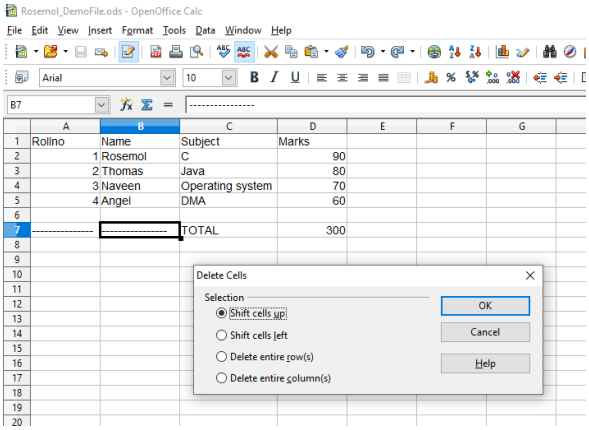


	A	B	C	D	E	F
1	Rollno	Name		Subject	Marks	
2		1	Rosemol	C	90	
3		2	Thomas	Java	80	
4		3	Naveen	Operating system	70	
5		4	Angel	DMA	60	
6						
7	----------	----------		TOTAL	300	
8						
9						

7. Click OK

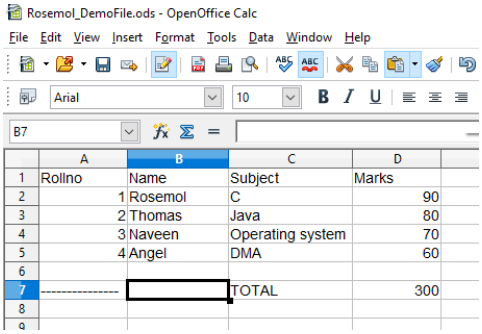
~ Delete Cell

1. Select at least one cell you want to delete.
2. Choose Edit->Delete Cells… to open the Delete Cells dialog box.
3. Delete Cells dialog box appears as shown in the figure.



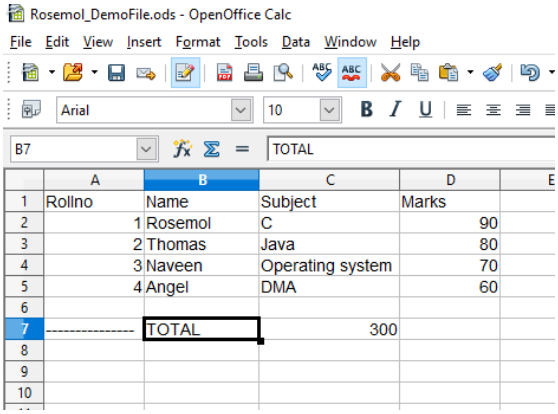
	A	B	C	D	E	F	G
1	Rollno	Name	Subject	Marks			
2		1	Rosemol	C	90		
3		2	Thomas	Java	80		
4		3	Naveen	Operating system	70		
5		4	Angel	DMA	60		
6							
7	----------	----------	TOTAL	300			

4. Shift cells up will move cells up



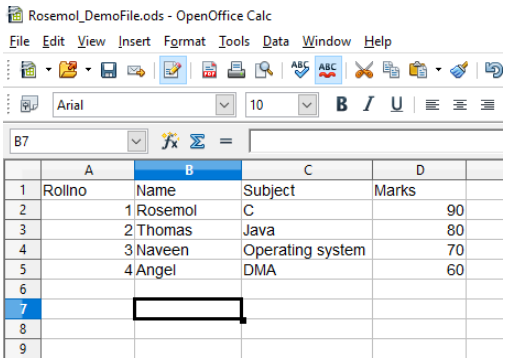
	A	B	C	D
1	Rollno	Name	Subject	Marks
2	1	Rosemol	C	90
3	2	Thomas	Java	80
4	3	Naveen	Operating system	70
5	4	Angel	DMA	60
6				
7	----------		TOTAL	300
8				

5. Shift cells left will move cells left



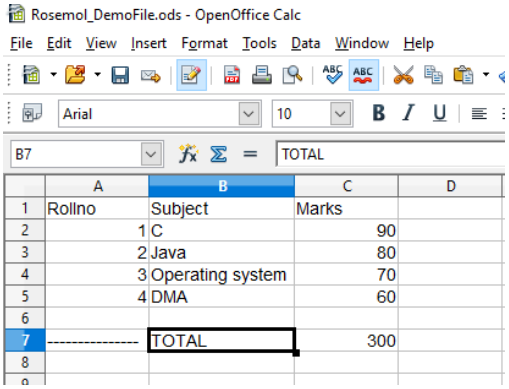
	A	B	C	D
1	Rollno	Name	Subject	Marks
2	1	Rosemol	C	90
3	2	Thomas	Java	80
4	3	Naveen	Operating system	70
5	4	Angel	DMA	60
6				
7	----------	TOTAL	300	
8				
9				
10				

6. Delete entire row will remove the entire row



	A	B	C	D
1	Rollno	Name	Subject	Marks
2	1	Rosemol	C	90
3	2	Thomas	Java	80
4	3	Naveen	Operating system	70
5	4	Angel	DMA	60
6				
7				
8				
9				

7. Delete entire column will remove the entire column



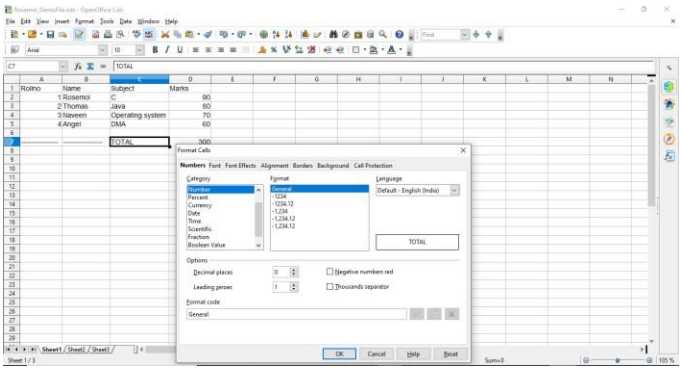
	A	B	C	D
1	Rollno	Subject	Marks	
2	1	C	90	
3	2	Java	80	
4	3	Operating system	70	
5	4	DMA	60	
6				
7	----------	TOTAL	300	
8				

8. Click OK.

Calc then renames all the columns or rows as though the deleted rows never existed.

~ Format cells

1. Select the cell or range of cells that you want to format.
2. Choose Format->Cells or right-click anywhere in the selected area, and choose Format Cells from the shortcut menu that appears.



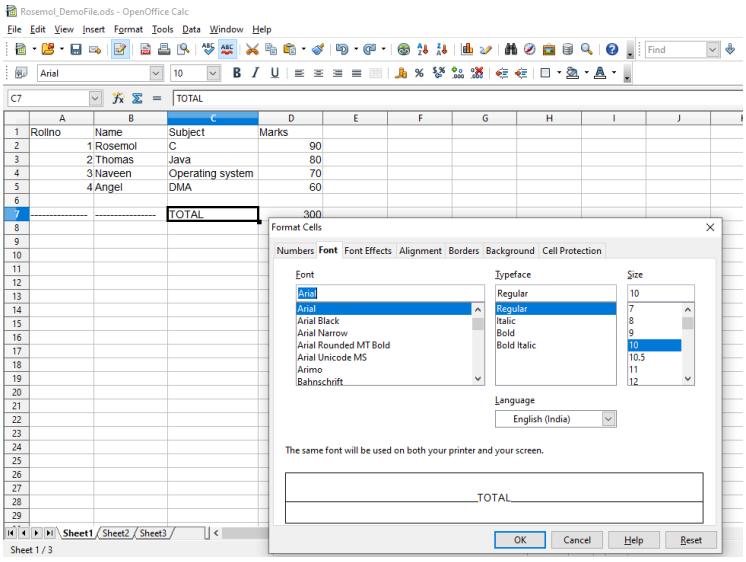
It has the following list of tabs to format the cell

Number

Number can be formatted as per our need like percent, currency, date, time. Decimal places can also be set. Leading zeros can be displayed

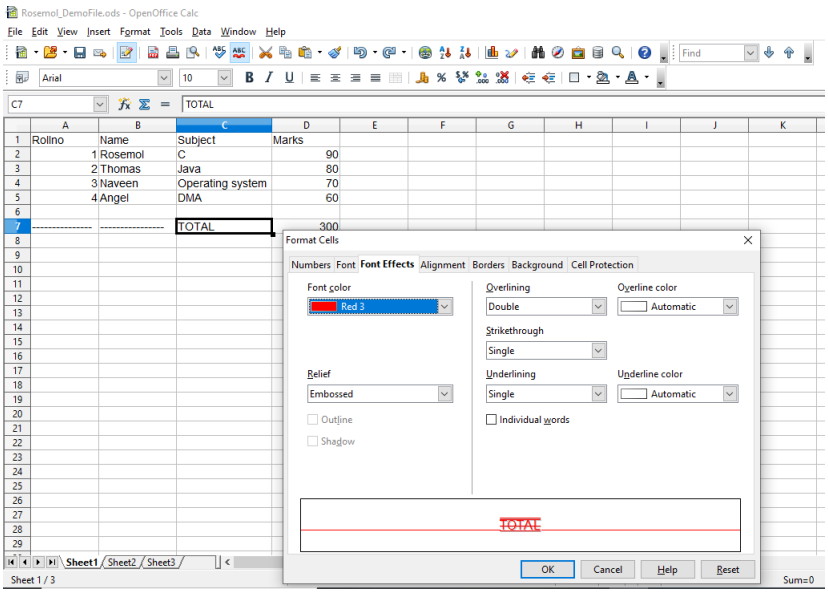
Font

Font tab allows to select the style of the font, font size. We can set font to be bold Italic as per our wish.

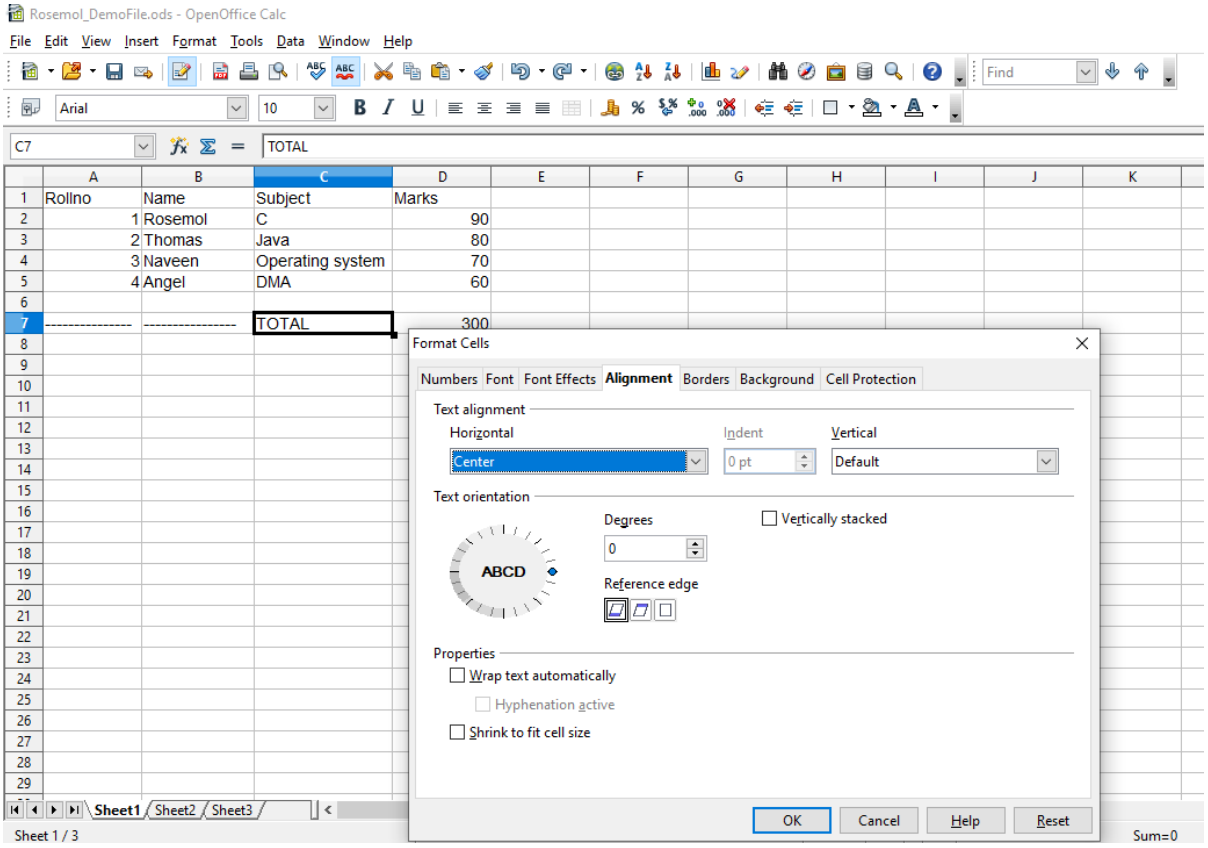


Font effects

Font colour, underline ,underline colour, shadow are various effects that can be given to font

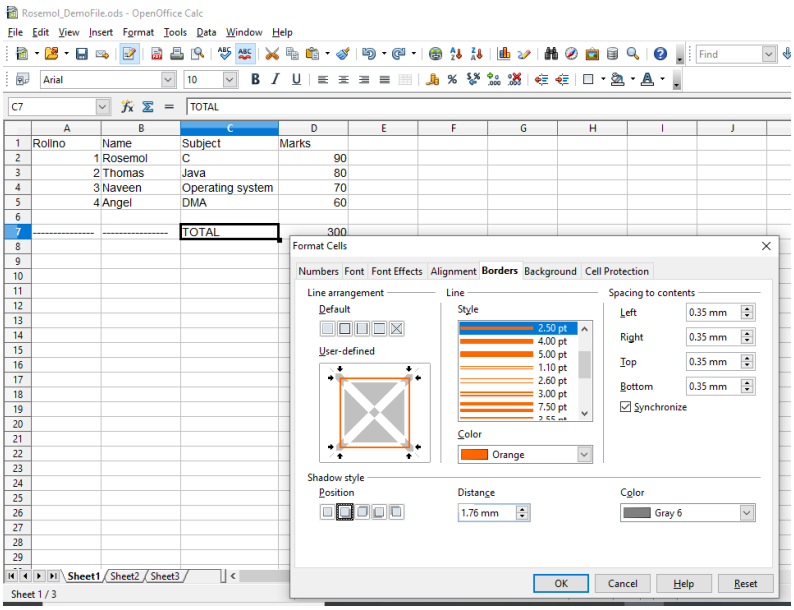


Alignment



Borders

Various border styles can be given to cell



Rollno	Name	Subject	Marks
1	Rosemol	C	90
2	Thomas	Java	80
3	Naveen	Operating system	70
4	Angel	DMA	60
----------	----------	TOTAL	300

Background

Background colour can be given to the selected cell

~Cell size: Changing column widths and row heights

To change column widths and row heights using the mouse, perform the following steps:

1. Click the line that separates two column names or row names. Choose the line that is to the right of the column that you want to resize (or below the row that you want to resize).

For example, the line between columns D and E resizes column D, and the line between rows 3 and 4 resizes row 3. The mouse pointer changes to a double-headed arrow.

2. While holding down the mouse button, drag the line to the desired column width or row height.

Double-click the line separating the two column names or row names that is to the right of the column that you want to resize (or below the row that you want to resize), Calc

optimizes the column width or row height according to the data that's currently in that column or row.

To change column widths and row heights using the main menu, follow these steps:
1. Click a cell in the column or row that you want to resize.
2. Choose Format->Column->Width to open the Column Width dialog box.
Or choose Format->Row->Height to open the Row Height dialog box.
3. Enter the desired size of your column or row.
4. Click OK.

~Renaming sheets

1. Select the sheet that you want to rename.
2. Choose Format->Sheet->Rename.
3. The Rename Sheet dialog box appears.
You can also right-click any sheet name tab and choose Rename from the shortcut menu that appears.
4. In the Rename Sheet dialog box, type in the new name
5. Click OK.

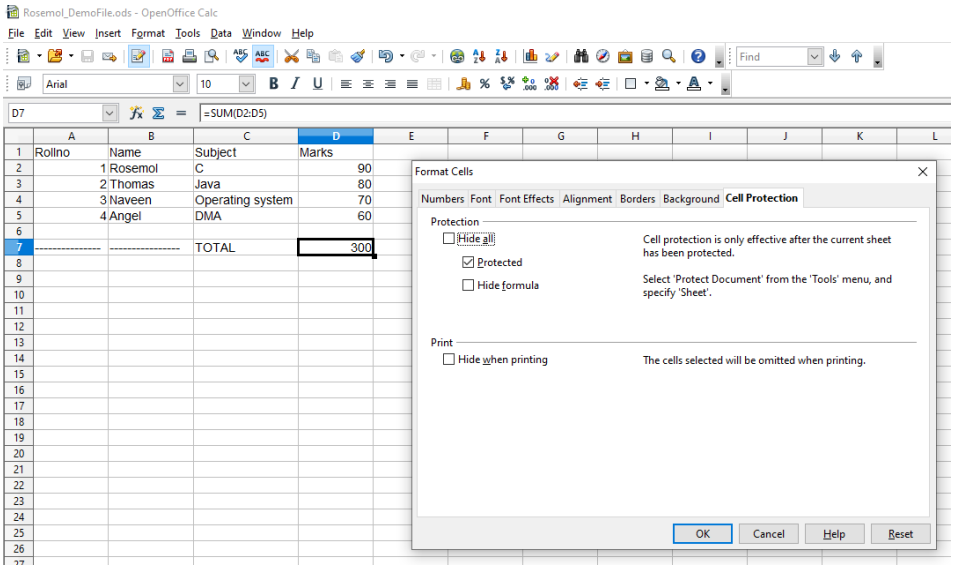
~Protection of a Sheet

To write-protect all of the cells of a sheet, you have to do the following:
1. Select Tools->Protect Document from the Menu Bar,
if you choose Sheet..., only your current sheet will be protected from writing,
If you choose Document..., your whole document (workbook) will be protected.
2. If you do not want to enter a password, you can simply click on the OK button to close the dialog window without typing anything.
However, if you choose a password, it will be requested every time you wish to modify the cells or the sheet.

To remove the protection
1. Choose Tools->Protect Document
2. Deselect the Sheet... or Document... option

~Lock cells

1. Select the cells that you want to specify the cell protection options for.
2. Choose Format -> Cells and click the Cell Protection tab.
3. Select the protection options that you want.
All options will be applied only after you protect the sheet from the Tools menu
-Uncheck Protected to allow the user to change the currently selected cells.
-Select Protected to prevent changes to the contents and the format of a cell.
-Select Hide formula to hide and to protect formulas from changes.
-Select Hide when printing to hide protected cells in the printed document. The cells are not hidden onscreen.
4. Click OK.


Rosemol_DemoFile.ods - OpenOffice Calc
File Edit View Insert Format Tools Data Window Help
Arial
10
D7
=SUM(D2:D5)
Rollno Name Subject Marks
1 Rosemol C 90
2 Thomas Java 80
3 Naveen Operating system 70
4 Angel DMA 60
TOTAL 300
Format Cells
Numbers Font Font Effects Alignment Borders Background Cell Protection
Protection
Hide all
Protected
Hide formula
Cell protection is only effective after the current sheet has been protected.
Select 'Protect Document' from the 'Tools' menu, and specify 'Sheet'.
Print
Hide when printing
The cells selected will be omitted when printing.
OK Cancel Help Reset

1.1.2

Copying, Pasting, Cutting, Dragging, and Dropping Your Cells

Now that you have your cells selected, it's child's play to cut and paste them, drag and drop them, and move them all around.

Copying and pasting cells

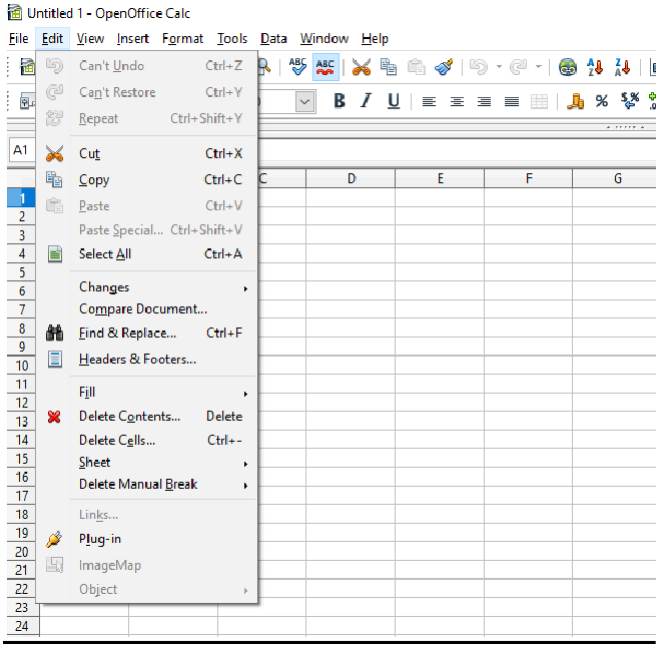
To copy and paste a range of cells, follow these steps:

1. Select the range of cells that you want to copy.
2. If the Function toolbar is not visible, choose View➪Toolbars➪ Function Bar to make it appear.
3. Choose Edit➪Copy, press Ctrl+C, or click the Copy icon on the Function toolbar.
4. Click the cell that you want to paste the upper-left corner of your range into.
5. Choose Edit➪Paste, press Ctrl+V, or click the Paste icon on the Function toolbar.

Cutting and pasting cells

To cut and paste a range of cells, follow these steps:

1. Select the range of cells that you want to copy.
2. Choose Edit➪Cut, press Ctrl+X, or click the Cut icon on the Function toolbar. (If the Function toolbar is not visible, choose View➪Toolbars➪ Function Bar to make it appear.)
3. Click the cell that you want to paste the upper-left corner of your range into.
4. Choose Edit➪Paste, press Ctrl+V, or click the Paste icon on the Function toolbar.



Dragging and dropping cells

You may find this is the most enjoyable way to cut and paste. Once you know how to drag and drop, you may want to do nothing else.

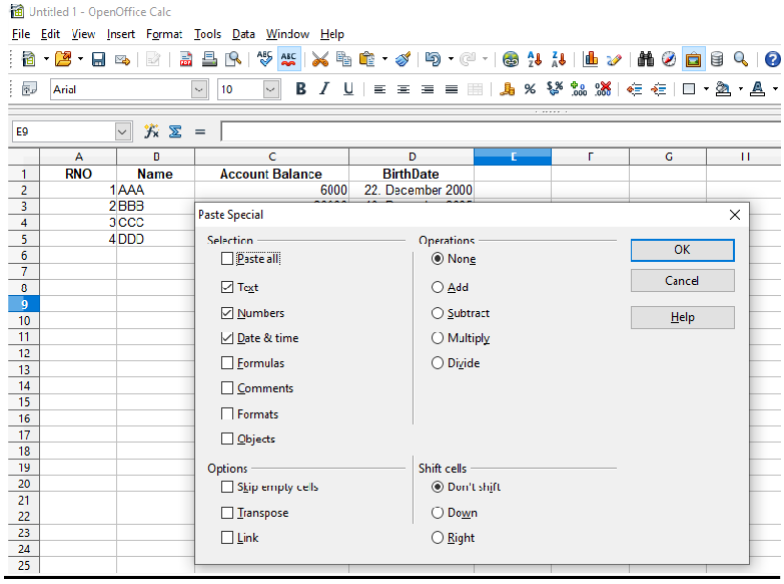
1. Click the box in the status bar until the box displays STD. Dragging and dropping can only be performed with single cells or standard rectangular ranges.
2. Click and drag the active cell to create a selected range.

3. Click anywhere in the selected area, and drag the entire range to a new location.

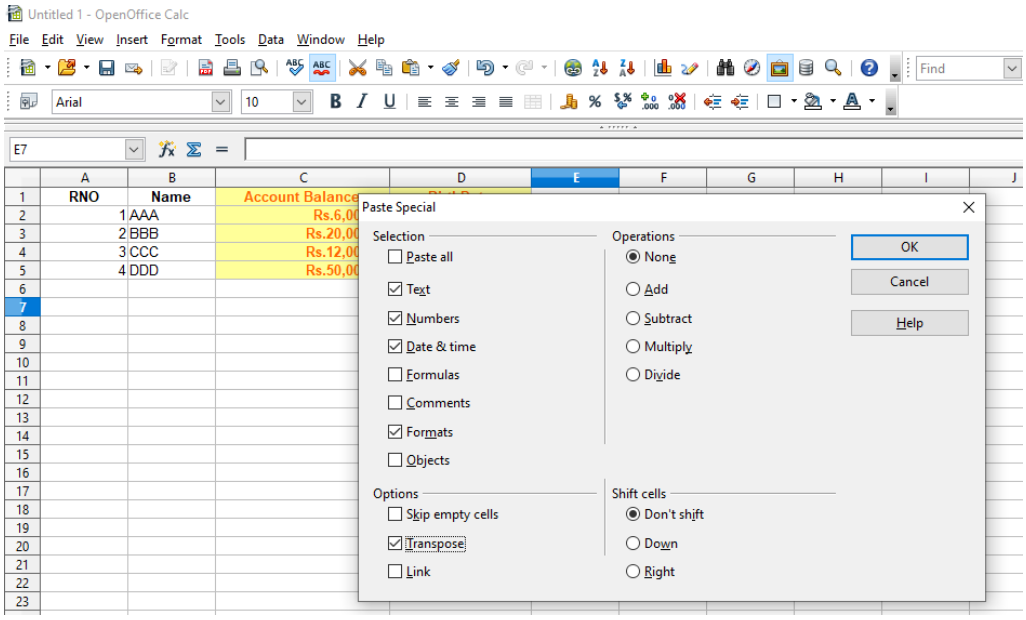
Paste Special

To copy and paste special a range of cells, follow these steps:

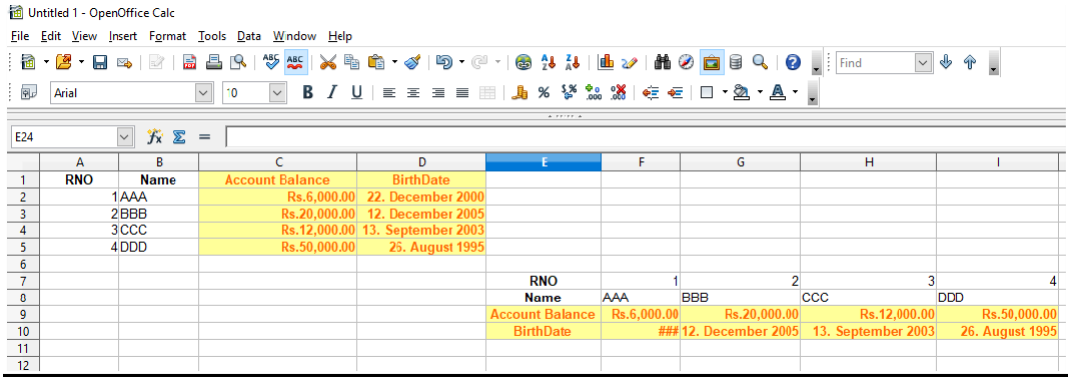
1. Select the range of cells that you want to copy.
2. Choose Edit➪Copy, press Ctrl+C, or click the Copy icon on the Function toolbar.
3. Click the cell that you want to paste the upper-left corner of your range into.
4. Choose Edit➪Paste Special or press Ctrl+Shift+V. Then dialog box appears as shown in the figure.



5. Check the ‘Selection’ you want like Text, Numbers, Formulas, Formats etc.
6. Check the ‘Options’ like Transpose, Skip empty cells etc.
7. Select the ‘Operations’ like Add, Subtract, Multiply etc.
8. Click OK.



Checked ‘Selection-Formats’ and ‘Options-Transpose’ will transpose the table with formatting.

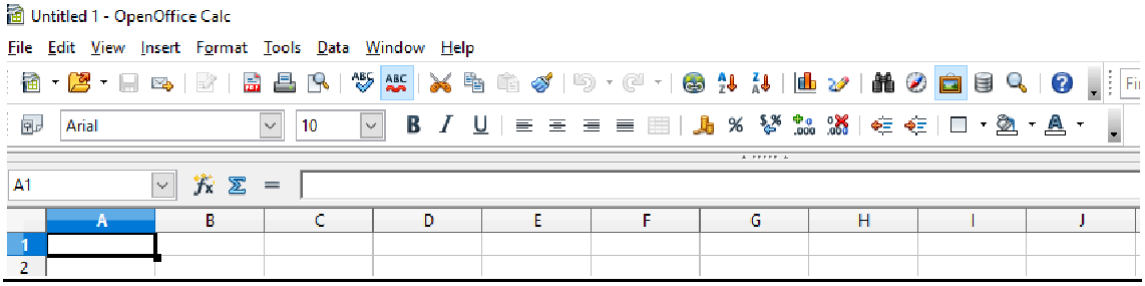


	A	B	C	D	E	F	G	H	I
1	RNO	Name	Account Balance	BirthDate					
2	1	AAA	Rs.6,000.00	22. December 2000					
3	2	BBB	Rs.20,000.00	12. December 2005					
4	3	CCC	Rs.12,000.00	13. September 2003					
5	4	DDD	Rs.50,000.00	25. August 1995					
6									
7					RNO	1	2	3	4
0					Name	AAA	BBB	CCC	DDD
9					Account Balance	Rs.6,000.00	Rs.20,000.00	Rs.12,000.00	Rs.50,000.00
10					BirthDate	###	12. December 2005	13. September 2003	26. August 1995
11									
12									

Adding style with the toobar

Want to change your font type, size, and style right from the toolbar? Calc makes it easy to format text. To choose your font type, size, and style, follow these steps:

1. Click the cell that you want to format. Or, select a range of cells — you can do this by clicking the active cell and dragging. The selected cells appear shaded.
2. Select your font from the list of fonts in the combo box on the Object toolbar. If the fonts combo box is not visible, choose View➪Toolbars➪ Object bar to make it appear.
3. Select the size of your type from the combo box on the Object toolbar. Calc resizes the rows to accommodate the larger or smaller sizes of type.
4. Choose your style. Choose either boldface, italic, or underline, or choose the color for your type.



Adding background colors

We never work on a spreadsheet without changing the background color of the table that we're working on. Somehow a soft golden glow on the screen is more soothing to the eyes than glaring white. To change the background color of the table, follow these steps:

1. Select the cells of your table.
2. Click the Background Color icon, and select your color from the pop-up swatches.

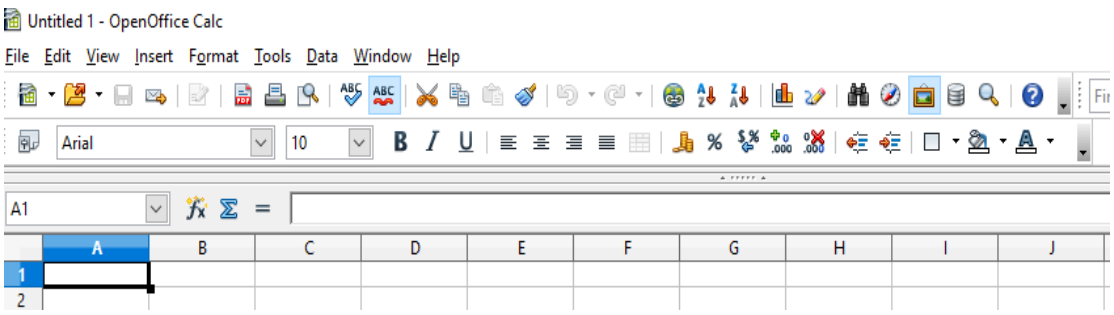
Choosing font colors

It's no secret that black is not our favorite color, so we never display our numbers in black. Of course, we don't want them in red, either, but the world is a rainbow of colors. It's fun to explore. To change your font colors, follow these steps:

1. Select the range of cells to be formatted.
2. Click the Font Color icon, and select your color from the pop-up swatches.

Format PaintBrush

1. Select the cell from which you wish to copy the format from.
2. Click on the Format Paintbrush icon in the toolbar.



3. When mouse look like a paint bucket, select the cell or range of cells to which you want to apply the format.
4. Let the mouse button go and the format will be applied to that cell or range of cells.

1.2

~Alignment

Alignment is how text flows in relation to the rest of the page (or column, table cell, text box, etc.). There are four main alignments: left, right, center, and justified.

- Left-aligned text is text that is aligned with a left edge.
- Right-aligned text is text that is aligned with a right edge.
- Centered text is text that is centered between two edges.
- Justification controls the spacing between words. A justified text increases the space between words to fill the entire line so that it is aligned with both the left and right edges.

1. Select the cell you want to align

2. On the formatting toolbar, select the alignment tool from formatting toolbar

~Indentation

In many documents, indenting is a good way to distinguish the start of a new paragraph, especially when there is no paragraph spacing.

To indent, hit the Tab key once on your keyboard at the start of a paragraph.

1. Select the text you want to indent

2. On the formatting toolbar, select the decrease or increase indent button to give spacing before the text

~Number format

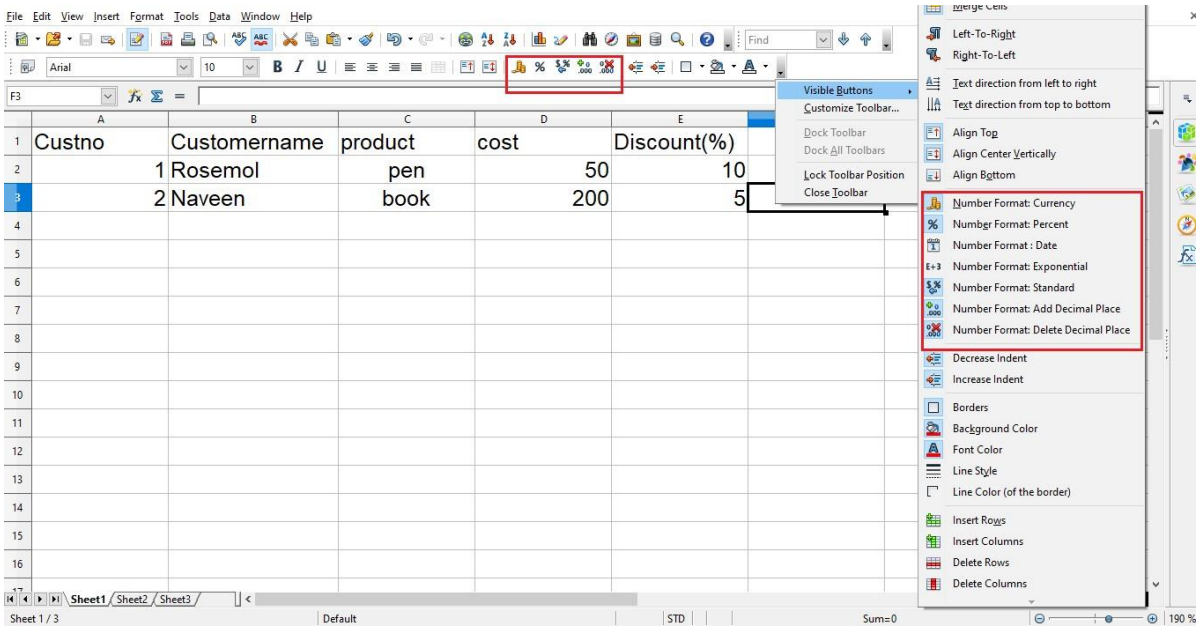
It allows to set the standard number format

~Percent Style

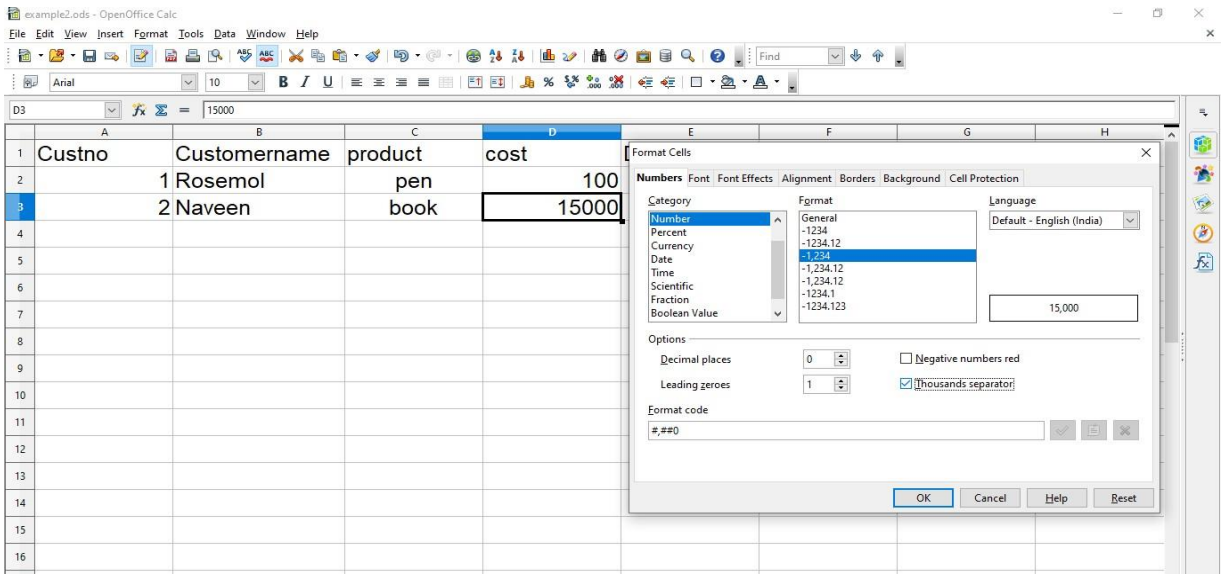
It Converts number into Percent and adds a % symbol

~Increase/Decrease Decimal

It increases or decreases the digits after the decimal point.



~Comma Style



1. Select the cell to give comma style.
2. Chose format>cells to open Format cells dialog box.
3. In the Numbers tab tick mark the ‘Thousands Separator’.
4. Click on OK button.

1.2.1

~Insert Picture

Click in the location in the Calc document where you want the image to appear.

1. Choose Insert > Picture > From File from the menu bar, or click the Insert Picture icon  on the Picture toolbar (f the toolbar is visible).

2. In the Insert Picture dialog, navigate to the file to be inserted, select it, and click Open.

~Insert Shapes

1. Open an OpenOffice Draw document, and look for the "Drawing" toolbar. This toolbar has shapes on it. If you don't see the toolbar, click "View" followed by "Toolbars" and "Drawing."
2. Click one of the toolbar's shapes to turn your cursor into a crosshair. Click inside your document, hold down your left mouse button and drag the cursor down towards the right to draw your shape. Release the mouse button when the shape reaches the size you desire.
3. Click the shape, hold down your left mouse button and drag the shape to the place on the document where you want it to appear. Resize the shape by clicking and dragging one of the handles along the shape's edge.

~Insert TextBox

1. Click on the Text icon  on the Drawing toolbar. If the toolbar with the text icon is not visible, choose View > Toolbars > Drawing.
2. Click and drag to draw a box for the text on the slide. Do not worry about the vertical size and position—the text box will expand if needed as you type.
3. Release the mouse button when finished. The cursor appears in the text box, which is now in edit mode (gray hashed border with green resizing handles).
4. Type or paste your text in the text box.
5. Click outside the text box to deselect it.

You can move, resize, and delete text boxes

~Insert Header & Footer

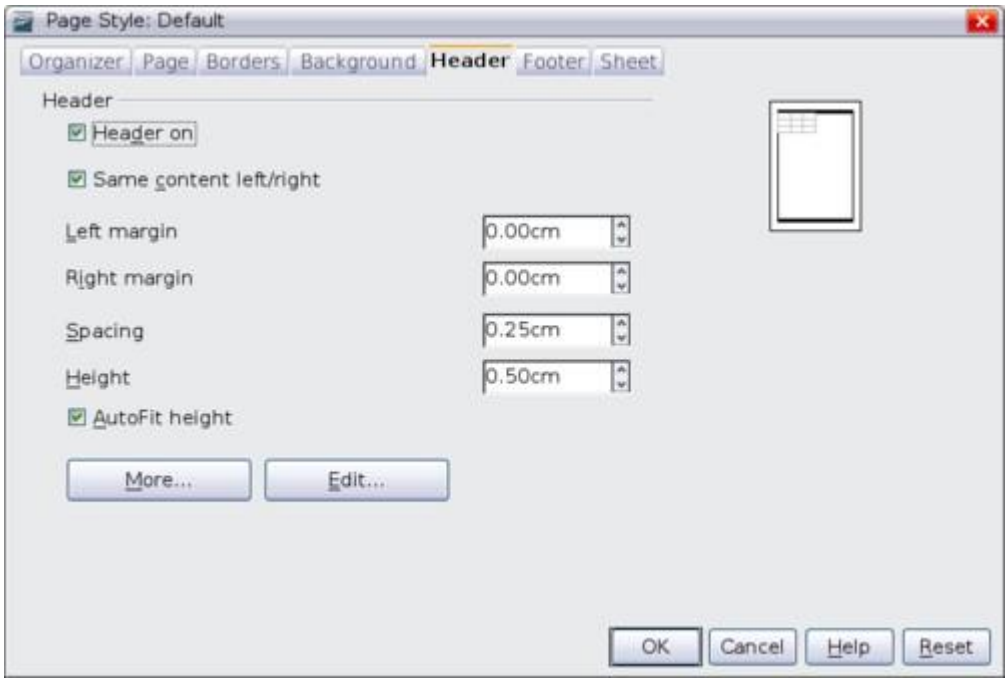
Headers and footers are predefined pieces of text that are printed at the top or bottom of a sheet outside of the sheet area. A header is the top margin of each page, and a footer is the bottom margin of each page. Headers and footers are useful for including material that you want to appear on every page of a document such as your name, the title of the document, or page numbers.

Setting a header or a footer

To set a header or footer:

1. Navigate to the sheet that you want to set the header or footer for. Select Format > Page.
2. Select the Header (or Footer) tab.

3. Select the Header on option.



Header dialog

From here you can also set the margins, the spacing, and height for the header or footer. You can check the AutoFit height box to have the height of the header or footer automatically adjust.

Margin

Changing the size of the left or right margin adjusts how far the header or footer is from the side of the page.

Spacing

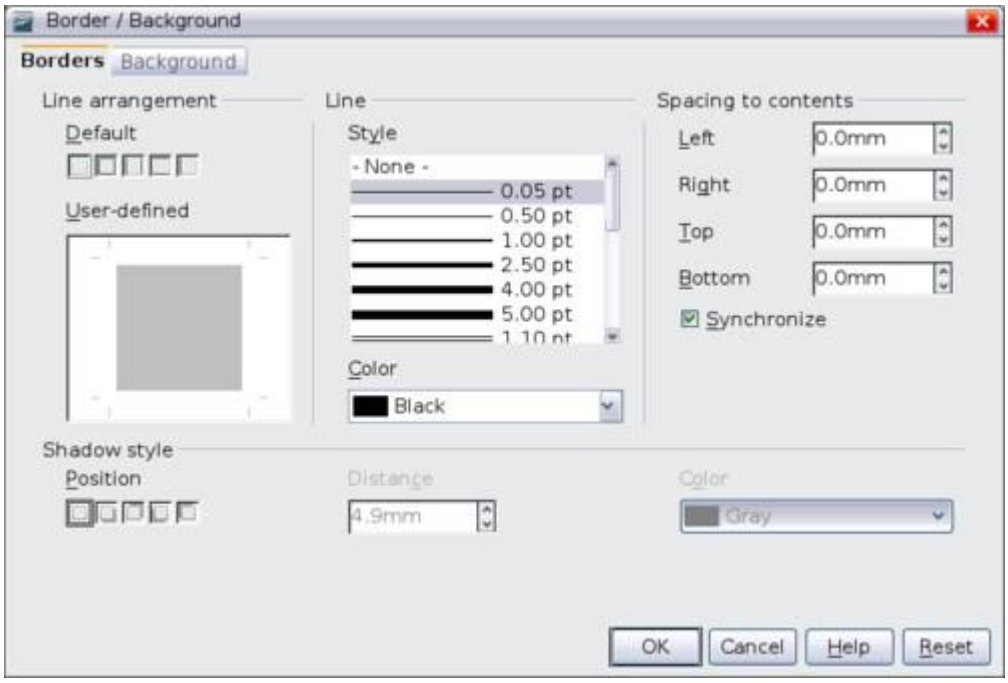
Spacing affects how far above or below the sheet the header or footer will print. So, if spacing is set to 1.00", then there will be 1 inch between the header or footer and the sheet.

Height

Height affects how big the header or footer will be.

Header or footer appearance

To change the appearance of the header or footer, click More.



Header/Footer Border/Background

From this dialog you can set the background and border of the header or footer.

Setting the contents of the header or footer

The header or footer of a Calc spreadsheet has three columns for text. Each column can have different contents.

To set the contents of the header or footer, click the Edit button in the header or footer dialog shown below to display the dialog shown below.

Edit contents of header or footer

Areas

Each area is independent and can have different information in it.

Header

You can select from several preset choices in the Header drop-down list, or specify a custom header using the buttons below. (If you are formatting a footer, the choices are the same.)

Custom header

Click in the area (Left, Center, Right) that you want to customize, then use the buttons to add elements or change text attributes.

 Opens the Text Attributes dialog.

 Inserts the total number of pages.

 Inserts the File Name field.

 Inserts the Date field.

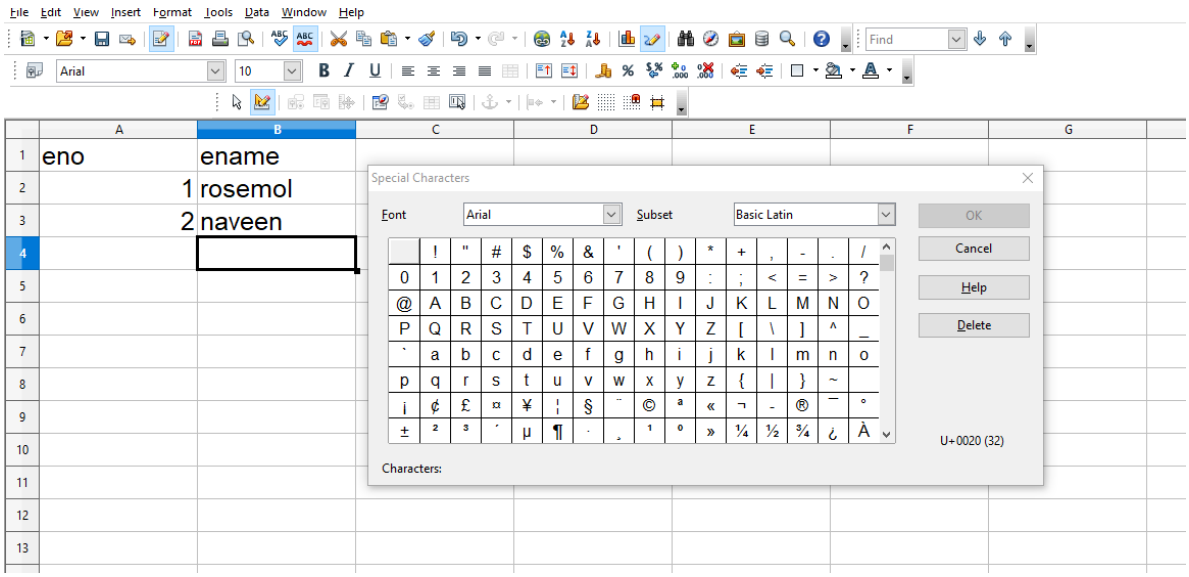
 Inserts the Sheet Name field.

 Inserts the Time field.

 Inserts the current page number.

~Insert Symbols

1. Place the cursor in the location where you want the symbol to appear.
2. On the Insert menu select Special Character.
3. In the dialog that appears, all the available characters in the current fonts will be displayed.
4. Select a character by clicking on it. If you do not see the desired character, it may not be available in the current font. If it is not there, try changing fonts. Choose Symbol from the Font list.
5. After selecting one or more characters, click OK to insert the characters at the location of the text cursor.


File Edit View Insert Format Tools Data Window Help
Arial
10
Find
eno
ename
1 rosemol
2 naveen
Special Characters
Font
Arial
Subset
Basic Latin
OK
Cancel
Help
Delete
U+0020 (32)
Characters:

1.2.3

Saving a spreadsheets

Spreadsheets can be saved in the following three ways.

From the menu bar

Click **File** and then select **Save** (or **Save All** or **Save As**).

From the toolbar

Click on the **Save** button  on the Function bar. If the file has been saved and no subsequent changes have been made, this button is grayed-out and unselectable.

From the keyboard

Use the key combination *Control+S*.

If the spreadsheet has been previously saved, then saving will overwrite the existing copy without opening the Save As dialog. If you want to save the spreadsheet in a different location or with a different name, then select **File > Save As**.

Password protection

To protect an entire document from being viewable without a password, use the option on the Save As dialog box to enter a password.

On the Save As dialog box, select the **Save with password** option, and then click **Save.** You will receive a prompt to type the same password in two fields. Click OK to save the document password-protected. If the passwords do not match, you receive the prompt to enter the password again.

Saving a document automatically

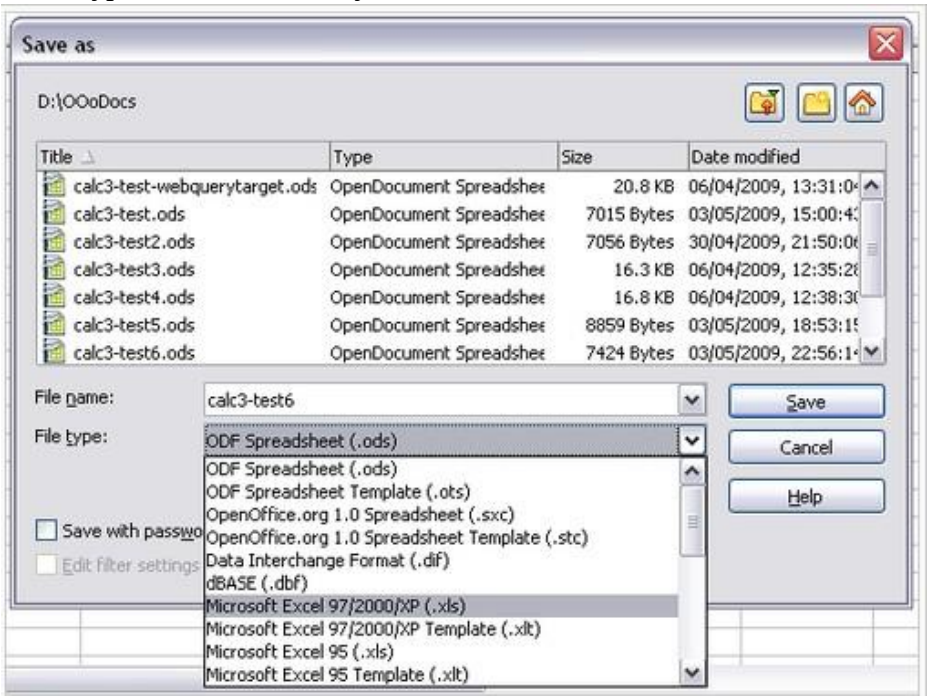
You can choose to have Calc save your spreadsheet automatically at regular intervals. Automatic saving, like manual saving, overwrites the last saved state of the file. To set up automatic file saving:

1. Select **Tools > Options > Load/Save > General**.
2. Click on **Save AutoRecovery information every**. This enables the box to set the interval. The default value is 30 minutes. Enter the value you want by typing it or by pressing the up or down arrow keys.

Saving as a Microsoft Excel document

Some users of Microsoft Excel may be unwilling or unable to receive *.ods files In this case, you can save a document as a Excel file.

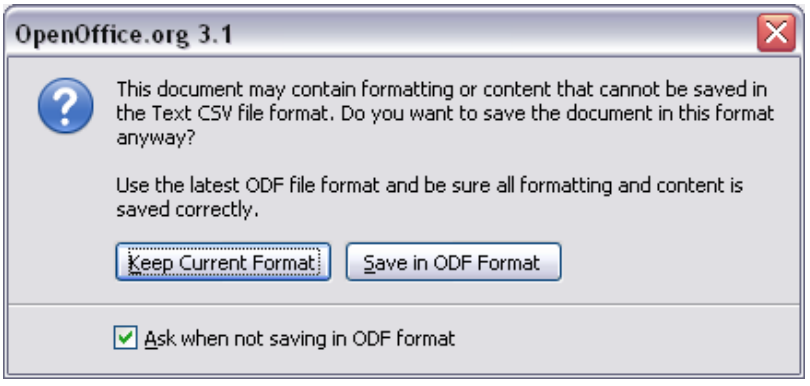
1. **IMPORTANT**—First save your spreadsheet in the file format used by OpenOffice.org, *.ods. If you do not, any changes you made since the last time you saved will only appear in the Microsoft Excel version of the document.
2. Then click **File > Save As**.
3. On the Save As dialog, in the **File type** (or **Save as type**) drop-down menu, select the type of Excel format you need. Click **Save**.



Saving as a CSV file

To save a spreadsheet as a comma separated value (CSV) file:

1. Choose **File > Save As**.
2. In the File name box, type a name for the file.
3. In the File type list, select **Text CSV** (comma-separated values **file)**, and click **Save**.You may see the message box shown below. Click **Keep Current Format**.



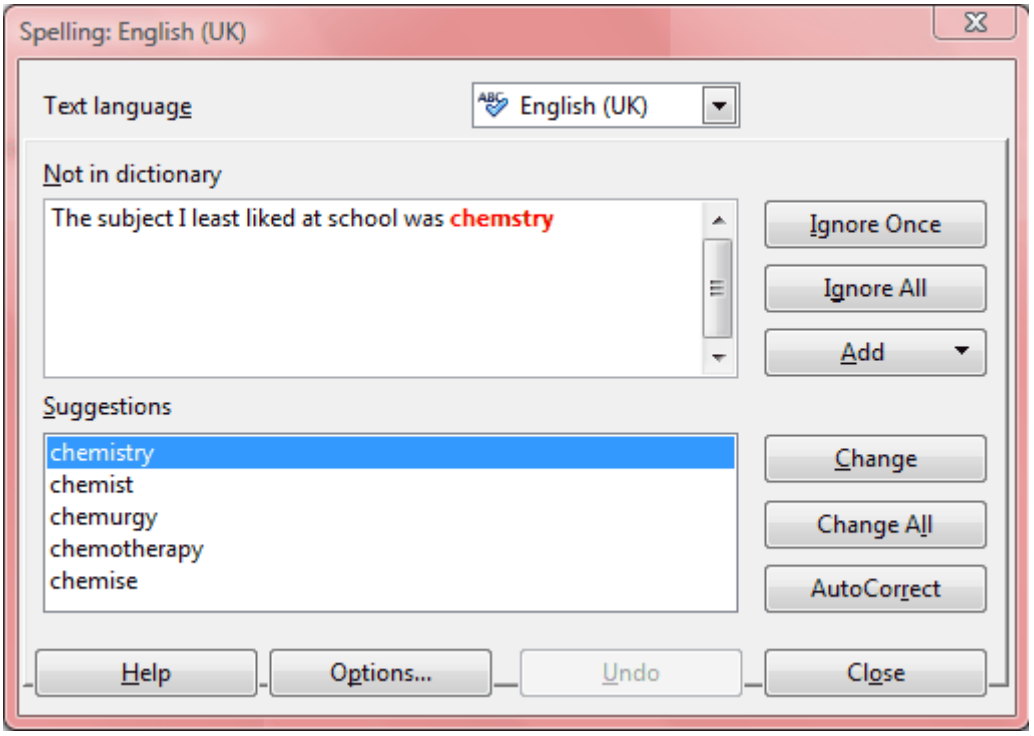
4. In the Export of text files dialog, select the options you want. Click **OK**.

Spell-checking

To toggle the Automatic Spellchecking feature on or off, simply click the Automatic Spellchecking icon on the Main toolbar.

You also have the option to spell-check the entire document with the help of a dialog box and an automatic dictionary that suggests revisions. Just click the Spellchecking icon on the Main toolbar or choose Tools➪Spelling…➪ Check to open the Spellcheck dialogbox.
The spell check dialog box has several useful options for dealing with misspelled words in a document.

1. **Ignore Once** — Clicking Ignore Once causes the spell checker to ignore this one instance of the misspelled word and continue on to the next misspelled word.
2. **Ignore All** — Ignore All ignores the current instance of the misspelled word as well as all future instances. Ignore All essentially "tricks" the spell checker into thinking the word is in the dictionary; however, the next time a spell check dialog box is opened, it will not remember these settings.
3. **Add to Dictionary** — Clicking Add to Dictionary will add the currently misspelled word to the user's dictionary.
4. **Change To** — If end users want to manually edit the misspelled word, they can enter a word to replace the misspelled word in this text box.
5. **Change** — Upon clicking this button, the misspelled word in the Not in Dictionary box will be replaced with the word in the Change To box. This will replace only the current instance of the word.
6. **Change All** — Clicking Change All will replace all occurrences of the misspelled word in the entire document to the word in the Change To box.
7. **Suggestions** — The list in the Suggestions box represents all of the possible words that the end user may have misspelled. Selecting one of these words will place it in the Change To box.
8. **Undo button** - Removes the last change made. The Undo button can be pressed several times to remove the last several changes.



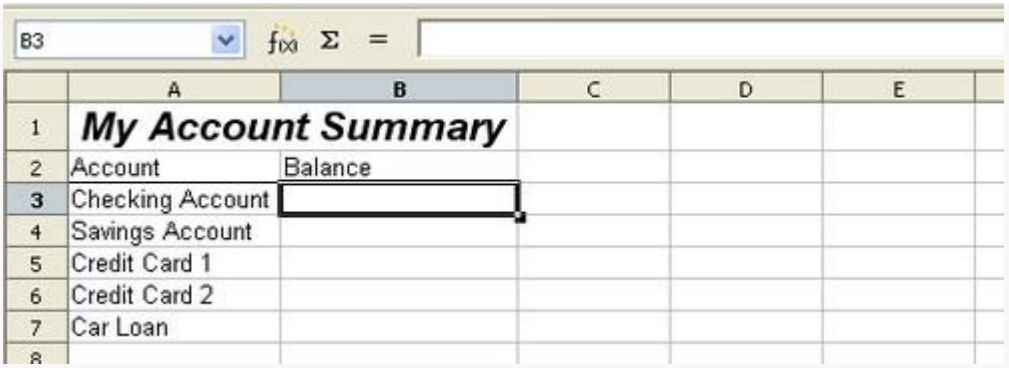
Linking spread sheets

On the *Summary* sheet we display the balance from each of the other sheets. If you copy the example above onto each account, the current balances will be in cell F3 of each sheet.

There are two ways to reference cells in other sheets: by entering the formula directly using the keyboard or by using the mouse. We will look at the mouse method first.

Creating the reference with the mouse

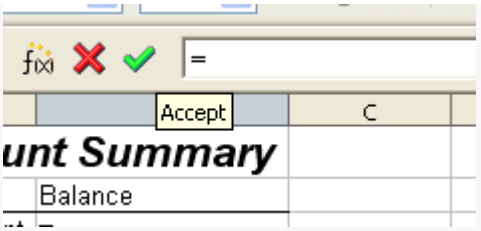
On the *Summary* sheet, set up a place for all five account balances, so we know where to put the cell reference. The figure below shows a summary sheet with a blank Balance column. We want to place the reference for the checking account balance in cell B3.



Blank summary

To make the cell reference in cell B3, select the cell and follow these steps.

1. Click on the = icon next to the input line. The icons change and an equals sign appears in the input line as shown below.



Equal sign in input line

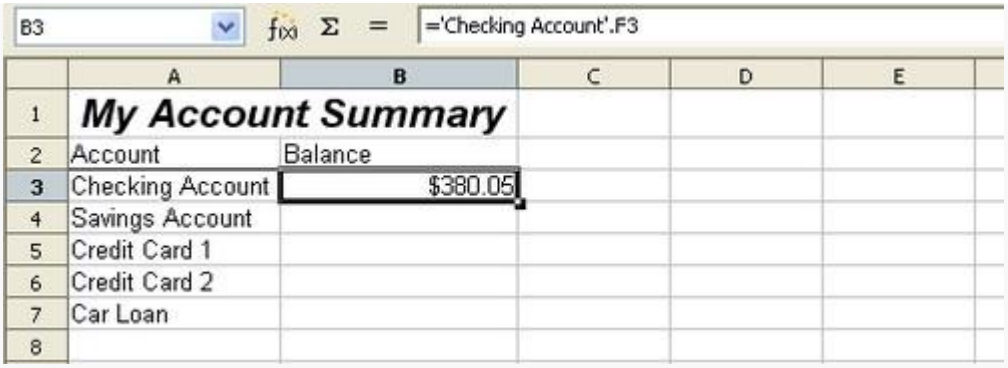
2. Now, click on the sheet tab for the sheet containing the cell to be referenced. In this case, that is the *Checking Account* sheet as shown below.

Click on the checking account tab

3. Click on cell F3 (where the balance is) in the *Checking Account* sheet. The phrase 'Checking Account'.F3 should appear in the input line as shown below.

Cell reference selected

4. Click the green checkmark in the input line to finish.
5. The *Summary* sheet should now look like the figure below.



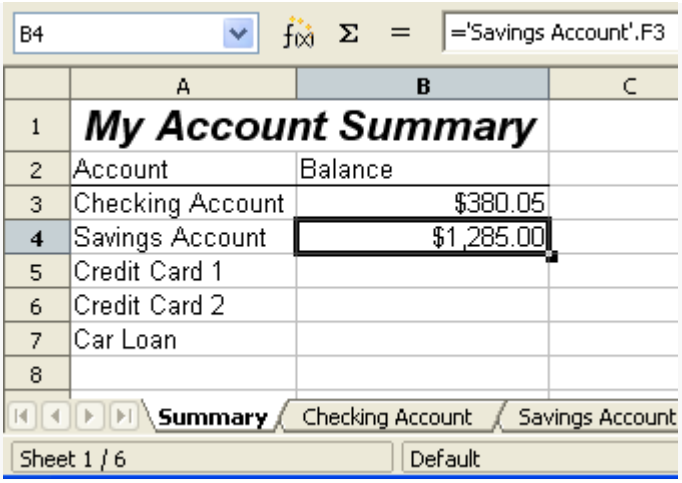
Finished checking account reference

Creating the reference with the keyboard

From the figure, you can deduce how the cell reference is constructed. The reference has two parts: the sheet name (*'Checking Account'*) and the cell reference (*F3*). Notice that they are separated by a period.

The sheet name is in single quotes because it contains a space, and the mandatory period (.) always falls outside any quotes.

So, you can fill in the Savings Account cell reference by just typing it in. Assuming that the balance is in the same cell in the *Savings Account* sheet, F3, the cell reference should be *='Savings Account'.F3*.



Choose Cells and Copy

Right-click a selected cell and then choose "Copy" from the context menu. Switch to the destination spreadsheet.

Establish a Link Between the Two Spreadsheets

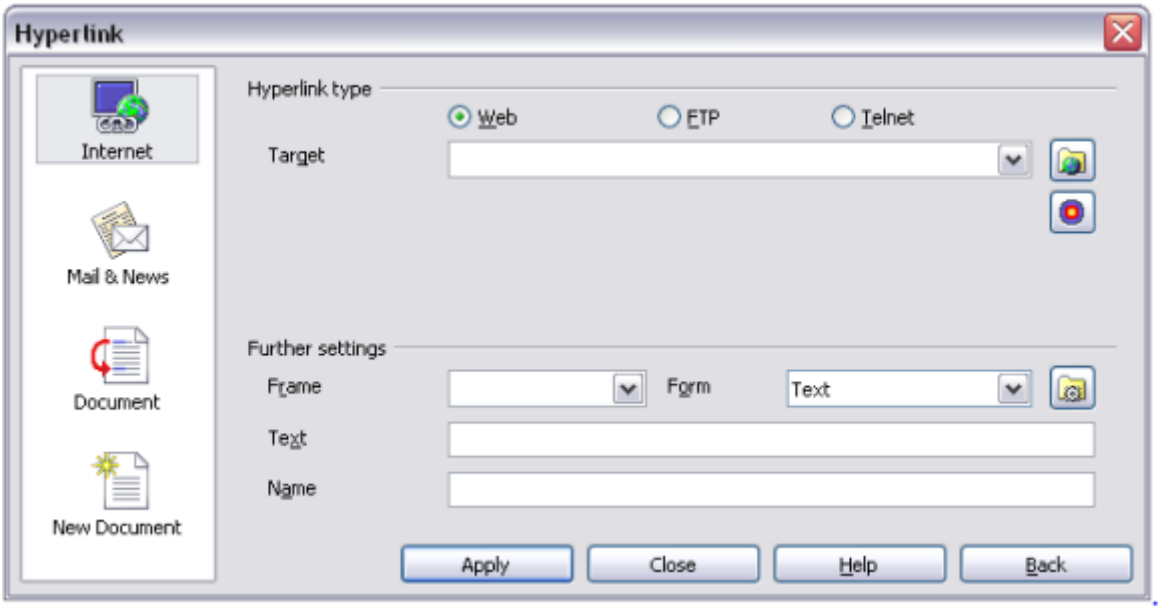
Right-click the target cell. Point to "Paste Special" and select the link option and click on ok

Hyperlink

Hyperlinks can be used in Calc to create spreadsheets that will be used in a web interface or to jump to a different location from within a spreadsheet.

You can also insert and modify links using the Hyperlink dialog as shown in the figure.
To display the dialog, click the Hyperlink icon on the Standard toolbar or select Insert > Hyperlink from the menu bar.
To turn existing text into a link, highlight it before opening the Hyperlink dialog.
On the left hand side, select one of the four types of hyperlinks:
• Internet: a web address, normally starting with http://
• Mail & News: for example an email address.

• Document: the hyperlink points to another document or to another place in the presentation.
• New document: the hyperlink creates a new document.



For a Document type hyperlink, specify the document path (the Open File button opens a file browser); leave this blank if you want to link to a target in the same. Optionally specify the target in the document (for example a specific slide). Click on the Target icon to open the Navigator where you can select the target, or if you know the name of the target, you can type it into the box.
For a New Document type hyperlink, specify whether to edit the newly created document immediately or just create it (Edit later) and the type of document to create (text, spreadsheet, etc.). The Select path button opens a directory picker.
The Further settings section in the bottom right part of the dialog is common to all the hyperlink types, although some choices are more relevant to some types of links.
Form specifies if the link is to be presented as text or as a button.
Text specifies the text that will be visible to the user.

note:while working in formulas when we do not want the reference to be changed when we copy or drag down the formula to other cell references, feature to use $ will keep the reference same for all the further calculations.

1.2.4

In general, the best way to print large spreadsheets is to first preview the print output, then adjust the print settings to arrive at the desired effect.

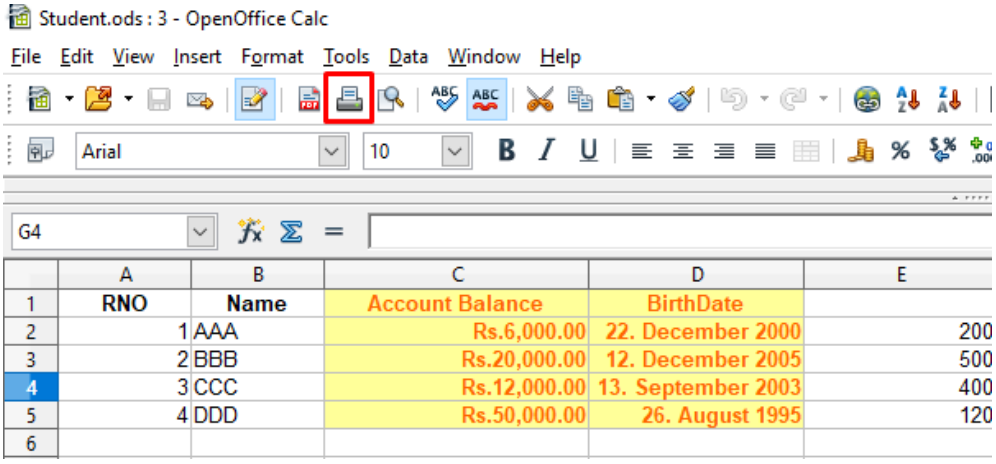
Previewing the print area:

1. With the spreadsheet open, go to **File > Page Preview** in the drop-down menus. Or On the Standard toolbar, select Page preview button for previewing the page.
2. To close the preview window, click the **Close Preview** button on the toolbar.

3. Make adjustments to the print settings, then preview again. Repeat until the print displays in the desired format.

Quick Print

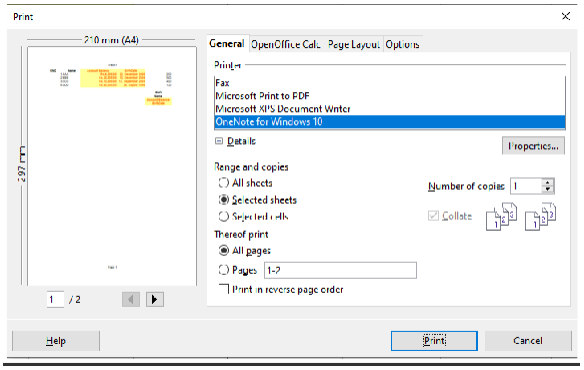
Click the **Print File Directly** icon  to send the entire document to the default printer defined for your computer.



	A	B	C	D	E
1	RNO	Name	Account Balance	BirthDate	
2	1	AAA	Rs.6,000.00	22. December 2000	200
3	2	BBB	Rs.20,000.00	12. December 2005	500
4	3	CCC	Rs.12,000.00	13. September 2003	400
5	4	DDD	Rs.50,000.00	26. August 1995	120
6					

Print

The Print dialog, reached from **File > Print**, has some Calc-specific options: **Print** and **Print Range**.



Select the General on option.

Printer

The **printer** (from the printers available) are displayed.

Properties

Select the **Properties** button to display the selected printer's properties dialog where you can choose portrait or landscape orientation, which paper tray to use, and the paper size to print on from 'Advanced' option.

Range and copies

You can select one or more sheets for printing. This can be useful if you have a large spreadsheet with multiple sheets and only want to print certain sheets.

There are three options: All sheets, Selected sheets, Selected cells. You can choose single sheets, multiple sheets, and selections of cells for printing.

Thereof point

Select whether to print all pages or only some pages. Click the **OK** button.

1. Enter the page number of the page you want to print. The preview box changes to show the selected page.
2. Enter the sequence numbers of the pages to print (for example, 1–4 or 1,3,7,11).

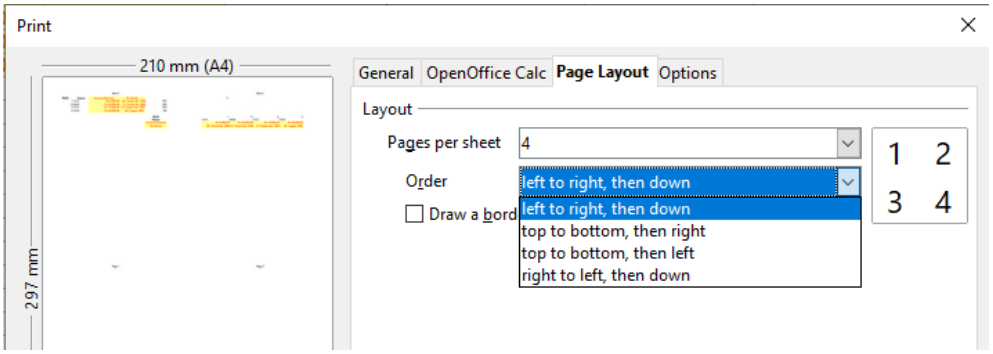
Number of copies

You can give no. of copes to print.

Printing multiple pages on a single sheet of paper

You can print multiple pages of a document on one sheet of paper. To do this:

1. In the Print dialog, select the Page Layout tab.
2. The *Layout* section, select from the drop-down list the number of pages to print per sheet. The preview panel on the left of the Print dialog shows how the printed document will look.
3. When printing more than 2 pages per sheet, you can choose the order in which they are printed across and down the paper.

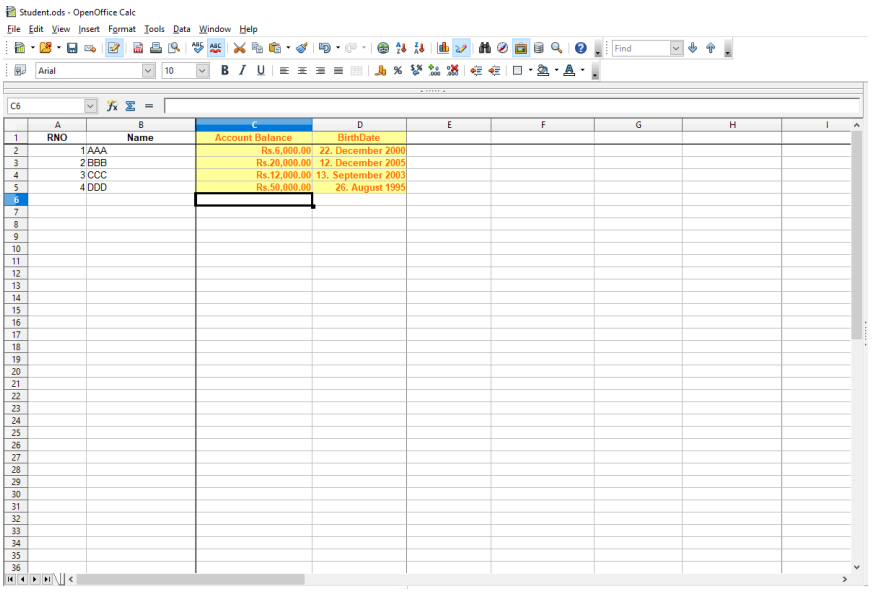


1.2.5

Freezing rows and columns

Freezing locks a number of rows at the top of a spreadsheet or a number of columns on the left of a spreadsheet or both. Then when scrolling around within the sheet, any frozen columns and rows remain in view.

You can set the freeze point at one row, one column, or both a row and a column as in figure above.



Freezing single rows or columns

1. Click on the header for the row below where you want the freeze or for the column to the right of where you want the freeze.
2. Select **Window > Freeze**.

A dark line appears, indicating where the freeze is put.

Freezing a row and a column

1. Click into the cell that is immediately below the row you want frozen and immediately to the right of the column you want frozen.
2. Select **Window > Freeze**.

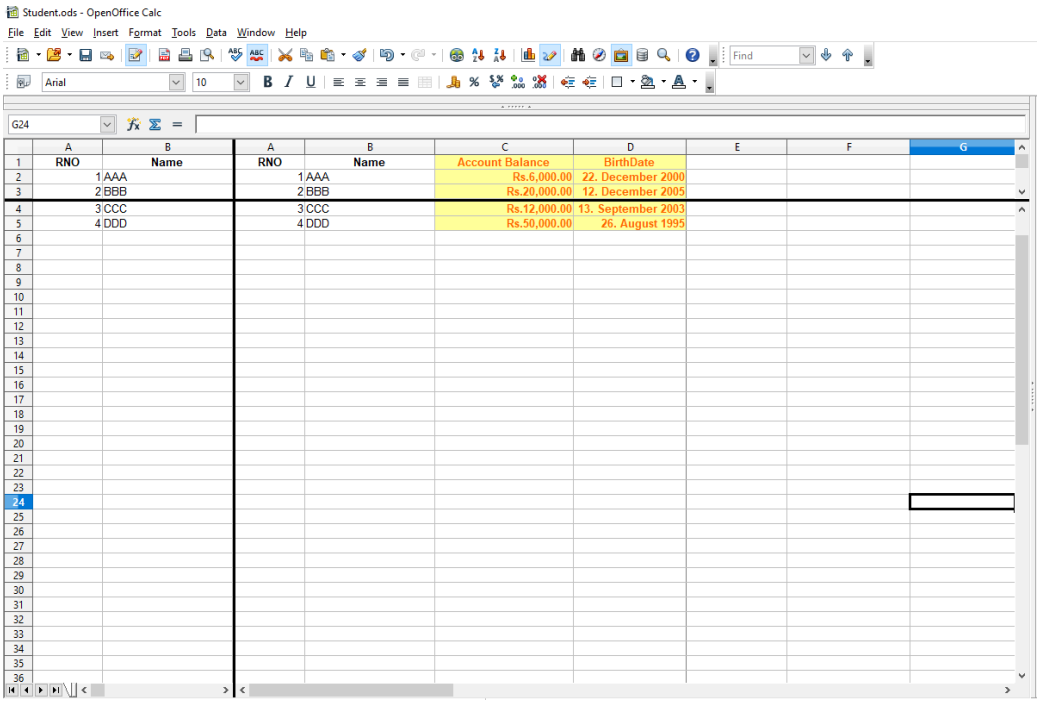
Two lines appear on the screen, a horizontal line above this cell and a vertical line to the left of this cell. Now as you scroll around the screen, everything above and to the left of these lines will remain in view.

Unfreezing

To unfreeze rows or columns, select **Window > Freeze**. The check mark by **Freeze** will vanish.

Splitting the window

Another way to change the view is by splitting the window—also known as splitting the screen. The screen can be split either horizontally or vertically or both. This allows you to have up to four portions of the spreadsheet in view at any one time.

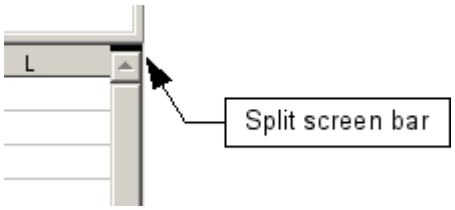


	A	B	A	B	C	D	E	F	G
1	RNO	Name	RNO	Name	Account Balance	BirthDate			
2	1	AAA	1	AAA	Rs.6,000.00	22. December 2000			
3	2	BBB	2	BBB	Rs.20,000.00	12. December 2005			
4	3	CCC	3	CCC	Rs.12,000.00	13. September 2003			
5	4	DDD	4	DDD	Rs.50,000.00	26. August 1995			

Splitting the screen horizontally

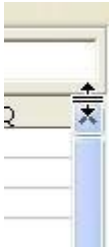
To split the screen horizontally:

1. Move the mouse pointer into the vertical scroll bar, on the right-hand side of the screen, and place it over the small button at the top with the black triangle.



Split screen bar on vertical scroll bar

2. Immediately above this button you will see a thick black line (see above). Move the mouse pointer over this line and it turns into a line with two arrows (see below).



Split screen bar on vertical scroll bar with cursor

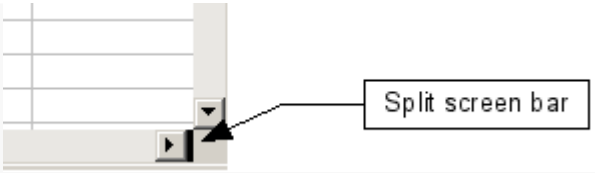
3. Hold down the left mouse button and a gray line appears, running across the page. Drag the mouse downwards and this line follows.

4. Release the mouse button and the screen splits into two views, each with its own vertical scroll bar.

Splitting the screen vertically

To split the screen vertically:

1. Move the mouse pointer into the horizontal scroll bar at the bottom of the screen and place it over the small button on the right with the black triangle.



Split bar on horizontal scroll bar

2. Immediately to the right of this button is a thick black line. Move the mouse pointer over this line and it turns into a line with two arrows.
3. Hold down the left mouse button and a gray line appears, running up the page. Drag the mouse to the left and this line follows.
4. Release the mouse button and the screen is split into two views, each with its own horizontal scroll bar.

Note: Splitting the screen horizontally and vertically at the same time gives four views, each with its own vertical and horizontal scroll bars.

Removing split views

To remove a split view:

1. Double-click on each split line, or
2. Click on and drag the split lines back to their places at the ends of the scroll bars, or
3. Select **Window > Split**. This will remove all split lines at the same time.

Hiding Data

To hide or show sheets, rows, and columns, use the options on the Format menu or the right-click (context) menu.

1. To hide a single row or multiple rows, first select the single row or multiple rows, and then choose **Format > Row > Hide** (or right-click and choose **Hide**).
2. To hide a single column or multiple columns, first select the single column or multiple columns, and then choose **Format > Column > Hide** (or right-click and choose **Hide**).
3. To hide a sheet, first select the sheet, and then choose **Format > Sheet > Hide**.

Unit 2: Formulas, Chart and Data

Charts

Creating 2D and 3D Charts (Columns, Line, Pie, Bar, Scatter)

Charts and graphs can be powerful ways to convey information to the reader.OpenOffice.org Calc offers a variety of different chart and graph formats for your data.
Open the chart Wizard dialog using one of two methods.

- Choose Insert > Chart from the menu bar.
- Or click the chart icon on the main toolbar.

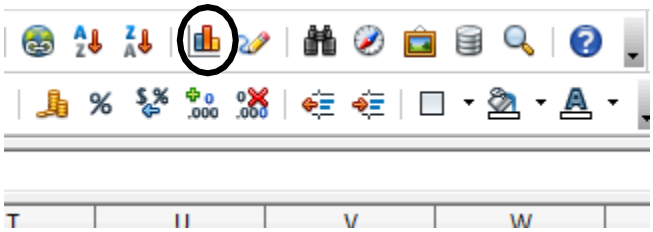


Figure 2.1: Insert chart from main toolbar

Either method inserts a sample chart on the worksheet, opens the Formatting toolbar, and opens the Chart Wizard, as shown in above figure.

Step 1 choosing a chart type

The Chart Wizard includes a sample chart with your data. The Chart Wizard has three main parts: a list of steps involved in setting up the chart, a list of chart types, and the options for each chart type. There are main 10 basic chart types which may be 2D or 3D.Select chart type and press next button.

The first tier of choice is for two-dimensional (2D) charts. Only those types which are suitable for 3D (Column, Bar, Pie, and Area) give you an option to select a 3D look.

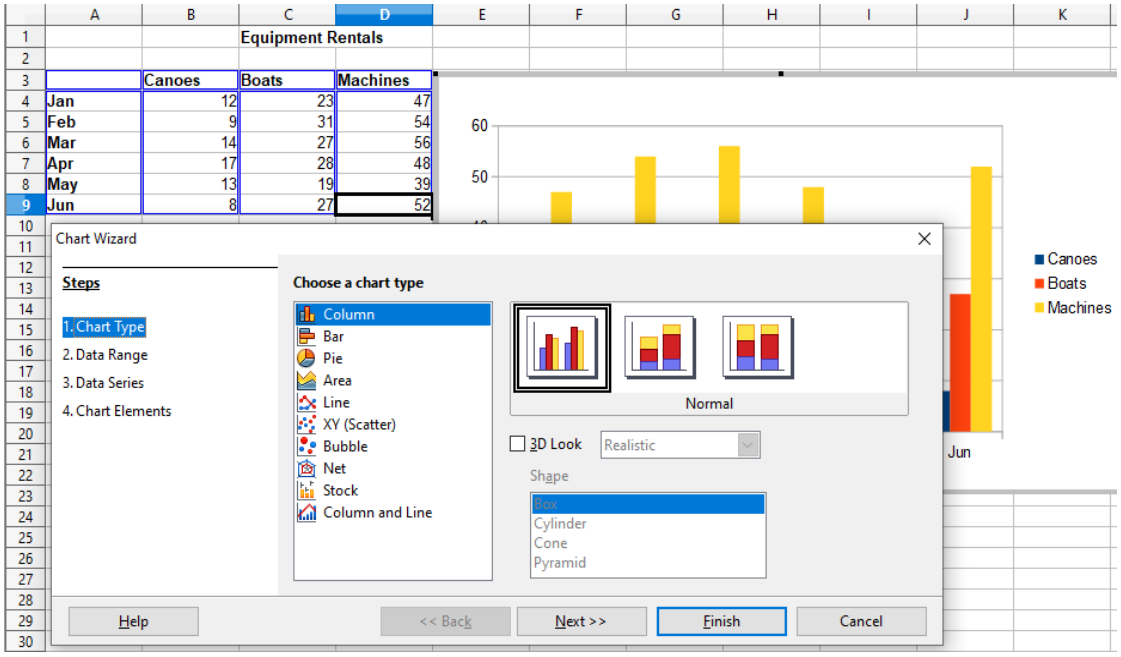


Figure 2.2: Chart Wizard, Choose a chart type

Step 2 Changing data ranges and axes labels

You can select data range manually or wizard. You can choose data series plotting by rows or columns. You can choose whether to use the first row or first column, or both, as labels on the axes of the chart. You can confirm what you have done so far by clicking the Finish button, or click Next to change some more details of the chart.

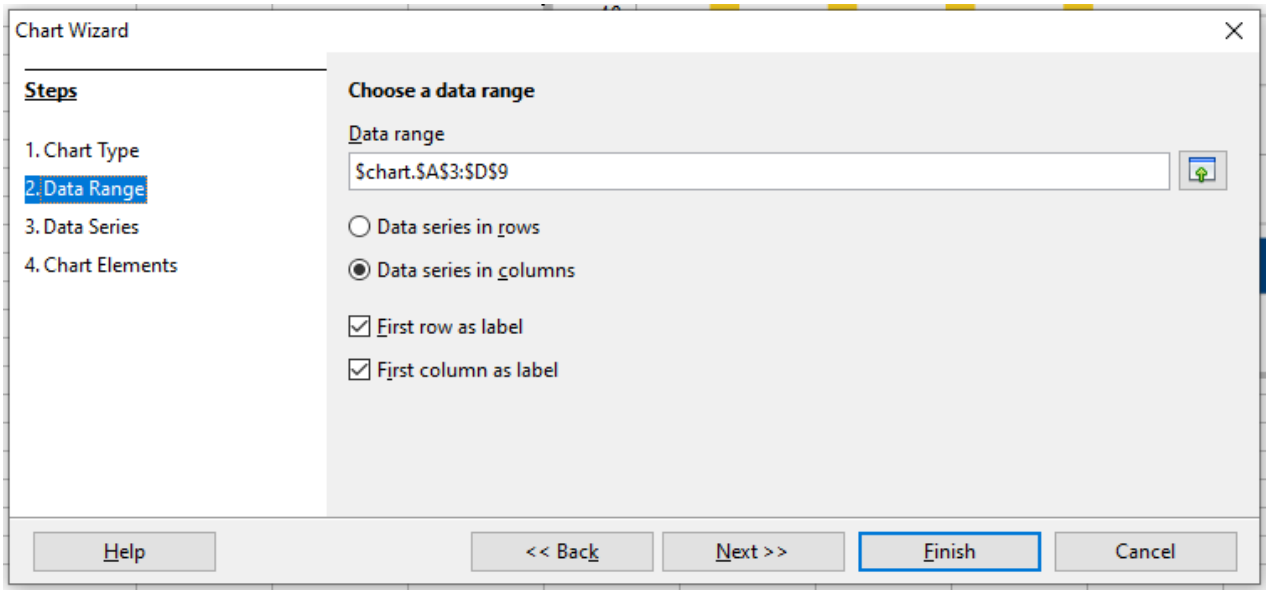


Figure 2.3: Changing data ranges and axes labels

Step 3 Selecting data series

On the Data series pages, you can select the data that you want to include in the chart. If you do not want any data just select and click on Remove.

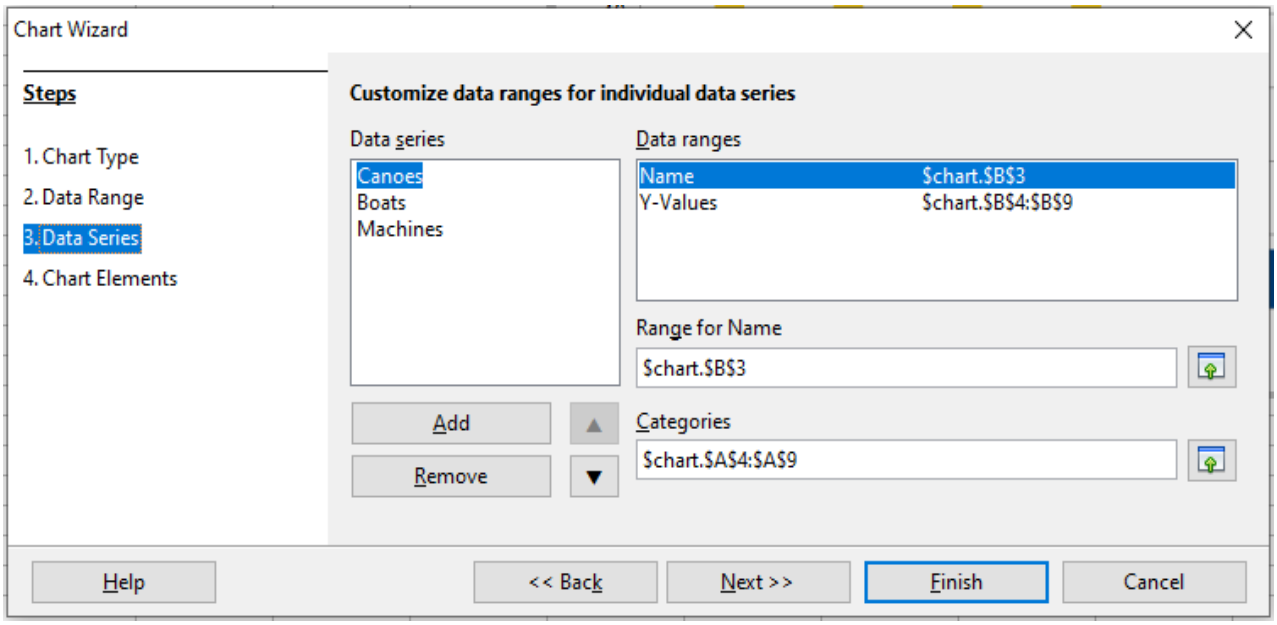
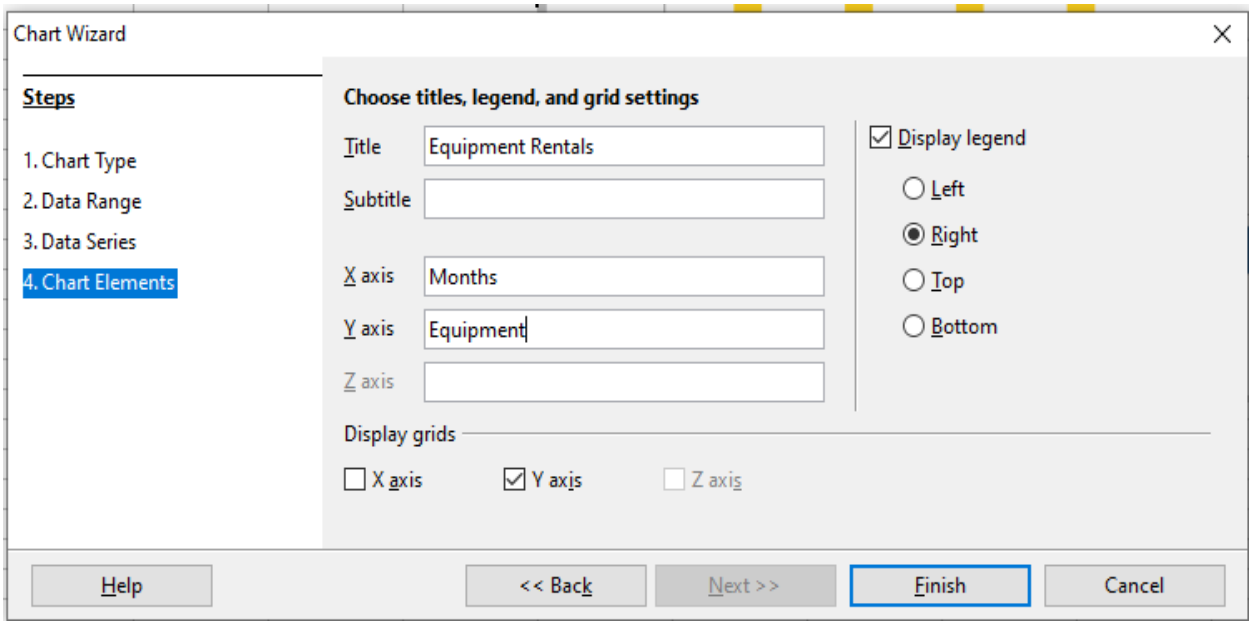


Figure 2.4: Amending data series and ranges

Step 4 Adding or changing titles, legend, and grids



On the Charts Elements page, you can give chart a title, subtitle, labels for the x axis or the y axis. You can leave out the legend or include it to the left, top or bottom. To confirm your selections and complete the chart, click Finish.

❖ **Adding or removing chart elements**

Below Figures show the elements of 2D and 3D charts. The default 2D chart includes only two of those elements:

- *Chart wall* contains the graphic of the chart displaying the data.
- *Chart area* is the area surrounding the chart graphic. The (optional) chart title and the legend (key) are in the chart area.
- The default 3D chart also has the *chart floor*, which is not available in 2D charts.

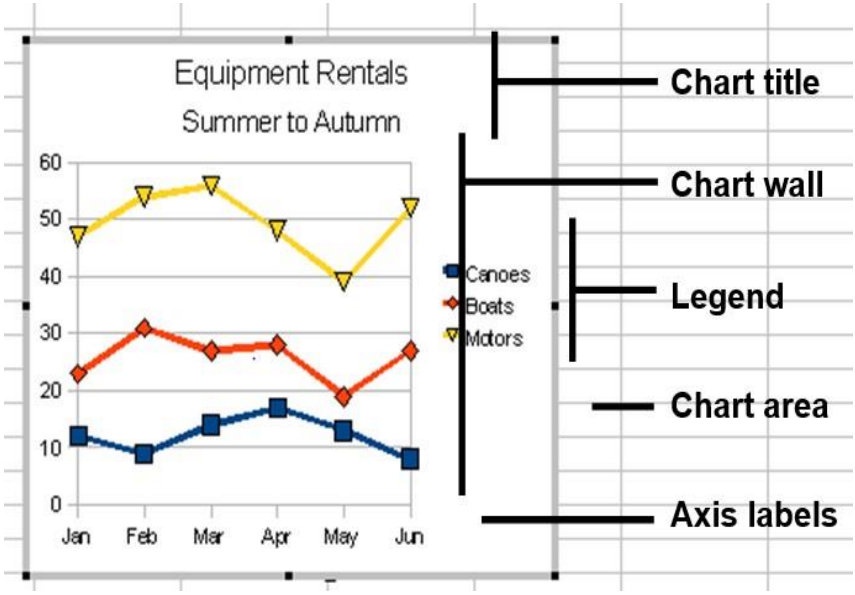


Figure 2.5: Elements of 2D chart

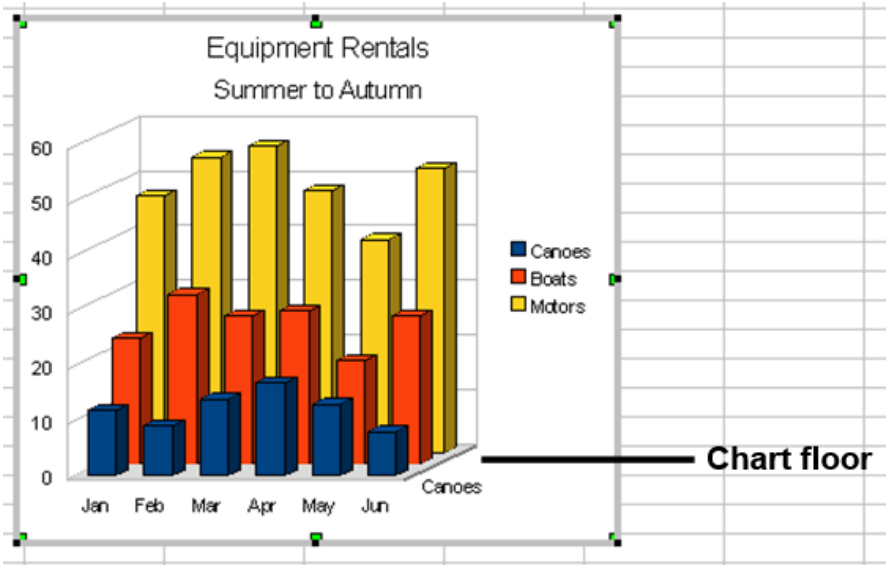


Figure 2.6: Elements of 3D chart

You can add other elements using the commands on the **Insert** menu. The various choices open dialogs in which you can specify details.

❖ Formatting 3D charts

Use **Format** > **3D View** to fine tune 3D charts. The 3D View dialog has three pages, where you can change the perspective of the chart, determine whether the chart uses the simple or realistic schemes or your own custom scheme, and the illumination that controls where the shadows will fall.

- **Rotation and perspective**
  To rotate a 3D chart or view it in perspective, enter the required values on the *Perspective* tab of the 3D View dialog.

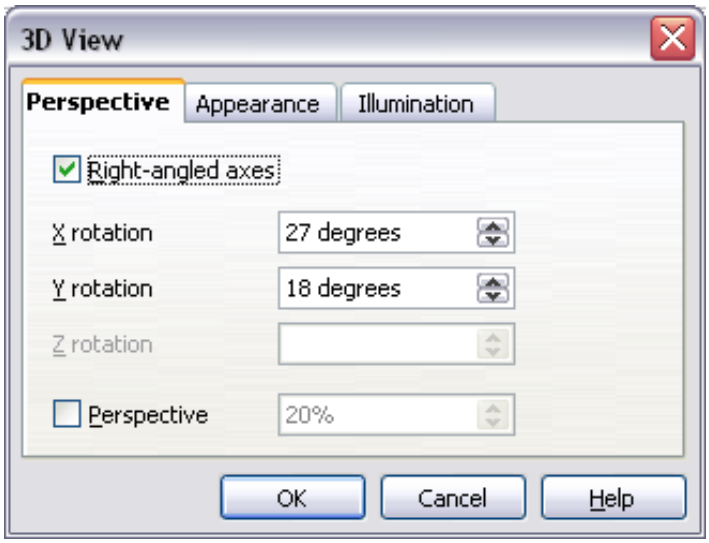


Figure 2.7: Rotating a chart

- **Appearance**
  Use the *Appearance* page to modify some aspects of a 3D chart's appearance.

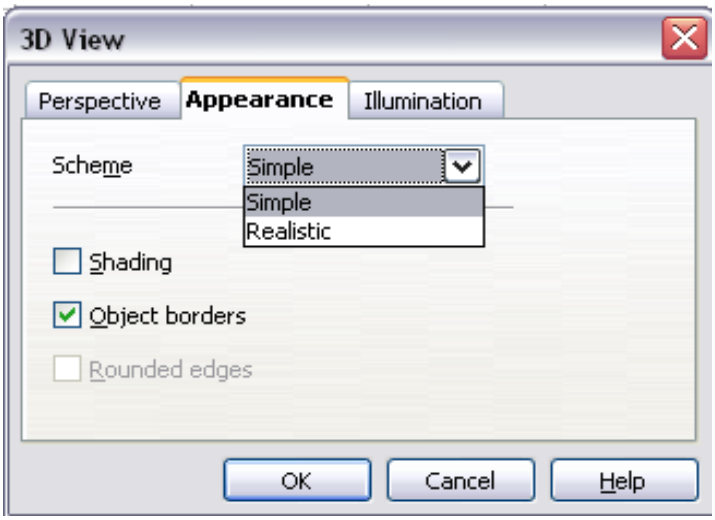


Figure 2.8: Modifying appearance of 3D chart

Select a scheme from the list box. When you select a scheme, the options and the light sources are set accordingly

Select **Shading** to use the Gouraud method for rendering the surface, which applies gradients for a smoother, more realistic look. Otherwise, a flat method is used. The flat method sets a single color and brightness for each polygon. The edges are visible, soft gradients and spot lights are not possible.

Select **Object Borders** to draw lines along the edges.

Select **Rounded Edges** to smooth the edges of box shapes. In some cases this option is not available.

- **Illumination**

Use the *Illumination* page to set the light sources for the 3D view. Click any of the eight buttons to switch a directed light source on or off. For the selected light source, you can then choose a color and intensity in the list just below the eight buttons. The brightness values of all lights are added, so use dark colors when you enable multiple lights.

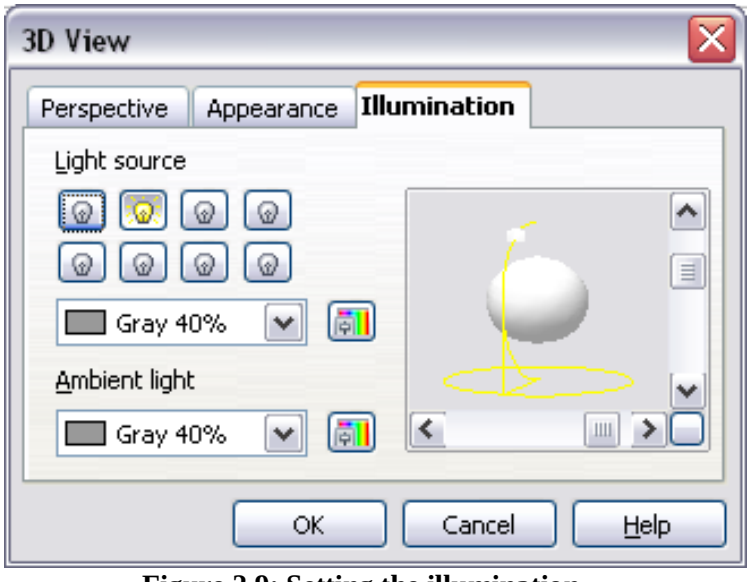
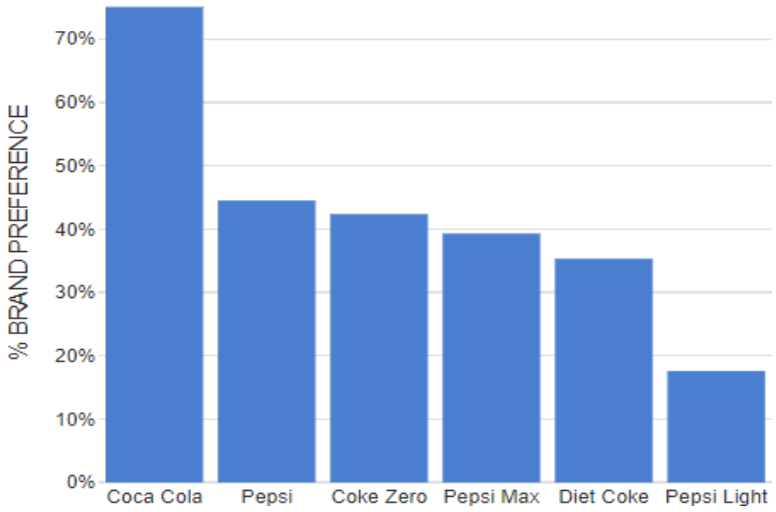


Figure 2.9: Setting the illumination

❖ Column chart

Column charts are commonly used for data that shows trends over time. They are best for charts that have a relatively small number of data points. It is the default chart type, as it is one of the most useful charts and the easiest to understand. A column chart is a data visualization where each category is represented by a rectangle, with the height of the rectangle being proportional to the values being plotted. Column charts are also known as vertical bar charts.
In the example below, the height of each bar is proportional to the percentage of people who listed each type of cola as being their favourite.



❖ Line Chart:

A *line chart* is a time series with a progression. It is ideal for raw data, and useful for charts with plentiful data that show trends or changes over time where you want to emphasize continuity. On line charts, the x-axis is ideal to represent time series data.

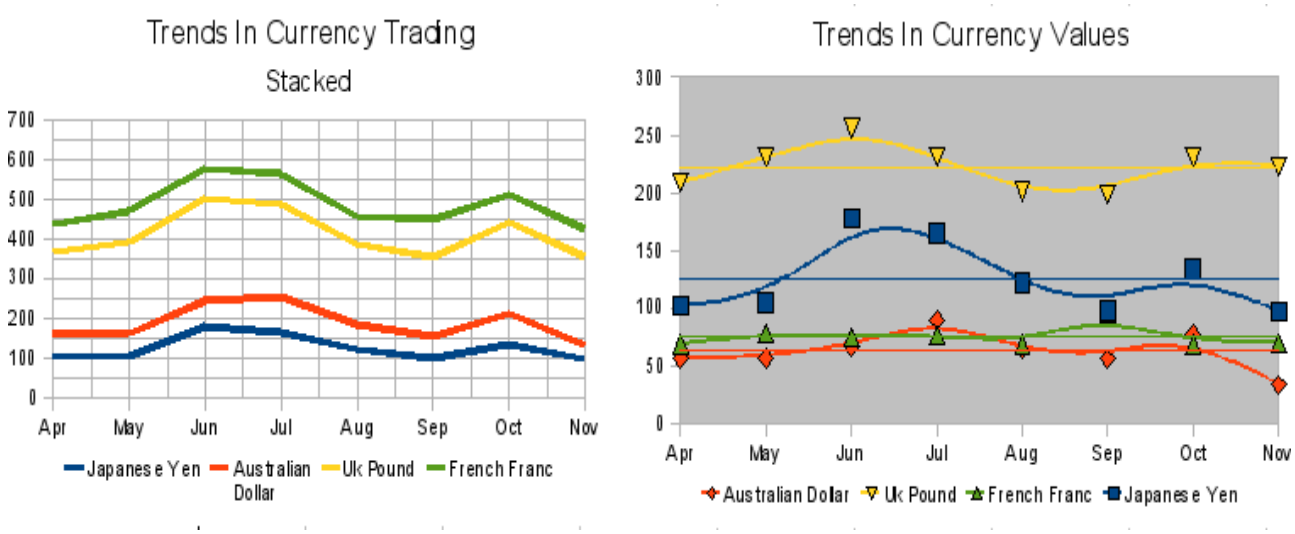


Figure 2.10: Line charts

❖ Pie Chart:

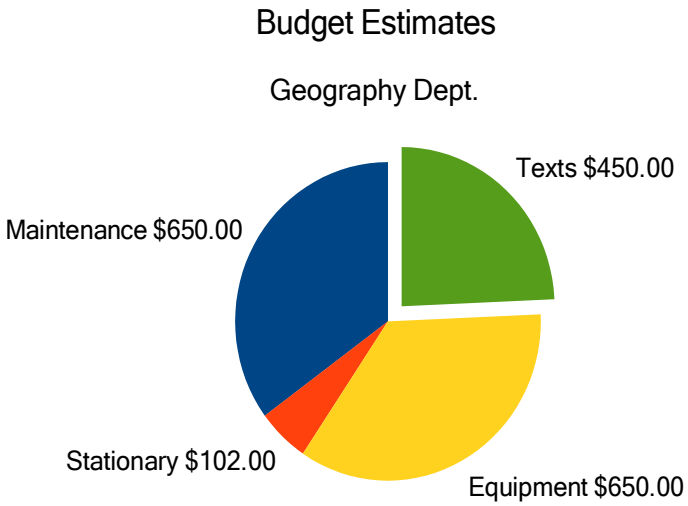
Pie charts are excellent when you need to compare proportions. For example, a pie chart would be ideal if you needed to figure out comparisons of departmental spending, what the department

spent on different items or what different departments spent. These charts work best with smaller numbers of values.

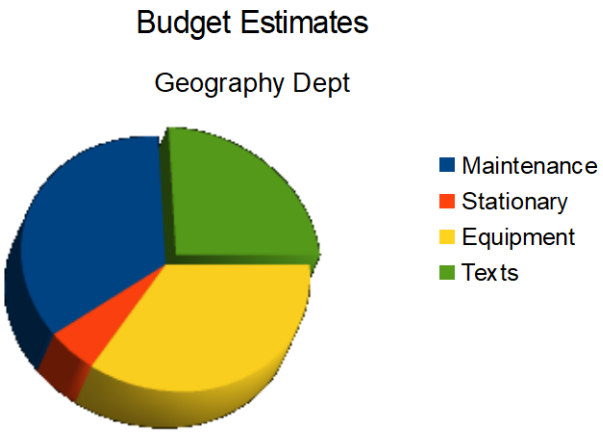
You can do some interesting things with a pie chart, especially if you make it into a 3D chart. It can then be tilted, given shadows, and generally turned into a work of art.

The effects achieved in Figure are explained below.

- The first example is a 2D pie chart with one part of the pie exploded. To produce this type of chart, first choose **Insert > Legend** and deselect **Display legend**. Choose **Insert > Data Labels** and choose **Show value as number**. Then carefully select the piece you wish to highlight, move the cursor to the edge of the piece and click (the piece will have nine green highlight squares to mark it), and then dragit out from the rest of the pieces.

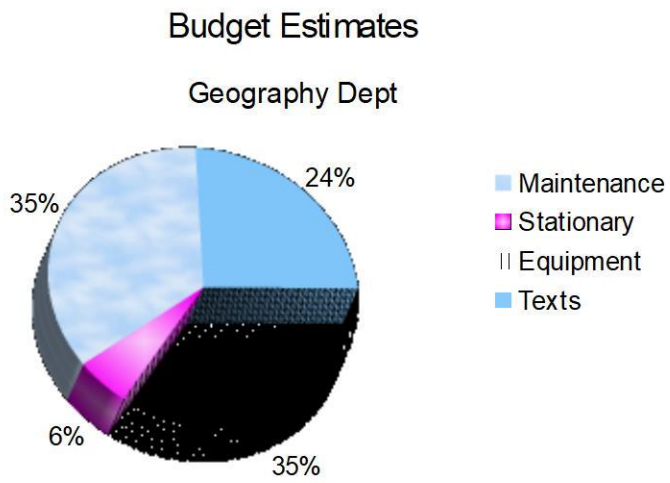


- The second example is a 3D pie chart with realistic schema and illumination. With a completed 2D pie chart, choose **Format > 3D view > Illumination** where you can change the direction of the light, the color of the ambient light, and the depth of the shade. We also adjusted the 3D angle of the disc in the *Perspective* dialog on the same set of tabs.



- The third example is a 3D pie chart with different fill effects in each portion ofthe pie.

Choose **Insert > Data labels** and select **Show value as percentage**. Carefully select each of the pieces so that it has a wire frame, then highlight and right-click to get the object properties dialog. Choose the *Area* tab. For one of the pieces we chose a bitmap effect, for another we selected a gradient feature and for the third we used the *Transparency* tab.



❖ Bar Chart:

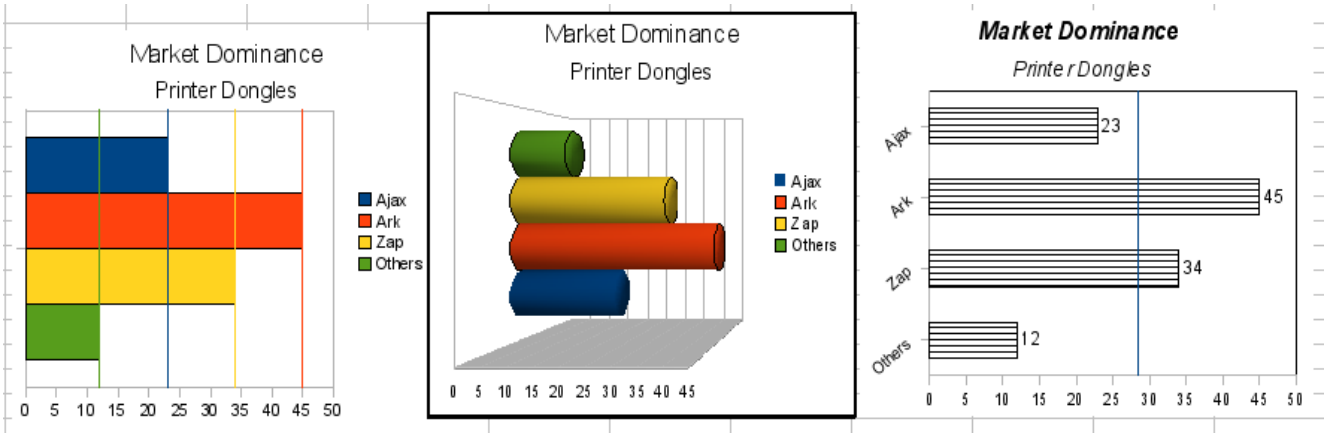


Figure 2.11: Three bar graph treatments.

Bar charts are excellent for giving an immediate visual impact for data comparison in cases when time is not an important factor, for example when comparing the popularity of a few products in a marketplace.

- The first chart in above Figure is achieved quite simply by using the chart wizard with **Insert > Grid**s, deselecting y-axis, and using **Insert > Mean Value Lines.**
- The second chart in the figure is the 3D option in the chart wizard witha simple border and the 3D chart area twisted around.
- The third chart in the figure is an attempt to get rid of the legend andput labels showing the names of the companies on the axis instead. We also changed the colors to a hatch pattern.

❖ Scatter or XY charts

Scatter charts are great for visualizing data that you have not had time to analyze, and they may be the best for data when you have a constant value against which to compare other data.

Examples of good scatter charts might include weather data, reactions under different acidity levels, conditions at altitude or any data which matches two series of numeric data. In contrast to line charts, the x-axis is to the left of the right labels, which usually indicates a time series.

Scatter charts may surprise those unfamiliar with how they work. While constructing the chart,

If you choose **Data Range > Data series in rows**, the first row of data represents the x-axis. The rest of the rows of data are then compared against the first row data. Figure 90 shows a comparison of three currencies with the Japanese Yen.
Even though the table presents the monthly series, the chart does not. In fact the Japanese Yen does not appear; it is merely used as the constant series that all the other data series are compared against.

	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov
Japanese Yen	102	105	178	165	122	98	134	97
Australian Dollar	56	57	67	90	64	56	78	34
Uk Pound	209	230	256	231	201	199	230	223
French Franc	69	78	75	76	69	95	69	70

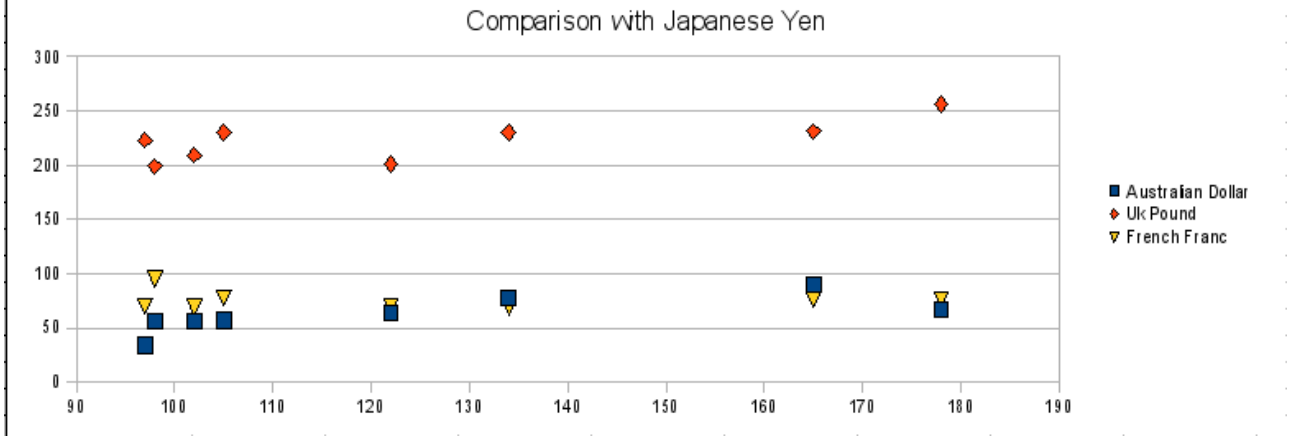


Figure 2.12: A particularly volatile time in the world currency market.

Difference among columns, Line and Bar Charts.

Bar Chart	Column Chart	Line Chart
Bar charts use horizontal bars to display data and are used to compare values across categories. The lengths of the bars are proportional to the values they represent.	A simple column chart uses vertical bars to display data. The lengths of the bars are proportional to the values they represent.	Line graphs (also called a *line chart* or *run chart*) can include a single line for one data set, or multiple lines to compare two or more sets of data.
Bar charts are used to compare values across categories and can be used to show change over a period of time.	Column charts are used to compare values across categories and can be used to show change over a period of time.	It is ideal for raw data, and useful for charts with plentiful data that show trends or changes over time.

For a bar chart the Y axis typically displays a category. While the X axis displays a discrete value.	In a column chart the Y axis typically displays a discrete value whilst the X axis displays the category.	On a line graph, the X axis is the independent variable and generally shows time periods. Y axis is the dependent variable and shows the data you are tracking.
Bar graphs are an extremely effective visual to use in presentations and reports. They are popular because they allow the reader to recognize patterns or trends far more easily than looking at a table of numerical data.	Column charts are ideal if you need to compare a single category of data between individual sub-items, such as, for example, when comparing revenue between regions, Showing average scores ranked by subjects to show which subject students are weak in.	Line charts should be used only for time series (chronological) or when there is some other sequence to the dimensions on the x-axis, e.g. dates, months, sequence of stages of a project, sequence of meters along on a gas pipeline.

Mathematical functions

SUM

Sums the contents of cells.

Syntax:

SUM(number1; number2; ... number30)

number1 to **number30** are up to 30 numbers or ranges/arrays of numbers whose sum is to be calculated.

SUM ignores any text or empty cell within a range or array.

Example:

SUM(2; 3; 4)

returns **9**, because **2**+**3**+**4** = **9**.

SUM(B1:B3)

(where cells **B1**, **B2**, **B3** contain **1.1**, **2.2**, **3.3**) returns **6.6**.

Statistical functions

AVERAGE

Returns the average of the arguments, ignoring text.

Syntax:

AVERAGE(number1; number2; ... number30)

number1 to **number30** are up to 30 numbers or ranges containing numbers.

Example:

AVERAGE(2; 6; 4)

returns **4**, the average of the three numbers in the list.

AVERAGE(B1:B3)

where cells **B1**, **B2**, **B3** contain **1**, **3**, and **apple** returns **2**, the average of **1** and **3**. Text is ignored.

COUNT

Counts the numbers in the list of arguments, ignoring text entries.

Syntax:

COUNT(value1; value2; ... value30)

value1 to **value30** are up to 30 values or ranges representing the values to be counted.

Examples:

COUNT(2; 4; 6; "eight")

returns **3**, because **2**, **4** and **6** are numbers (**"eight"** is text).

COUNT(B1:B3)

where cells **B1**, **B2**, **B3** contain **1.1**, **2.2**, and **apple** .It returns **2**.

COUNT(B1:B3)

where cells **B1**, **B2**, **B3** are empty, returns **0**.

MAX

Returns the maximum of a list of arguments, ignoring text entries.

Syntax:

MAX(number1; number2; ... number30)

number1 to **number30** are up to 30 numbers or ranges containing numbers.

Example:

MAX(2; 6; 4)

returns **6**, the largest value in the list.

MAX(B1:B3)

where cells **B1**, **B2**, **B3** contain **1.1**, **2.2**, and **apple** returns **2.2**.

MIN

Returns the minimum of a list of arguments, ignoring text entries.

Syntax:

MIN(number1; number2; ... number30)

number1 to **number30** are up to 30 numbers or ranges containing numbers.

Example:

MIN(2; 6; 4)

returns **2**, the smallest value in the list.

MIN(B1:B3)

where cells **B1**, **B2**, **B3** contain **1.1**, **2.2**, and **apple** returns **1.1**.

Mathematical functions

SUMIF

Conditionally sums the contents of cells in a range.

Syntax:

SUMIF(test_range; condition; sum_range)

This function identifies those cells in the range **test_range** that meet the **condition**, and sums the corresponding cells in the range **sum_range**. If **sum_range** is omitted the cells in **test_range** are summed.

condition may be:

a number, such as **34.5**

an expression, such as **2/3** or **SQRT(B5)**

a text string

SUMIF looks for cells in **test_range** that are equal to **condition**, unless **condition** is a text string that starts with a comparator:

>, <, >=, <=, =, <>

In this case **SUMIF** compares those cells in **test_range** with the remainder of the text string (interpreted as a number if possible or text otherwise).

For example the condition "**>4.5**" tests if the content of each cell is greater than the number 4.5, and the condition "**<dog**" tests if the content of each cell would come alphabetically before the text **dog**.

Blank (empty) cells in **test_range** are ignored (they never satisfy the condition).

condition can only specify one single condition.

Example:

SUMIF(A1:A9;"<0")

returns the sum of the negative numbers in **A1:A9**.

SUMIF(A1:A9; F1)

where **F1** contains the text **>=0** (without double quotes) returns the sum of the positive numbers in **A1:A9**.

SUMIF(B2:B4; "<"&F2; C2:C4)

where **F2** contains **10** and cells **B2**, **B3**, **B4** contain **7**, **9**, **11**, returns the sum of **C2** and **C3**, because cells **B2** and **B3** are less than **10**.

SUMIF(D1:D9; "apples"; E1:E9)

where cells in **D1:D9** contain either **apples** or **pears** and cells in **E1:E9** contain the corresponding quantities of each fruit, returns the total quantity of **apples**.

Statistical functions

STDEV

Returns the sample standard deviation of the arguments.

Syntax:

STDEV(number1; number2; ... number30)

number1 to **number30** are up to 30 numbers or ranges containing numbers.

STDEV returns the standard deviation where **number1** to **number30** are a **sample** of the entire population. With *N* values in the sample, the calculation formula is:

$$\sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i-\bar{x})^2}$$

.

Example:

STDEV(2; 6; 4)

returns **2**.

Note:

In statistics, the standard deviation is a measure of the amount of variation or dispersion of a set of values. A low standard deviation indicates that the values tend to be close to the mean (also called the expected value) of the set, while a high standard deviation indicates that the values are spread out over a wider range.

It is the **square root** of the **Variance.**

The Variance is defined as:

The average of the squared differences from the Mean.

Financial functions

PMT

Returns the payment per period for a fixed rate loan.

Syntax:

PMT(rate; numperiods; principal; finalbalance; type)

rate: the interest rate per period.

numperiods: the total number of payment periods in the term. nper-number of periods

principal: the initial sum borrowed.PV-PRESENT VALUE

finalbalance: the cash balance you wish to attain at the end of the term (optional - defaults to 0). With a loan, this would normally be 0.

type: when payments are made (optional - defaults to 0):

0 - at the end of each period.

1 - at the start of each period (including a payment at the start of the term).

See the examples for how this function can be used for building up savings with fixed regular payments.

Example:

PMT(5.5%/12; 12*2; 5000; 0; 0)

returns -220.48 in currency units. You take out a 2 year loan of 5000 currency units at a yearly interest rate of 5.5%, making monthly payments at the end of the month. You pay 220.48 currency units each month; it is given as negative because you pay it.

PMT(5%/12; 12*2; 0; 1000; 1)

returns -39.54 in currency units. You wish to save 1000 currency units over 2 years, making monthly payments, beginning today. You assume the rate will remain the same at 5%. Interest is compounded monthly. If you save 39.54 currency units each month, the value at the end of 2 years will be 1000 currency units.

PMT(5.5%/12; 12*2; 5000; 1000; 0)

returns -259.99 in currency units. You take out a 2 year loan of 5000 currency units at a yearly interest rate of 5.5%, making monthly payments at the end of the month. You wish to build up a lump sum of 1000 currency units, to be paid to you at the end of the term. Interest is compounded monthly.

List of Calc Logical functions

The logical functions operate on logical ('boolean') values, that is, **TRUE** or **FALSE**.

AND returns **TRUE** if all the arguments are **TRUE**.

FALSE returns the logical value **FALSE**.

IF returns one of two values, depending on a test condition.

NOT returns **TRUE** if the argument is **FALSE**, and **FALSE** if the argument is **TRUE**.

OR returns **TRUE** if any of the arguments are **TRUE**.

TRUE returns the logical value **TRUE**.

IF

Returns one of two values, depending on a test condition.

Syntax:

IF(test; value1; value2)

where:

test is or refers to a logical value or expression that returns a logical value (**TRUE** or **FALSE**).

value1 is the value that is returned by the function if **test** yields **TRUE.**

value2 is the value that is returned by the function if **test** yields **FALSE**.

If **value2** is omitted it is assumed to be **FALSE**; if **value1** is also omitted it is assumed to be **TRUE**.

Example:

IF(A1>5; 100; "too small")

returns the number 100 if A1 is greater than 5, and the text "too small" otherwise.

IF(1>2; "nonsense")

returns **FALSE** - because **value2** has been omitted and 1 is not greater than 2.

IF(2>1)

returns **TRUE** - because both **value1** and **value2** have been omitted and 2 is more than 1.

IF(1=2; 1/0; SQRT(4))

returns **2**, the square root of **4**. **IF()** only calculates the value chosen - in this case **1/0** would give a #DIV/0! error, but is not calculated.

AND

Returns **TRUE** if all the arguments are considered **TRUE**, and **FALSE** otherwise.

Syntax:

AND(argument1; argument2 ...argument30)

argument1 to **argument30** are up to 30 arguments, each of which may be a logical result or value, or a reference to a cell or range.

AND tests every value (as an argument, or in each referenced cell), and returns **TRUE** if they are all **TRUE**. Any value which is a **non-zero number** or **text** is considered to be **TRUE**.

Example:

If cells A5:B8 all contain **TRUE**, cell C2 contains **=TRUE()** and cell C3 contains "**dog**":

AND(2<4;A5:B8;C2)

returns **TRUE**.

AND(2<4;FALSE)

returns **FALSE**.

AND(C2:C3))

returns **TRUE**.

OR

Returns **TRUE** if any of the arguments are considered **TRUE**, and **FALSE** otherwise.

Syntax:

OR(argument1; argument2 ...argument30)

argument1 to **argument30** are up to 30 arguments, each of which may be a logical result or value, or a reference to a cell or range.

OR tests every value (as an argument, or in a each referenced cell), and returns **TRUE** if any of them are **TRUE**. Any **non-zero number** is considered to be **TRUE**. Any **text** cells in ranges are ignored.

Example:

OR(TRUE; FALSE)

returns **TRUE**.

OR(0; 5)

returns **TRUE**, because 5 is considered **TRUE**.

If cells A5:B8 all contain **FALSE**, and cell C2 contains **=TRUE()**:

OR(1>2; A5:B8; C2)

returns **TRUE**, because cell C2 is **TRUE**.

NOT

Reverses the logical value. Returns **TRUE** if the argument is **FALSE**, and **FALSE** if the argument is **TRUE**.

Syntax:

NOT(logical_value)

where **logical_value** is the logical value to be reversed.

Example:

NOT(TRUE())

returns **FALSE**

Issues:

- Entering **=NOT(TRUE)** in a cell correctly returns **FALSE**, but the display in the formula bar is **=NOT(1)**.

TRUE

Returns the logical value **TRUE**.

Syntax:

TRUE()

The **TRUE()** function has no arguments, and always returns the logical value **TRUE**.

Example:

TRUE()

returns **TRUE**

NOT(TRUE())

returns **FALSE**

FALSE

Returns the logical value **FALSE**.

Syntax:

FALSE()

The **FALSE()** function has no arguments, and always returns the logical value **FALSE**.

Example:

FALSE()

returns **FALSE**

NOT(FALSE())

returns **TRUE**

Date and Day function

DATE

Returns the date, expressed as a date-time serial number.

Syntax:

DATE(year ;month ;day)

year is an integer between 1583 and 9956 or between 0 and 99;month and day are integers.

If month and day are not within range for a valid date, the date will 'roll over'.

Example:

DATE(2007;11;9)

returns the date 9th November 2007(as a date-time serial number).

DATE(2007;12;32)

returns 1st January 2008,the date rolls over, as 32nd December 2007 is not valid.

DATE(2004;3;0)

returns 29th February 2004,the date rolls over backwords, as 0th March 2004 is not valid,2004 was a leap year.

DATE(2006;15;8)

returns 8th March 2007,the date rolls over,as there are only 12 months in a year

Day

returns the day of date as a number(1-31).

Syntax:

Day(date)

Date may be text or a date-time serial number.

Example:

Day("2008-06-04")

returns 4.

TIME

returns the time, given hours, minutes and seconds.

Syntax:

TIME(hours; minutes; seconds)

returns the time, expressed as a date-time serial number.

hours, **minutes** and **seconds** are integers.

If **hours**, **minutes** and **seconds** are not within range for a valid time, the time will 'roll over', as shown below.

Example:

TIME(9; 31; 20)

returns the time 9:31:20 am (as a date-time serial number).

TIME(9; 31; 75)

returns 9:32:15 am - the time rolls over, as there are only 60 seconds in a minute.

NOW

Returns the current date and time

Syntax:

NOW()

Returns the current date and time (as a date-time serial number). **NOW** is updated at every recalculation, for instance if a cell is modified.

Example:

NOW()

when calculated at say 12 noon on 1Apr08 returns that date and time.

HOUR

Returns the hour of a given time.

Syntax:

HOUR(time)

returns the hour of **time** as a number, **0 - 23**.

time may be text or a date-time serial number.

Example:

HOUR("2008-01-06 21:30:15")

returns **21**.

HOUR(A1)

where cell A1 contains the time **9:25:10** as a date-time serial number, returns **9**.

MINUTE

Returns the minutes of a given time.

Syntax:

MINUTE(time)

returns the minutes of **time** as a number, **0 - 59**.

time may be text or a date-time serial number.

Example:

MINUTE("2008-01-06 21:30:15")

returns **30**.

MINUTE(A1)

where cell A1 contains the time **9:25:10** as a date-time serial number, returns **25**.

SECOND

Returns the seconds of a given time.

Syntax:

SECOND(time)

returns the seconds of **time** as a number, **0 - 59**.

time may be text or a date-time serial number.

Example:

SECOND("2008-01-06 21:30:15")

returns **15**.

SECOND(A1)

where cell A1 contains the time **9:25:10** as a date-time serial number, returns **10**.

MONTH

Returns the month of a given date.

Syntax:

MONTH(date)

returns the month of **date** as a number, where January is **1** and December is **12**.

date may be text or a date-time serial number.

Example:

MONTH("2008-06-04")

returns **6**.

MONTH(A1)

where cell A1 contains the date **23Nov83** as a date-time serial number, returns **11**.

DAYS360

Returns the number of days between two dates, using the 360 day year.

Syntax:

DAYS360(enddate; startdate; method)

startdate and **enddate** are the starting and ending dates (text or date-time serial numbers). If **startdate** is earlier than **enddate**, the result will be negative.

method is an optional parameter; if **0** or omitted, the US National Association of Securities Dealers (NASD) method of calculation is used; if **1** (or <>0) the European method of calcuation is used.

The calculation assumes that all months have 30 days, so a year (12 months) has 360 days.

See Financial date systems for more details.

Example:

DAYS360("2008-02-29"; "2008-08-31")

returns **180**, that is, 6 months of 30 days.

WEEKDAY

Returns the day of the week for a given date.

Syntax:

WEEKDAY(date; type)

returns the day of the week that **date** falls on, as a number.

The number returned depends on **type**, as follows:

day of the week	**type=1**	**type=2**	**type=3**
Sunday	1	7	6
Monday	2	1	0

Tuesday	3	2	1
Wednesday	4	3	2
Thursday	5	4	3
Friday	6	5	4
Saturday	7	6	5

If **type** is omitted, it is assumed to be **1**.

Example:

WEEKDAY("2008-06-14"; 1)

returns **7**.

14Jun08 falls on a Saturday.

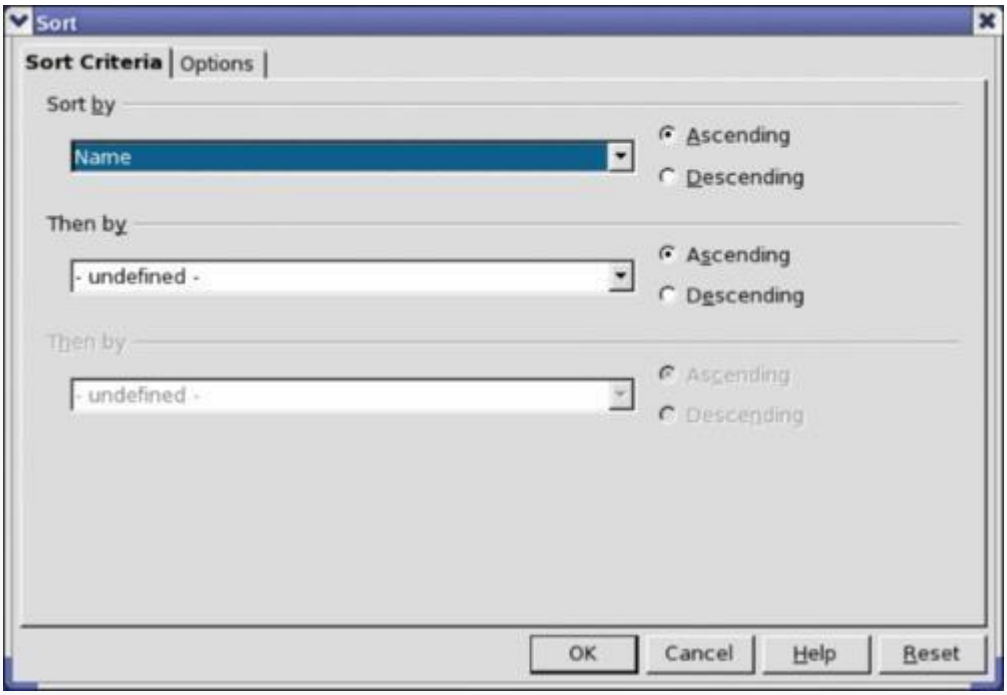
Unit 2: Formulas, Chart and Data

Data:

Sort Data, Filter data

Sort Data:

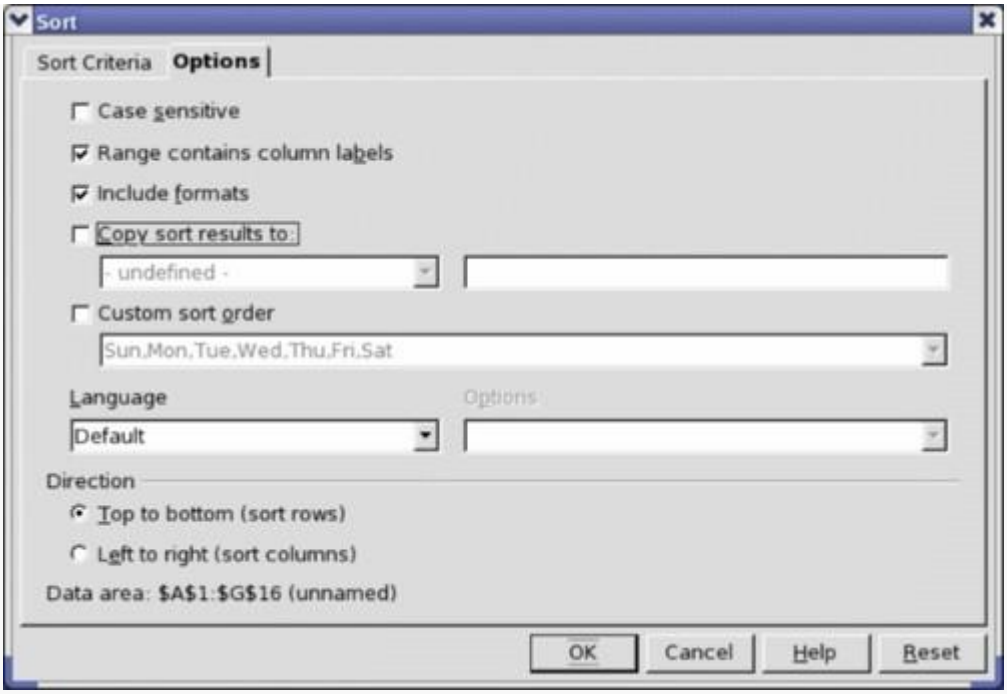
The sorting mechanism in a Calc document rearranges the data in the sheet. The first step in sorting data is to select the data that you want to sort. To sort the data, select the cells from A1 to G16—if you include the column headers, indicate this in the sort dialog. Use **Data > Sort** to open the Sort dialog. You can sort by up to three columns or rows at a time in ascending or descending order.



Sort by the Name column.

Click on the Options tab to set the sort options. Check the **Range contains column labels** checkbox to prevent column headers from being sorted with the rest of the data. The Sort by list box in the figure above displays the columns using the column headers if the **Range contains column labels** checkbox in the figure below is checked. If the **Range contains column labels** checkbox is not checked, however, then the columns are identified by their column name; Column A, for example.

Normally, sorting the data causes the existing data to be replaced by the newly sorted data. The **Copy sort results to** checkbox, however, causes the selected data to be left unchanged and a copy of the sorted data is copied to the specified location. You can either directly enter a target address (Sheet3.A1, for example) or select a predefined range.



Set sort options.

Check the **Custom sort order** checkbox to sort based on a predefined list of values. To set your own predefined lists, use **Tools > Options > OpenOffice.org Calc > Sort Lists** and then enter your own sort lists. Predefined sort lists are useful for sorting lists of data that should not be sorted alphabetically or numerically. For example, sorting days based on their name.

Filter Data:

There are three types of filter,

1) AutoFilter

2) Standard Filter

3) Advanced Filter

1) AutoFilter

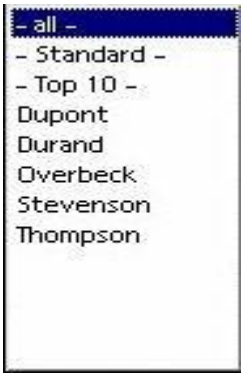
The Autofilter is slightly different from the standard filter. In order to understand what it does, let's use it and see what we get. Select the range of data, including column names.

- under **Data**, select **Filter – Autofilter**, and see what appears on the screen :

Next to each field name, a small button with an arrow has appeared. Click on the one next to the field '**Name**' to see what it does :

2)

	A	B	C	D	E
1	Name	Age	Year of Birth	Gender	Salary
2	Thompson	22	1980	Female	41250
3	Overbeck	23	1979	Female	31800
4	Dupont	25	1975	Male	26150
5	Durand	21	1981	Male	35475
6	Stevenson	20	1982	Female	38650
7					



As You will have noticed, the list of names represents the filter criteria and you can apply them differently to each column.
If you click on **Standard**, the same window appears that already explained.

Let's click on Dupont and observe the result obtained :

	A	B	C	D	E
1	Name	Age	Year of Birth	Gender	Salary
4	Dupont	25	1975	Male	26150

The criteria corresponding to the name Dupont, and only Dupont, are displayed.

To cancel the filter operation, select the range and choose **Data – Filter – Autofilter**, untick **Autofilter**

2) Standard Filter

The use of filters is as simple as sorting. After having selected your range :

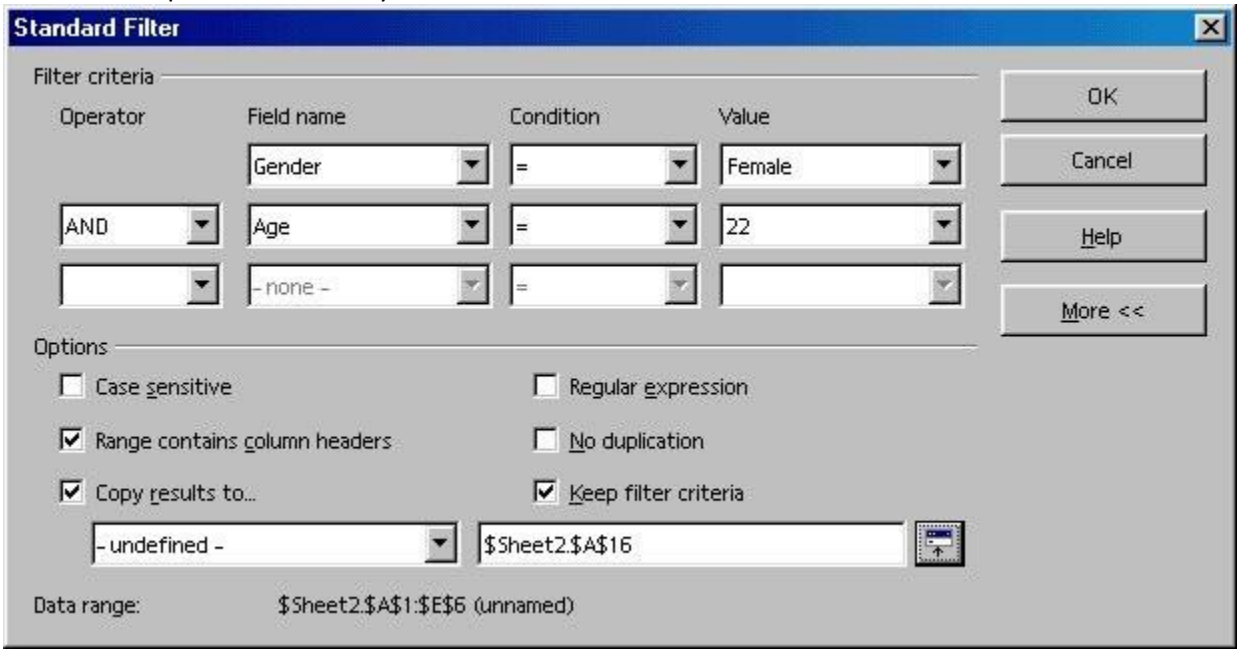
- go to the menu **Data – Filter – Standard Filter**

- The following window should appear :

- fill in the data field names on which you want to base your filter by selecting them in the drop-down menu,

- here again, you can use up to 3 criteria with Boolean operators (and, or) and other conditions such as equals, greater than, less than, less than or equal to.....The value is represented by the data on which the filter is to be carried out in relation to the field name chosen,

- the More button, enables you to edit a table, taking into account regular expressions, case sensitivity or duplicates,



- in our example, we chose to copy the result onto the same sheet, a bit lower down (using the selection icon ).

Here's the result :

16	Name	Age	Year of Birth	Gender	Salary
17	Thompson	22	1980	Female	41250
18					

3) Advanced Filter

The advanced filter is a filter that lets you use more than 3 filter criteria, up to a maximum of 8. In order to use this filter, you have to create an array in which you'll enter the criteria; but let's use an example, it makes life so much easier !

Let's start from the example we already have :

	A	B	C	D	E
1	Name	Age	Year of Birth	Gender	Salary
2	Thompson	22	1980	Female	41250
3	Overbeck	23	1979	Female	31800
4	Dupont	25	1975	Male	26150
5	Durand	21	1981	Male	35475
6	Stevenson	20	1982	Female	38650

1. copy the row bearing the field names of your range (Name, Age...) into empty cells on your sheet, for example at row 10,

11	**Name**	**Age**	**Year of Birth**	**Gender**	**Salary**
12	Dupon	<=25			>35000

2. now enter your sort criteria, under each column, bearing in mind that criteria on **a same horizontal row** are attached to the logical value by an 'AND', whereas **vertically** they are attached as 'OR'.

So, in this example, we are searching for people whose age is less than or equal to 25 AND whose salary is greater than 35000. Here are the results after applying the filter (we'll see how in a second)

17	Name	Age	Year of Birth	Gender	Salary
18	Thompson	22	1980	Female	41250
19	Durand	21	1981	Male	35475
20	Stevenson	20	1982	Female	38650
21					

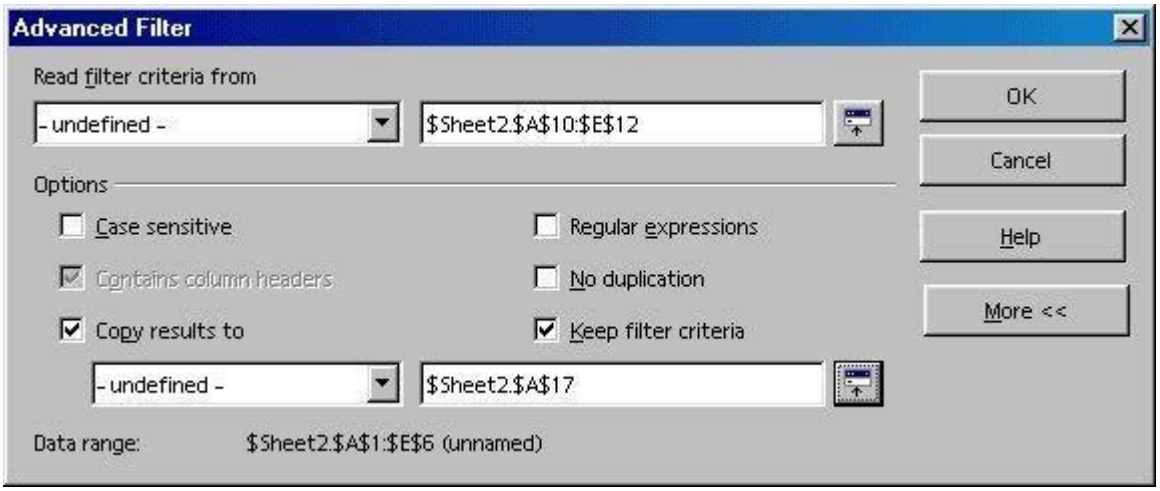
This is exactly what we asked for !
So, after having created your array

3. select the data range to which the filter should apply

4. choose **Data – Filter – Advanced Filter**,

5. in the window that appears, using the selection button, select the array that you defined at rows 10 to 12,

6. in the options (More>>), also indicate where you want your filtered data to appear
(we put it at row 17) and click on **OK**.



Text to columns, Remove DuplicationText

to columns:

To import delimited text (typically comma or tab delimited) into the spreadsheet through a file or a clipboard, Calc will parse the text and split it apart using whichever delimiter character you choose including tab, comma, space, or semicolon. A delimiter simply means a separator.

Steps and example:

Spreadsheet contains city and country names delimited by a comma:

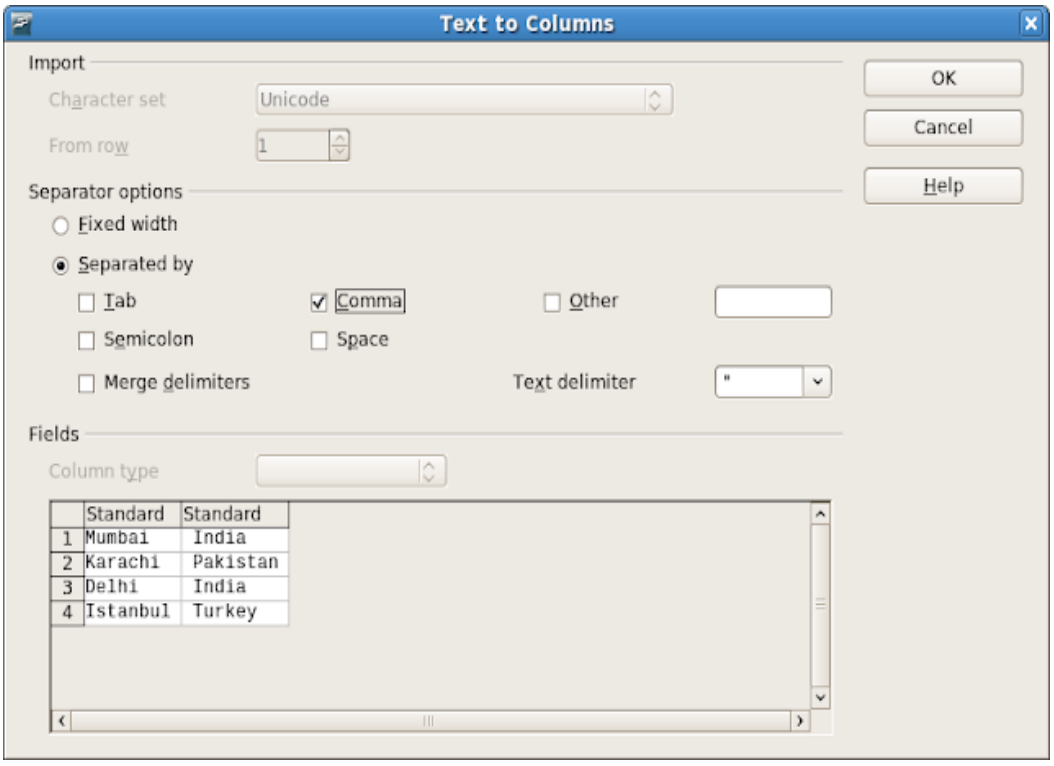
	A	B
1	Mumbai, India	
2	Karachi, Pakistan	
3	Delhi, India	
4	Istanbul, Turkey	
5		
6		

To remove the comma and put the country names into column B, we proceed as follows:

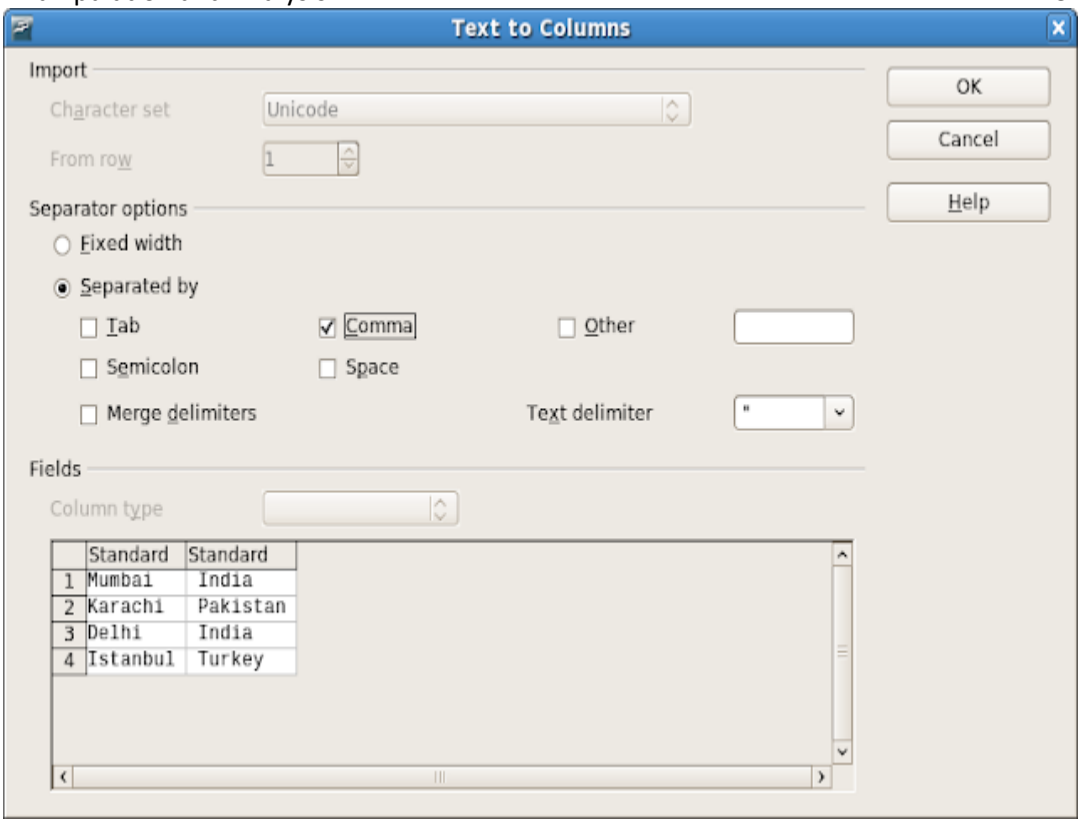
1. Make sure there are enough clear cells to accept the new values. Otherwise, the cells will be overwritten. In this case, column B must be empty because the data will grow one column to the right.

2. Highlight the full range of cells (in this case, A1 through A4).

3. Click **Data** > **Text to Columns**.



4. Uncheck the box **Tab**.
5. Check the box **Comma**.

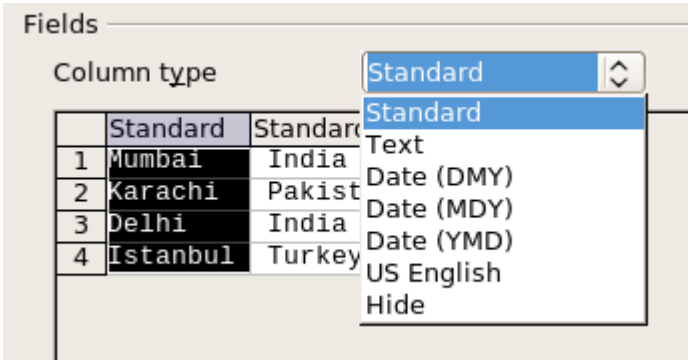


6. Click the box **OK**.
   You're done, and here are the results:

	A	B
1	Mumbai	India
2	Karachi	Pakistan
3	Delhi	India
4	Istanbul	Turkey
5		
6		

Column options

To specify the type of data in the column, in the preview area click the top of the column. Then, choose a type from the drop-down list. In many cases, this step is unnecessary.

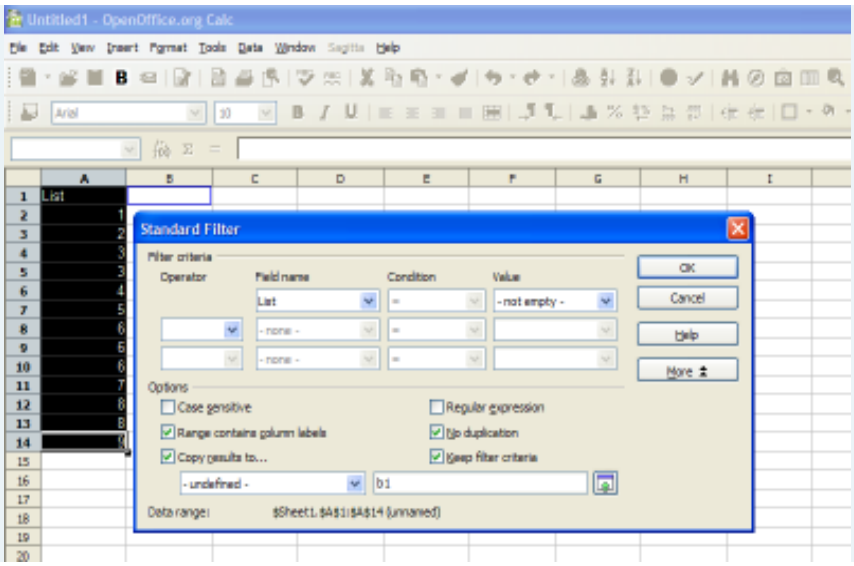


Remove Duplication:

Select the entire range containing data to filter, then click on the menu Data > Filter > Standard Filter and:

1. Use a condition that is always TRUE, like field1 = Not empty
2. Click on the button more, select Remove Duplicate, select Copy to and put the address of an empty cell

The whole range (without duplicate) will be analyzed and copied at that new address.



Consolidated Data (Sum, Count, Max, Min, Average)

Data > Consolidate provides a way to combine data from two or more ranges of cells into a new range while running one of several functions (such as Sum or Average) on the data. During consolidation, the contents of cells from several sheets can be combined into one place.

1) Open the document containing the cell ranges to be consolidated.

2) Choose **Data > Consolidate** to open the Consolidate dialog. Figure 240 shows this dialog after making the changes described below.

3) The **Source data range** list contains any existing named ranges (created using **Data > Define Range**) so you can quickly select one to consolidate with other areas.

4) If the source range is not named, click in the field to the right of thedrop-down list and either type a reference for the first source data range or use themouse to select the range on the sheet. (You may need to move the Consolidate dialog or click on the Shrink icon to reach the required cells.)

5) Click **Add**. The selected range is added to the Consolidation ranges list.

6) Select additional ranges and click **Add** after each selection.

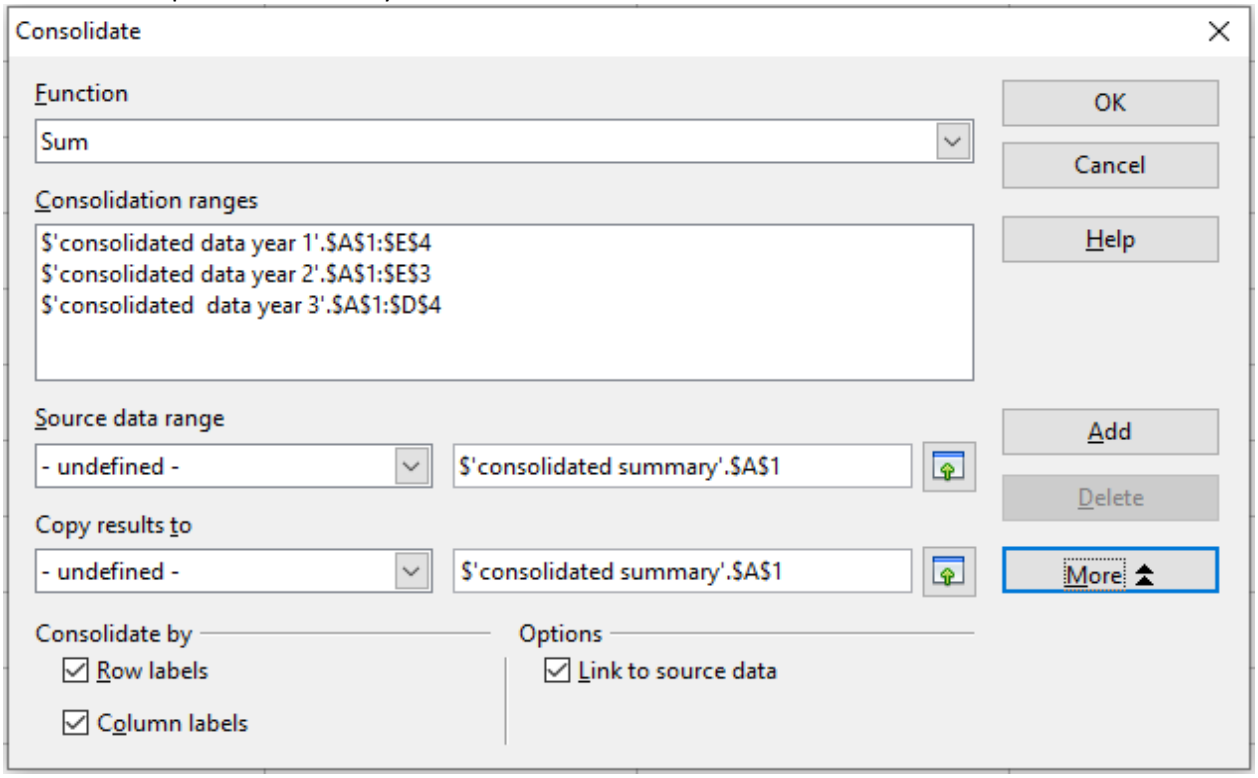


Figure : Defining the data to be consolidated

7) Specify where you want to display the result by selecting a target rangefrom the **Copy results to** drop-down list.

If the target range is not named, click in the field next to **Copy results to** and enter the reference of the target range or select the range using the mouse or position the cursor in the top left cell of the target range. *Copy results to* takes only the first cell of the target range instead of the entire range as is the case for *Source data range*.

8) Select a function from the Function list. This specifies how the values of the consolidation ranges will be calculated. The default setting is Sum, whichadds the corresponding cell values of the Source data range and gives the result in the target range.

Most of the available functions are statistical (such as Average, Min, Max, Stdev), and the tool is most useful when you are working with the same data over and over.

9) At this point you can click **More** in the Consolidate dialog to access the following additional settings:

- Select **Link to source data**,If you link the data, any values subsequently modified in the source range are automatically updated in the target range.
- Under **Consolidate by**, to consolidate by row labels or column labels, the label must be contained in the selected source ranges. The text in the labels must be identical, so that rows or columns can be accurately matched.

10) Click **OK** to consolidate the ranges.

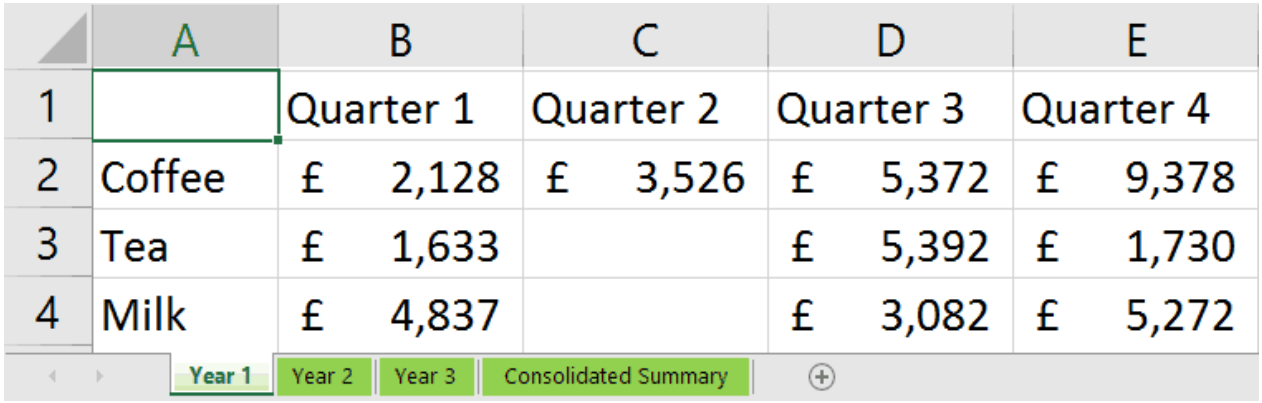
11) If you are continually working with the same range, then you probably want to use **Data** > **Define Range** to give it a name.

The consolidation ranges and target range are saved as part of the document. If you later open a document in which consolidation has been defined, this data is still available.

Example:

In our example, we have data for 3 years expenditure on tea, coffee and milk. The data is broken down into quarters and stored in one year per worksheet in one workbook. We can create a 'Consolidated Summary' sheet which will show expenditure by year and quarter. It does not matter if the data has the same arrangement of columns and rows or not. Excel will sort that out for you. Amazing!

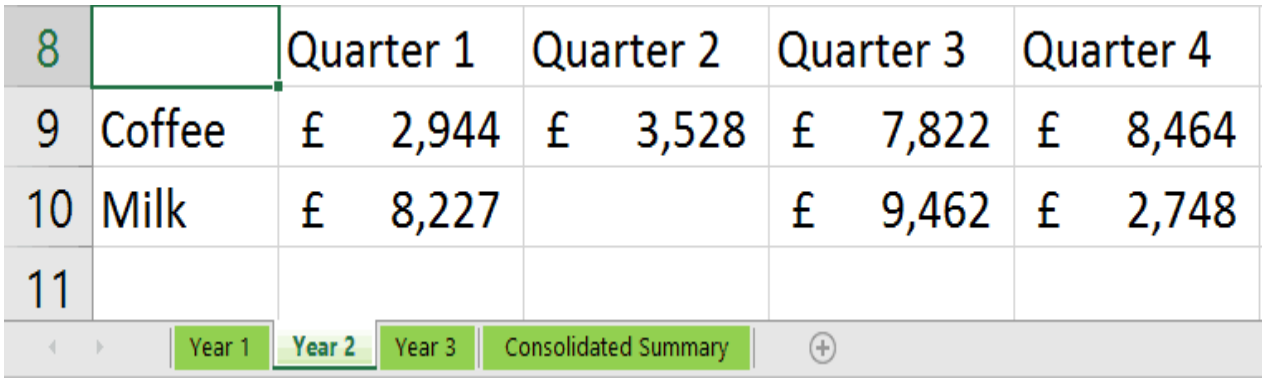
Year 1 worksheet



	A	B	C	D	E
1		Quarter 1	Quarter 2	Quarter 3	Quarter 4
2	Coffee	£ 2,128	£ 3,526	£ 5,372	£ 9,378
3	Tea	£ 1,633		£ 5,392	£ 1,730
4	Milk	£ 4,837		£ 3,082	£ 5,272

Year 1 | Year 2 | Year 3 | Consolidated Summary

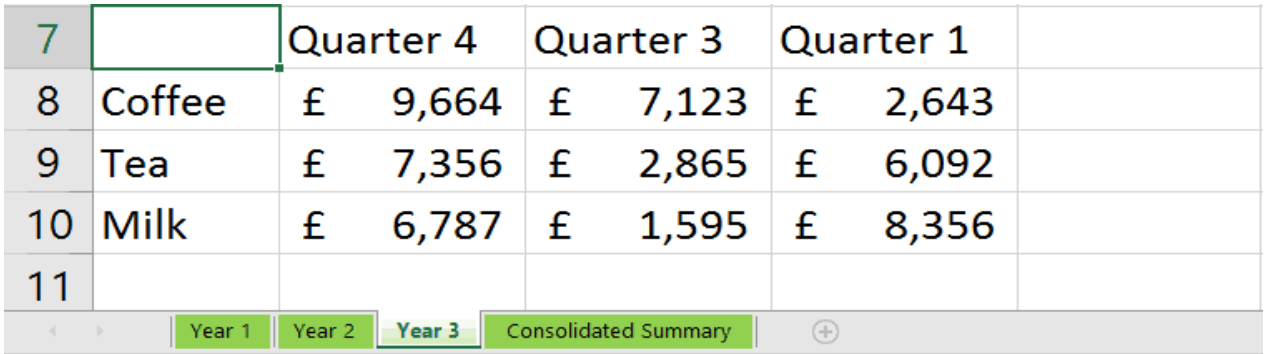
Year 2 worksheet



8		Quarter 1	Quarter 2	Quarter 3	Quarter 4
9	Coffee	£ 2,944	£ 3,528	£ 7,822	£ 8,464
10	Milk	£ 8,227		£ 9,462	£ 2,748
11					

Year 1 | Year 2 | Year 3 | Consolidated Summary

Year 3 worksheet



7		Quarter 4	Quarter 3	Quarter 1	
8	Coffee	£ 9,664	£ 7,123	£ 2,643	
9	Tea	£ 7,356	£ 2,865	£ 6,092	
10	Milk	£ 6,787	£ 1,595	£ 8,356	
11					

Year 1 | Year 2 | Year 3 | Consolidated Summary

As you can see, Years 1, 2 and 3 each have different arrangements of columns and rows. There is no tea in Year 2; in Year 3 the first quarter appears at the end of the table, there is no Quarter 2

and the Quarters are not in order. The ranges you consolidate do not have to be of the same size in each worksheet, the number of rows or columns might be different from sheet to sheet. And yet, you can still consolidate the data into a summary sheet. How incredible is that!

Consolidation steps:

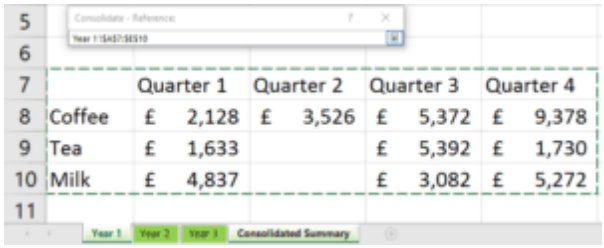
To start using the Data Consolidation tool, you need to select an empty sheet in the workbook as your master worksheet or add a new one if necessary. The worksheet is renamed 'Consolidated Summary'.

Select the upper-left cell of the area where you want the consolidated data to appear.

Choose Data > Consolidate to view the Consolidate dialog:

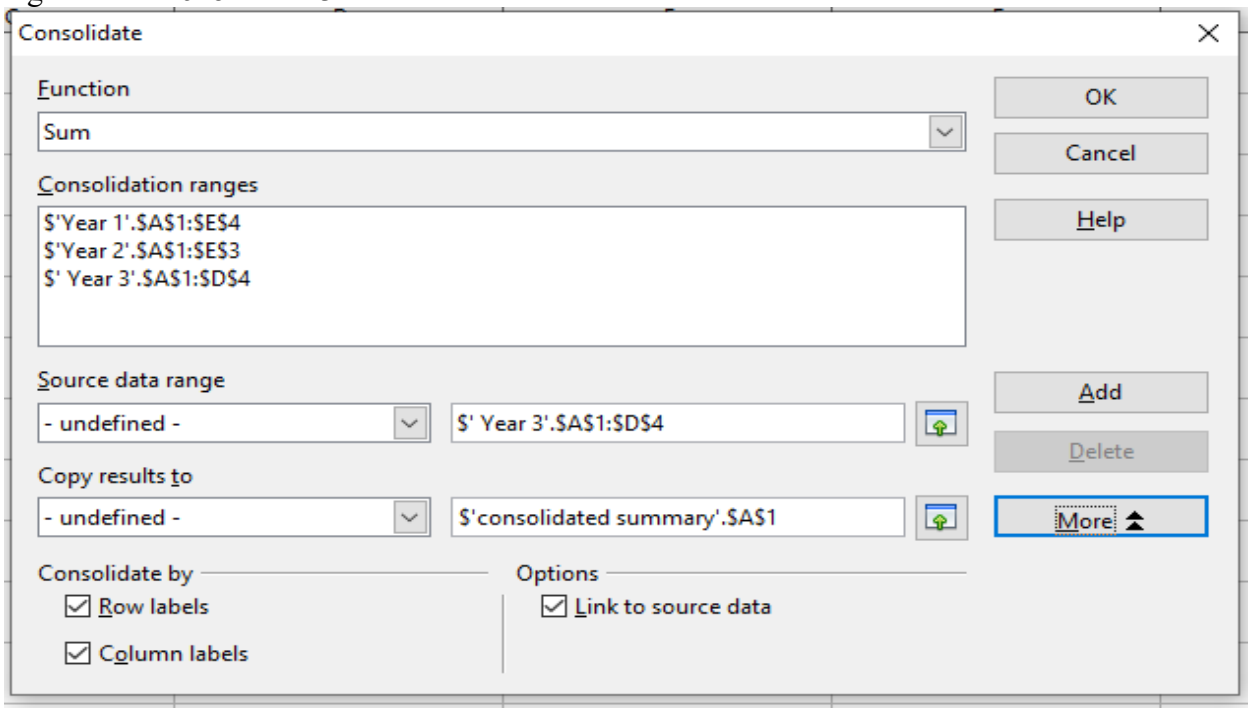
In the Function box, click the summary function that you want to use to consolidate the data. As you will see from the drop-down, there are 11 functions to choose from. For our data we want to add up the values so we'll set the Function to Sum.

Click in the Reference area and select the first data range to consolidate – to do this you will need to click the Sheet tab i.e. "Year 1" and then drag over the data (including row and column headings) and then click the Add button to add this first set of data to the consolidation dialog.



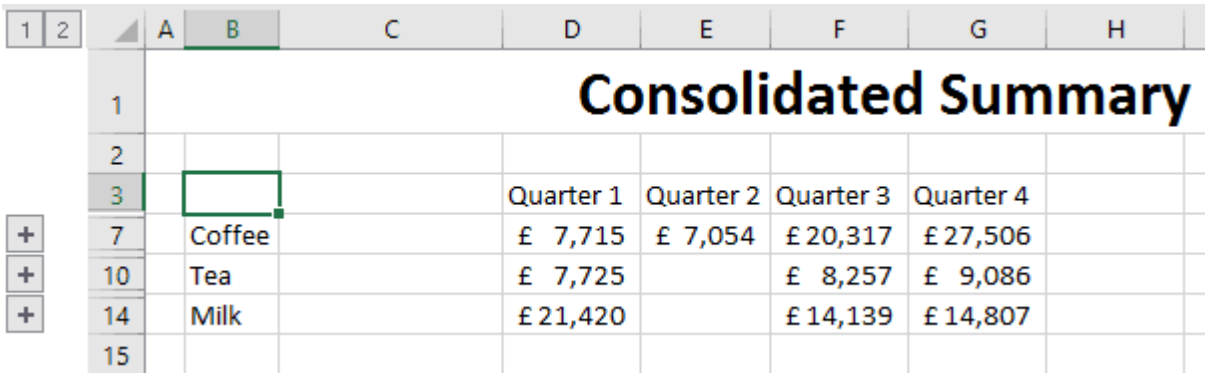
	Quarter 1	Quarter 2	Quarter 3	Quarter 4
Coffee	£ 2,128	£ 3,526	£ 5,372	£ 9,378
Tea	£ 1,633		£ 5,392	£ 1,730
Milk	£ 4,837		£ 3,082	£ 5,272

Continue in the same way by clicking on the next sheet, highlighting the data, and clicking on the Add button until all your data and worksheets appear in the References section of the dialog e.g. "Year 2" and "Year 3".



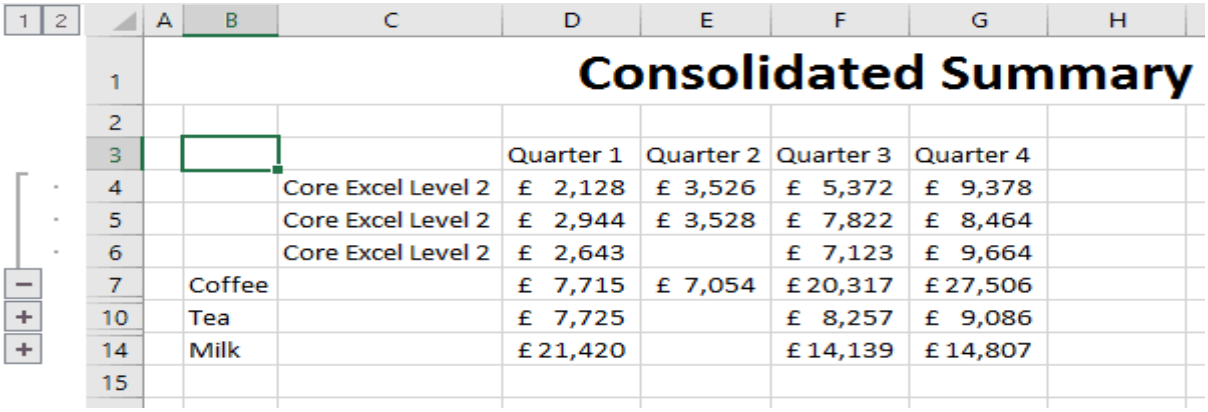
To indicate where the labels are located in the source ranges, select the check boxes under Use labels in: either the Top row, the Left column, or both. In this example, Top row is the name of the quarters, i.e. Quarter 1, Quarter 2, etc. and the Left Column are the list of items, i.e. Coffee, Tea and Milk.
Automatic vs. Manual updates: If you want Excel to update your consolidation table automatically when the source data changes, select the Create links to source data check box. If unchecked, you can still update the consolidation manually.

When you click OK, Excel summarises all the data into your new sheet as your master worksheet (Consolidated Summary).



		A	B	C	D	E	F	G	H
	1		Consolidated Summary						
	2								
	3				Quarter 1	Quarter 2	Quarter 3	Quarter 4	
+	7		Coffee		£ 7,715	£ 7,054	£ 20,317	£ 27,506	
+	10		Tea		£ 7,725		£ 8,257	£ 9,086	
+	14		Milk		£ 21,420		£ 14,139	£ 14,807	
	15								

You'll immediately notice a change to the Excel worksheet that you may never have seen before. You will see grouping tools down the left of the screen which you can use to display and hide the data. Next to rows 7, 10 and 14, there are plus signs. This signifies that cells are part of a group that is currently collapsed. Clicking on the plus sign will expand the group and there is a line Connecting these rows to the left:



		A	B	C	D	E	F	G	H
	1		Consolidated Summary						
	2								
	3				Quarter 1	Quarter 2	Quarter 3	Quarter 4	
	4			Core Excel Level 2	£ 2,128	£ 3,526	£ 5,372	£ 9,378	
	5			Core Excel Level 2	£ 2,944	£ 3,528	£ 7,822	£ 8,464	
	6			Core Excel Level 2	£ 2,643		£ 7,123	£ 9,664	
−	7		Coffee		£ 7,715	£ 7,054	£ 20,317	£ 27,506	
+	10		Tea		£ 7,725		£ 8,257	£ 9,086	
+	14		Milk		£ 21,420		£ 14,139	£ 14,807	
	15								

You'll find that the second column (Column C) of data shows the name of the workbook (Core Excel Level 2) that contains the data. You can hide this column if you want to, by right clicking it and choosing Hide. This simply hides the column so the data is there should you need to refer.

Unit 3: Concepts of Database

DATA: Data is a raw material that can be processed for any computing machine. For example: student name, marks of student, any number, image.

INFORMATION: It is the data that has been converted into more useful form. For example: Report card of student.

A database-management system (DBMS) is a collection of interrelated data and a set of programs to access those data. The collection of data, usually referred to as the database, contains information relevant to an enterprise. The primary goal of a DBMS is to provide a way to store and retrieve database information that is both *convenient* and *efficient.*

Database characteristics:

A modern DBMS has the following characteristics −

- **Real-world entity** − A modern DBMS is more realistic and uses real-world entities to design its architecture. It uses the behavior and attributes too. For example, a school database may use students as an entity and their age as an attribute.
- **Relation-based tables** − DBMS allows entities and relations among them to form tables. A user can understand the architecture of a database just by looking at the table names.
- **Isolation of data and application** − A database system is entirely different than its data. A database is an active entity, whereas data is said to be passive, on which the database works and organizes. DBMS also stores metadata, which is data about data, to ease its own process.
- **Less redundancy** − In the database approach, ideally, each data item is stored in only one place in the database. In some cases, data redundancy still exists to improve system performance, but such redundancy is controlled by application programming and kept to minimum by introducing as little redundancy as possible when designing the database.
- **Query Language** − DBMS is equipped with query language, which makes it more efficient to retrieve and manipulate data. A user can apply as many and as different filtering options as required to retrieve a set of data. Traditionally it was not possible where file-processing system was used.
- **ACID Properties** − DBMS follows the concepts of **A**tomicity, **C**onsistency, **I**solation, and **D**urability (normally shortened as ACID).
    - **Atomicity**
      either the entire transaction takes place at once or doesn't happen at all.
    - **Consistency**

      This means that integrity constraints must be maintained so that the database is consistent before and after the transaction. It refers to the correctness of a database.
    - **Isolation**

      This property ensures that multiple transactions can occur concurrently without leading to the inconsistency of database state. Transactions occur independently without interference.

- **Durability**

  This property ensures that once the transaction has completed execution, the updates and modifications to the database are stored in and written to disk and they persist even if a system failure occurs.

These concepts are applied on transactions, which manipulate data in a database. ACID properties help the database stay healthy in multi-transactional environments and in case of failure.

- **Multiuser and Concurrent Access** − DBMS supports multi-user environment and allows them to access and manipulate data in parallel. Though there are restrictions on transactions when users attempt to handle the same data item, but users are always unaware of them.
- **Multiple views** − DBMS offers multiple views for different users. A user who is in the Sales department will have a different view of database than a person working in the Production department. This feature enables the users to have a concentrate view of the database according to their requirements.

- **Data sharing**-The integration of all the data, for an organization, within a database system has many advantages. First, it allows for data sharing among employees and others who have access to the system. Second, it gives users the ability to generate more information from a given amount of data than would be possible without the integration.
- **Security** − Features like multiple views offer security to some extent where users are unable to access data of other users and departments. DBMS offers methods to impose constraints while entering data into the database and retrieving the same at a later stage. DBMS offers many different levels of security features, which enables multiple users to have different views with different features. For example, a user in the Sales department cannot see the data that belongs to the Purchase department. Additionally, it can also be managed how much data of the Sales department should be displayed to the user.

Data Independence (Logical and Physical)

Instance: The collection of information stored in the database at a particular moment is called an instance of the database.
Schema: The overall design of the database is called the database schema. Database systems have several schemas, partitioned according to the levels of abstraction.

- **Physical Database Schema** − The physical schema describes the database design at the physical level. This schema pertains to the actual storage of data and its form of storage like files, indices(index), etc. It defines how the data will be stored in a secondary storage.
- **Logical Database Schema** − The logical schema describes the database design at the logical level. This schema defines all the logical constraints that need to be applied on the data stored. It defines tables, views, and integrity constraints.
- **Subschemas-**A database may also have several schemas at the view level, sometimes called subschemas, that describe different views of the database.

Data Independence

Data Independence is defined as a property of DBMS that helps you to change the Database schema at one level of a database system without requiring to change the schema at the next higher level. Data independence helps you to keep data separated from all programs that make use of it.

- **Types of Data Independence**

In DBMS there are two types of data independence

1. Physical data independence
2. Logical data independence.

- **Levels of Database**

Before we learn Data Independence, a refresher on Database Levels is important. The database has 3 levels as shown in the diagram below

1. Physical/Internal
2. Conceptual/Logical
3. External/View

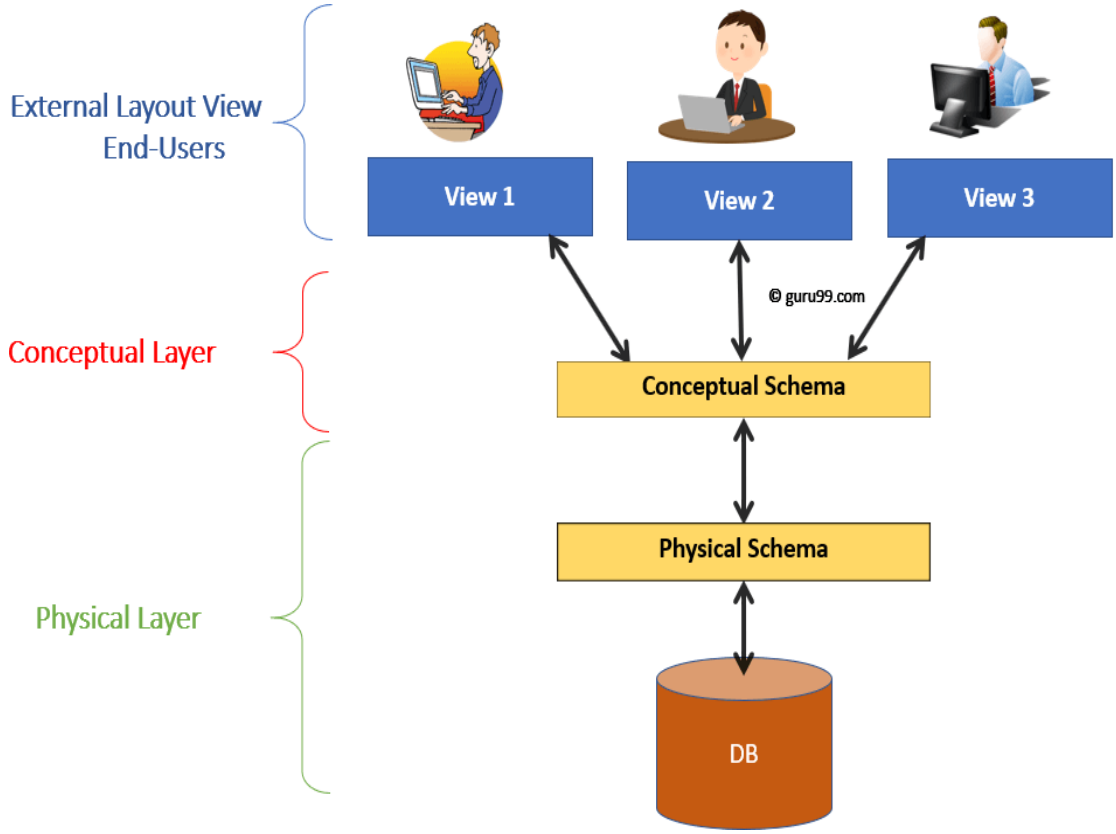


Figure 3.1: Levels of DBMS Architecture Diagram

1. **Physical/Internal level:** It describes how a record (e.g., customer) is stored.

   Features:

   a) Lowest level of abstraction.
   b) It describes how data are actually stored.
   c) It describes low-level complex data structures in detail.

d) At this level, efficient algorithms to access data are defined.

2. **Conceptual/Logical level:** It describes what data stored in database, and the relationships among the data.

   Features:

   a) It is next-higher level of abstraction. Here whole Database is divided into small simple structures.
   b) Users at this level need not be aware of the physical-level complexity used to implement the simple structures.
   c) Generally, database administrators (DBAs) work at logical level of abstraction.

3. **External/View level:** Application programs hide details of data types. Views can also hide information (e.g., salary) for security purposes.

   Features:

   a) It is the highest level of abstraction.
   b) It describes only a part of the whole Database for particular group of users.
   c) This view hides all complexity.
   d) It exists only to simplify user interaction with system.
   e) The system may provide many views for the whole system.

Consider an Example of a University Database. At the different levels this is how the implementation will look like:

Type of Schema	Implementation
External Schema	**View 1**: Course info(cid:int,cname:string) **View 2**: studentinfo(id:int. name:string)
Conceptual Shema	Students(id: int, name: string, login: string, age: integer) Courses(id: int, cname.string, credits:integer) Enrolled(id: int, grade:string)
Physical Schema	• Relations stored as unordered files. • Index on the first column of Students.

- **Physical Data Independence**

Physical Data Independence is defined as the ability to make changes in the structure of the lowest level of the Database Management System (DBMS) without affecting the higher-level schemas. Hence, modification in the Physical level should not result in any changes in the Logical or View levels.

Example –
Changes in the lowest level (physical level) are: creating a new file, storing the new files in the system, creating a new index etc.
Instances of why we may want to do any sort of Data modification in the physical level- We may want to alter or change the data in the physical level. This is because we may want to add or remove files and indexes to enhance the performance of the database system and make it faster. Hence, in this way, the Physical Data Independence enables us to do Performance Tuning. Ideally, when we change the physical level, we would not want to alter the logical and view level.

- **Logical Data Independence**

Logical Data Independence is defined as the ability to make changes in the structure of the middle level of the Database Management System (DBMS) without affecting the highest-level schema or application programs. Hence, modification in the logical level should not result in any changes in the view levels or application programs.

Example –
Due to Logical independence, any of the below change will not affect the external layer.

1. Add/Modify/Delete a new attribute, entity or relationship is possible without a rewrite of existing application programs
2. Merging two records into one
3. Breaking an existing record into two or more records

Difference between Physical and Logical Data Independence

Logica Data Independence	Physical Data Independence
Logical Data Independence is mainly concerned with the structure or changing the data definition.	Mainly concerned with the storage of the data.
It is difficult as the retrieving of data is mainly dependent on the logical structure of data.	It is easy to retrieve.
it is difficult to achieve logical data independence.	Compared to Logical Independence it is easy to achieve physical data independence.

You need to make changes in the Application program if new fields are added or deleted from the database.	A change in the physical level usually does not need change at the Application program level.
Modification at the logical levels is significant whenever the logical structures of the database are changed.	Modifications made at the internal levels may or may not be needed to improve the performance of the structure.
Concerned with conceptual schema	Concerned with internal(physical) schema
Example: Add/Modify/Delete a new attribute	Example: change in compression techniques, hashing algorithms, storage devices, etc

Importance of Data Independence

- Helps you to improve the quality of the data
- Database system maintenance becomes affordable
- Enforcement of standards and improvement in database security
- You don't need to alter data structure in application programs
- Permit developers to focus on the general structure of the Database rather than worrying about the internal implementation
- Easily make modifications in the physical level is needed to improve the performance of the system.

Components of Database (User, Application, DBMS, Database)

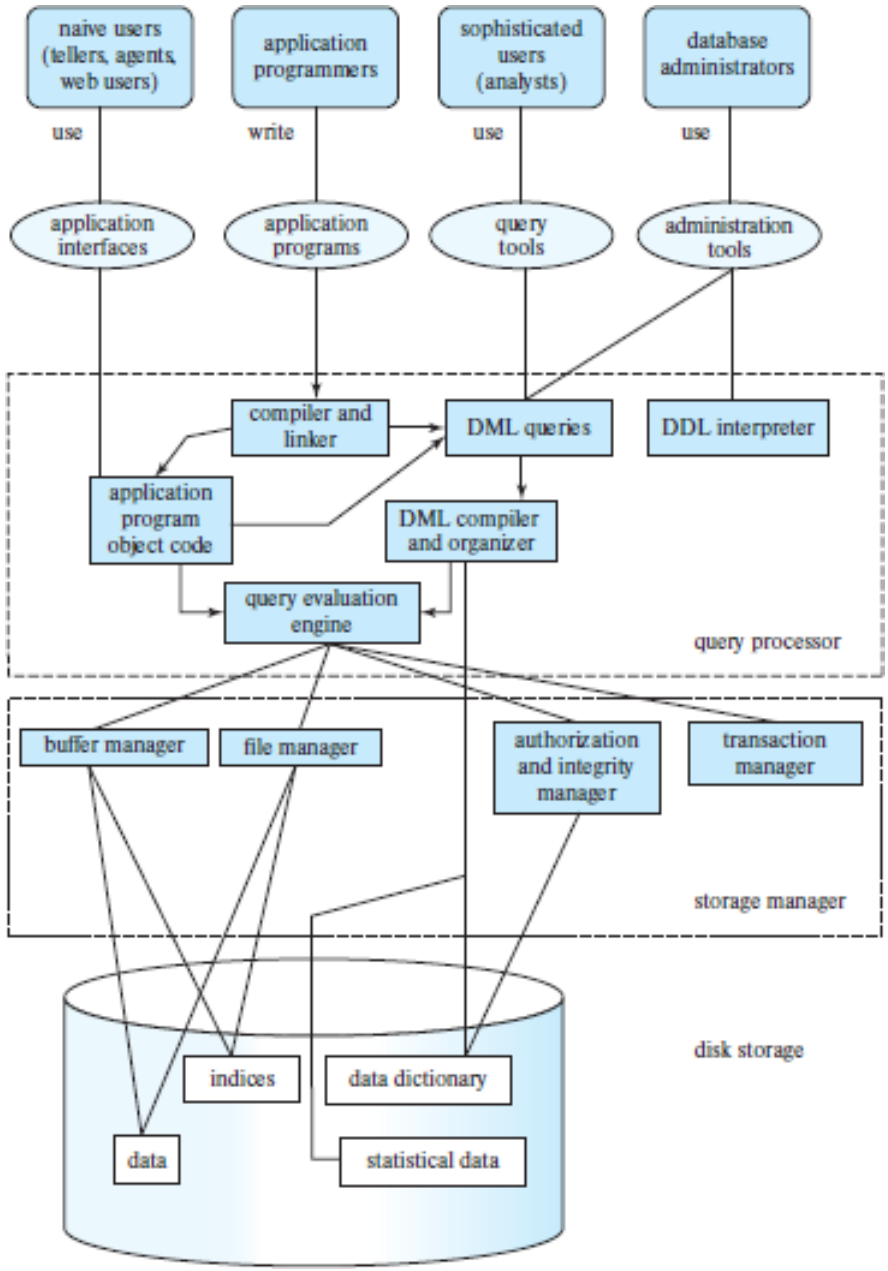


Figure 3.2: System structure

➢ Database Users and Administrators

Database Users

There are four different types of database-system users, differentiated by the way they expect to interact with the system. Different types of user interfaces have been designed for the different types of users.

• **Naive users** are unsophisticated users who interact with the system by using predefined user interfaces, such as web or mobile applications. The typical user interface for naive users is a forms

interface, where the user can fill in appropriate fields of the form. Naive users may also view read *reports* generated from the database.

• **Application programmers** are computer professionals who write application programs. Application programmers can choose from many tools to develop user interfaces.

• **Sophisticated users** interact with the system without writing programs. Instead, they form their requests either using a database query language or by using tools such as data analysis software. Analysts who submit queries to explore data in the database fall in this category.

Database Administrator

One of the main reasons for using DBMSs is to have central control of both the data and the programs that access those data. A person who has such central control over the system is called a database administrator (DBA).

The functions of a DBA include:
• **Schema definition.** The DBA creates the original database schema by executing a set of data definition statements in the DDL.

• **Storage structure and access-method definition.** The DBA may specify some parameters pertaining to the physical organization of the data and the indices to be created.

• **Schema and physical-organization modification.** The DBA carries out changes to the schema and physical organization to reflect the changing needs of the organization,or to alter the physical organization to improve performance

• **Granting of authorization for data access.** By granting different types of authorization,the database administrator can regulate which parts of the database various users can access. The authorization information is kept in a special system structure that the database system consults whenever a user tries to access the data in the system.

• **Routine maintenance.** Examples of the database administrator's routine maintenance activities are:
- Periodically backing up the database onto remote servers, to prevent loss of data in case of disasters such as flooding.
- Ensuring that enough free disk space is available for normal operations, and upgrading disk space as required.
- Monitoring jobs running on the database and ensuring that performance is not degraded by very expensive tasks submitted by some users.

➢ Query processor:

Database Languages
A database system provides a data-definition language (DDL) to specify the database schema and a data-manipulation language (DML) to express database queries and updates.

Data-Definition Language
We specify a database schema by a set of definitions expressed by a special language called a data-definition language (DDL). The DDL is also used to specify additional properties of the data.

We specify the storage structure and access methods used by the database system by a set of statements in a special type of DDL called a data storage and definition language. These statements define the implementation details of the database schemas, which are usually hidden from the users.

Data-Manipulation Language

A data-manipulation language (DML) is a language that enables users to access or manipulate data as organized by the appropriate data model. The types of access are:

- Retrieval of information stored in the database.
- Insertion of new information into the database.
- Deletion of information from the database.
- Modification of information stored in the database.

There are basically two types of data-manipulation language:

- **Procedural DMLs** require a user to specify *what* data are needed and *how* to get those data.

- **Declarative DMLs** (also referred to as nonprocedural DMLs) require a user to specify *what* data are needed *without* specifying how to get those data.

A query is a statement requesting the retrieval of information. The portion of a DML that involves information retrieval is called a query language. Although technically incorrect, it is common practice to use the terms *query language* and *data-manipulation language* synonymously.

The query processor is important because it helps the database system to simplify and facilitate access to data. The query processor allows database users to obtain good performance while being able to work at the view level and not be burdened with understanding the physical-level details of the implementation of the system. It is the job of the database system to translate updates and queries written in a nonprocedural language, at the logical level, into an efficient sequence of operations at the physical level.

The **query processor components** include:

- **DDL interpreter,** which interprets DDL statements and records the definitions in the data dictionary.

- **DML compiler,** which translates DML statements in a query language into an evaluation plan consisting of low-level instructions that the query-evaluation engine understands.

A query can usually be translated into any of a number of alternative evaluation plans that all give the same result. The DML compiler also performs query optimization; that is, it picks the lowest cost evaluation plan from among the alternatives.

- **Query evaluation engine,** which executes low-level instructions generated by the DML compiler.

➢ Storage Manager

The storage manager is the component of a database system that provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system. The storage manager is responsible for the interaction with the file manager. The raw data are stored on the disk using the file system provided by the operating system. The storage manager translates the various DML statements into low-level file-system commands. Thus, the storage manager is responsible for storing, retrieving, and updating data in the database.

The **storage manager components** include:
• **Authorization and integrity manager,** which tests for the satisfaction of integrity constraints and checks the authority of users to access data.

• **Transaction manager,** which ensures that the database remains in a consistent (correct) state despite system failures, and that concurrent transaction executions proceed without conflicts.

• **File manager,** which manages the allocation of space on disk storage and the data structures used to represent information stored on disk.

• **Buffer manager,** which is responsible for fetching data from disk storage into main memory, and deciding what data to cache in main memory. The buffer manager is a critical part of the database system, since it enables the database to handle data sizes that are much larger than the size of main memory.

The storage manager implements several **data structures** as part of the physical system implementation:

• **Data files,** which store the database itself.

• **Data dictionary,** which stores metadata about the structure of the database, in particular the schema of the database.

• **Indices,** which can provide fast access to data items. Like the index in this textbook, a database index provides pointers to those data items that hold a particular value.

For example, we could use an index to find the *instructor* record with a particular *ID*, or all *instructor* records with a particular *name*.

Database Architecture (1-tier, 2-tier, 3-tier)

The DBMS design depends upon its architecture. The basic client/server architecture is used to deal with a large number of PCs, web servers, database servers and other components that are connected with networks.

The client/server architecture consists of many PCs and a workstation which are connected via the network. DBMS architecture depends upon how users are connected to the database to get their request done.

Types of DBMS Architecture

1 Tier => The Client, Server and Database resides on the same machine.

2 Tier => The client on one machine and the server and database on one machine, i.e. two machines.

3 Tier => We have three different machines one for each client, server and a separate machine dedicated to database.

❖ 1-Tier Architecture

- In this architecture, the database is directly available to the user. It means the user can directly sit on the DBMS and uses it.
- Any changes done here will directly be done on the database itself. It doesn't provide a handy tool for end users.
- The 1-Tier architecture is used for development of the local application, where programmers can directly communicate with the database for the quick response.

Advantage: Fast for a single user because communication with another system is not necessary.

Disadvantage: Completely unscalable. Only one user can access the system at a given time via the local client.

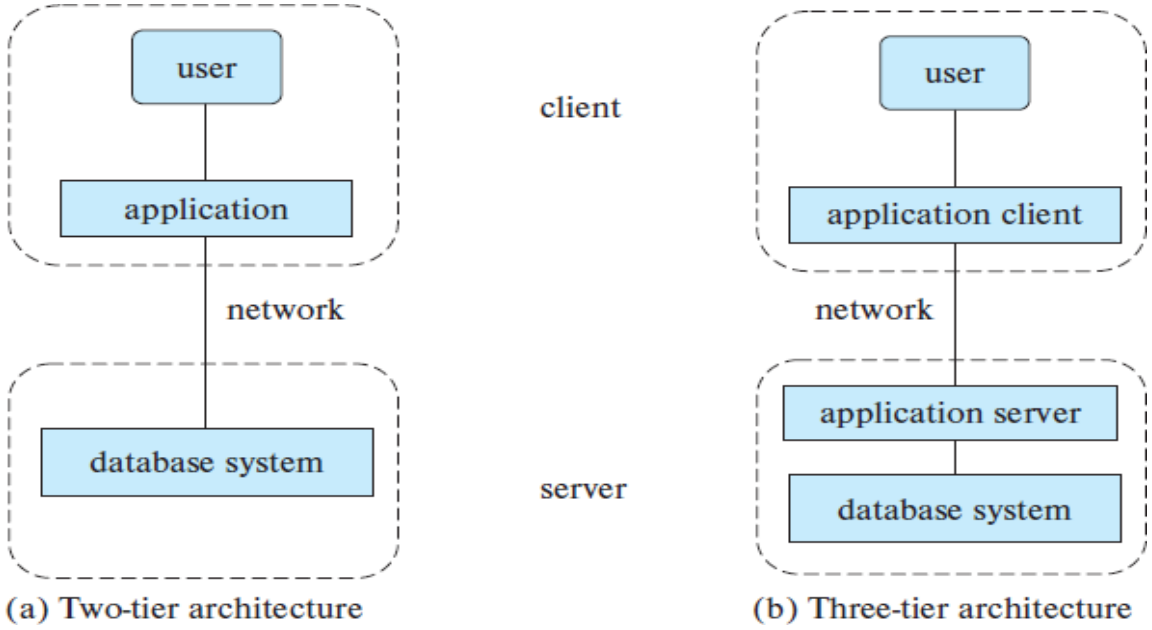
❖ 2-Tier Architecture

- Earlier-generation database applications used a two-tier architecture, where the application resides at the client machine, and invokes database system functionality at the server machine through query language statements.
- The 2-Tier architecture is same as basic client-server. In the two-tier architecture, applications on the client end can directly communicate with the database at the server side. For this interaction, API's like: ODBC**,** JDBC are used.
- The user interfaces and application programs are run on the client-side.
- The server side is responsible to provide the functionalities like: query processing and transaction management.
- To communicate with the DBMS, client-side application establishes a connection with the server side.

Advantages:

An advantage of this type is that maintenance and understanding is easier, compatible with existing systems.

Disadvantages:
However this model gives poor performance when there are a large number of users.



(a) Two-tier architecture (b) Three-tier architecture

❖ 3-Tier Architecture

- In contrast, modern database applications use a three-tier architecture, where the client machine acts as merely a front end and does not contain any direct database calls;web browsers and mobile applications are the most commonly used application clients today.
- The front end communicates with an application server. The application server,in turn, communicates with a database system to access data.
- The business logic of the application, which says what actions to carry out under what conditions, is embedded in the application server, instead of being distributed across multiple clients.
- Three tier applications provide better security as well as better performance than two-tier applications .
- The 3-Tier architecture contains another layer between the client and server. In this architecture, client can't directly communicate with the server.
- The application on the client-end interacts with an application server which further communicates with the database system.
- End user has no idea about the existence of the database beyond the application server. The database also has no idea about any other user beyond the application.
- The 3-Tier architecture is used in case of large web application.

Advantages:

- **Enhanced scalability** due to distributed deployment of application servers. Now, individual connections need not be made between client and server.
- **Data Integrity** is maintained. Since there is a middle layer between client and server, data corruption can be avoided/removed.

- **Security** is improved. This type of model prevents direct interaction of the client with the server thereby reducing access to unauthorized data.

Disadvantages:

Increased complexity of implementation and communication. It becomes difficult for this sort of interaction to take place due to presence of middle layers.

Unit 3: Concepts of Database

Database Models (Hierarchical, Network, E/R, Relational)

Is a collection of conceptual tools for describing data, data relationship, data semantics and constraints.Data models define how data is connected to each other and how they are processed and stored inside the system.

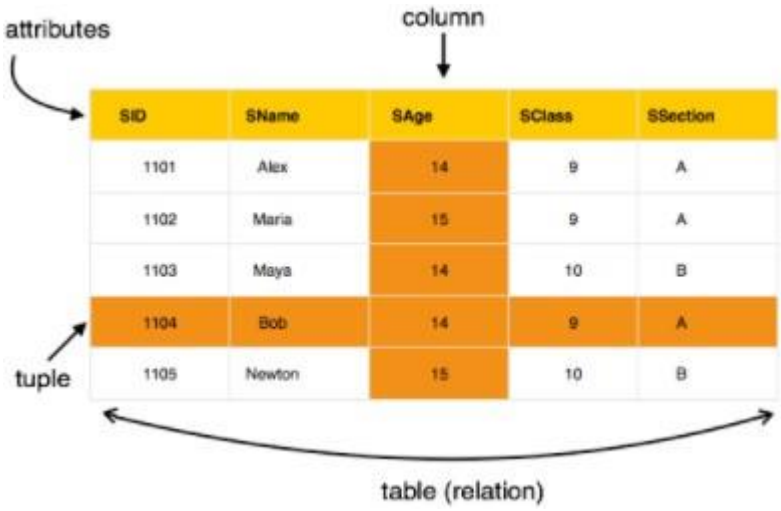
- **Three types of data models**

- Physical Model
- Record Base Logical Model
    1. Relational Model
    2. Network Model
    3. Hierarchical Model
- Object Base Logical Model
    1. Object Oriented Model
    2. Entity relationship model

- **Physical Data Models** – These models describe data at the lowest level of abstraction. The Physical data model specifies implementation details which may be features of a particular product or version as well as configuration choice for the database instance. Physical data model is a representation of a data design which takes into account the facilities and constraints of a given database management system. The physical data model can used to calculate storage estimates and may include specific storage allocation details for a given database system. This involves the actual design of a database according to the requirement that were established during logical modelling.

- **Record based Data model:**
  Record base data models are so popular because the database is structured in fixed format that is record. It describe data at the conceptual and view levels. These models specify logical structure of database with records, fields and attributes. Each record type defines a fixed number of fields or attributes. Each attributes of a fix length.

  Relational Data Model:
  Data and relationships are represented by a collection of tables. Each table has a number of columns with unique name.
  In relational model, the data and relationships are represented by collection of inter-related tables. Each table is a group of column and rows, where column represents attribute of an entity and rows represents records.



Sample relationship Model: Student table with 3 columns and four records.

Table: Student

Stu_Id	Stu_Name	Stu_Age
111	Ashish	23
123	Saurav	22
169	Lester	24
234	Lou	26

Table: Course

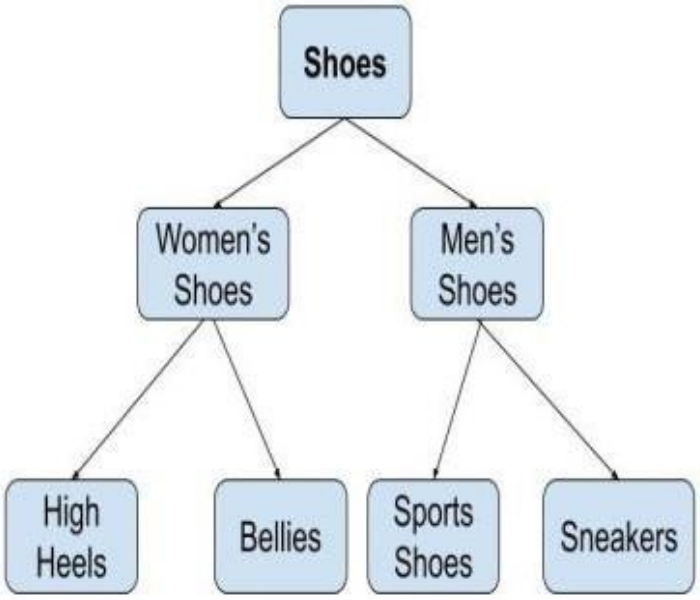
Stu_Id	Course_Id	Course_Name
111	C01	Science
111	C02	DBMS
169	C22	Java
169	C39	Computer Networks

Here Stu_Id, Stu_Name & Stu_Age are attributes of table Student and Stu_Id, Course_Id & Course_Name are attributes of table Course. The rows with values are the records (commonly known as tuples).

Hierarchical Model:

In **hierarchical model**, data is organized into a tree like structure with each record is having one parent record and many children. This model organises the data in the hierarchical tree structure. The hierarchy starts from the root which has root data and then it expands in the form of a tree adding child node to the parent node. This model easily represents some of the real-world relationships like food recipes, sitemap of a website etc.

Example: We can represent the relationship between the shoes present on a shopping website in the following way:



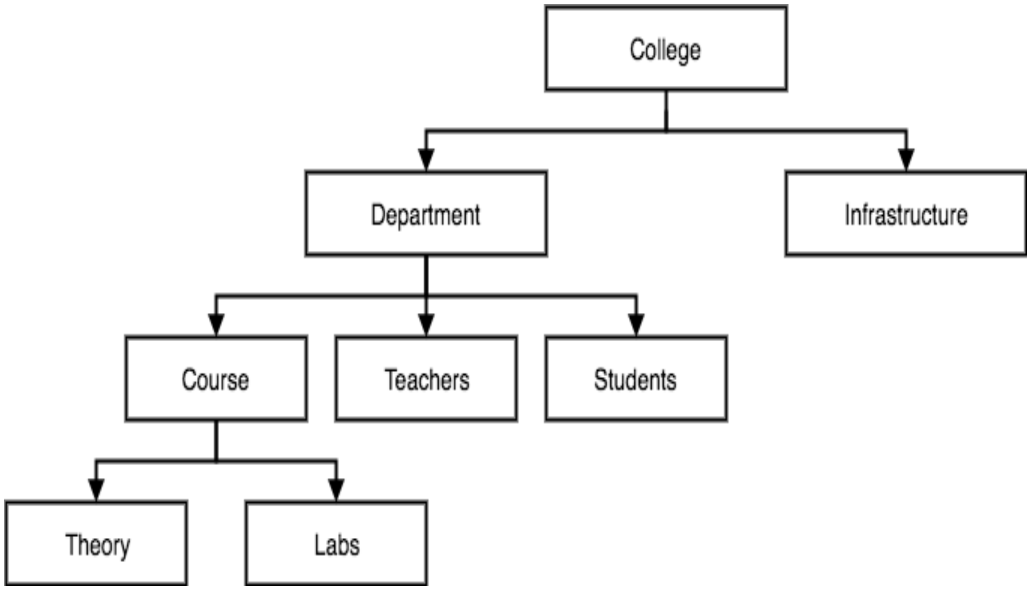
Hierarchical Model

The hierarchy starts from the **Root** data, and expands like a tree, adding child nodes to the parent nodes.

In this model, a child node will only have a single parent node.

In hierarchical model, data is organised into tree-like structure with one-to-many relationship between two different types of data.

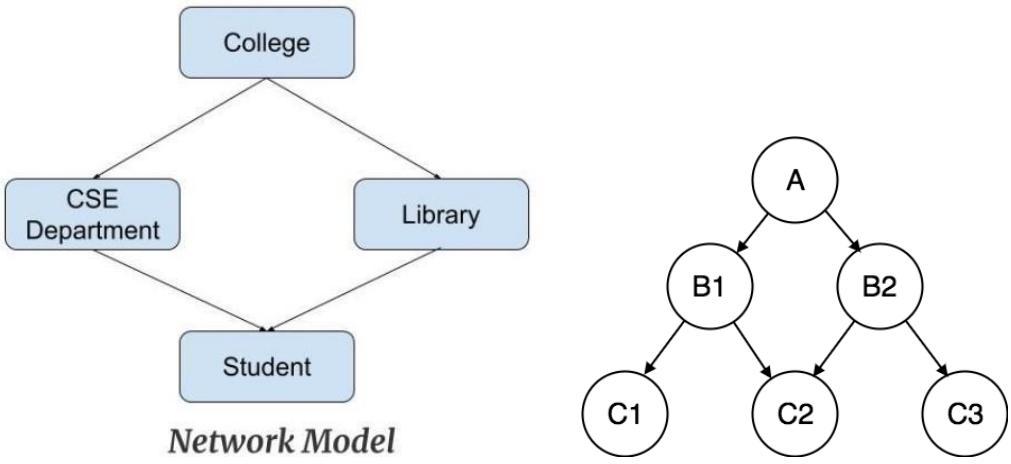
For example, one department can have many courses, many professors and of-course many students.



Network Model:

Data are represented by collection of records. Relationships among data are represented by links. This links can be viewed as a pointer. The network data model represents data for an entity set by logical record type. Network Model is same as hierarchical model except that it has graph-like structure rather than a tree-based structure. Unlike hierarchical model, this model allows each record to have more than one parent record.

Example: In the example below we can see that node student has two parents i.e. CSE Department and Library. This was earlier not possible in the hierarchical model.



Network Model

- **Object Base Logical Model**

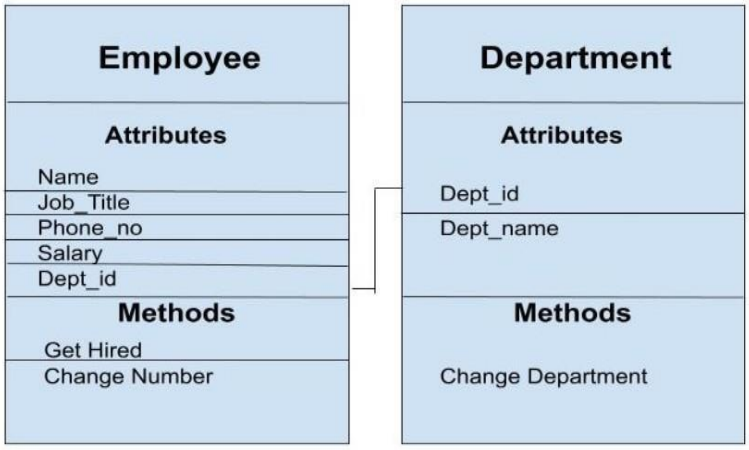
Object base Logical Model are use in describing data at the logical and view levels.

They provide flexible structuring capabilities and allow data constraints to be specifying explicitly.

- Object Oriented Model
- Entity relationship model

Object Oriented Model

This model is based on a collection of objects .An object contains values stored in instance variable within the objects. An object also contain a body of code that operate and this bodies of code are called methods. Objects that contain the same types of values and the same methods are grouped into class. A class may be view as a type definition for object.
The real-world problems are more closely represented through the object-oriented data model. In this model, both the data and relationship are present in a single structure known as an object. In this model, two are more objects are connected through links. We use this link to relate one object to other objects. This can be understood by the example given below.



Object_Oriented_Model

In the above example, we have two objects Employee and Department. All the data and relationships of each object are contained as a single unit. The attributes like Name, Job_title of the employee and the methods which will be performed by that object are stored as a single object. The two objects are connected through a common attribute i.e the Department_id and the communication between these two will be done with the help of this common id.

Entity Relationship Model

This model is based on perception of real word that consists of collection of basic object called entity & relation among this objects.

ER Model is based on

Entities and their attributes.

Relationships among entities

Entity: It is a thing or object in real world that is distinguishable from others.

- Ex Each student is an entity.
- Entities are described in a database by a set of attributes. Ex Roll No, Name and Address are attributes of student entity.

Entity Set: The set of all entities of the same type is called as entity set.

E.g. student and teacher, common properties are name, address, phone, email, etc…

Relationship: A relationship is an association among these several entities.

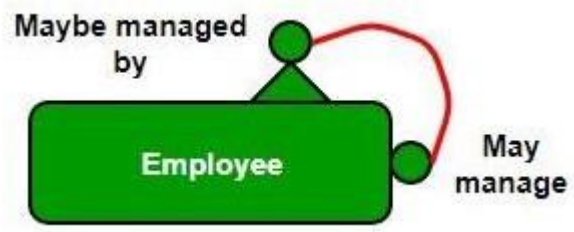
- Ex A depositor relationship associate a customer entity with A/C entity.

Relationship Set: The set of all relationship of same type is known as relationship set.

Types of relationship set:

Recursive relationship set:

The same entity set participates in a relationship set more than once then it is called recursive relationship set.



eg. Let us suppose that we have an employee table. A manager supervises a subordinate. Every employee can have a supervisor except the CEO and there can be at most one boss for each employee. One employee may be the boss of more than one employee.

Binary relationship set:

Relationship between 2 entities is called Binary relationship.

eg. The relationship sets borrower and loan branch provide an example of a binary relationship set- that is, one that involves two entity sets

Most of the relationship set in a database system are binary.

Ternary relationship set:

Relationship between 3 entities is called binary relationship.

eg. Considers the entity sets employee, branch and job.

The number of entity sets that participate in a relationship set also the degree of the relationship set.

A binary set relationship set is of degree 2 and a ternary relationship set is of degree 3.

Attribute: Attributes are properties hold by each member of an entity set.

- E.g. Entity is student and attributes of students are enrollment no, name, address etc …

Types of Attributes

- **Simple attribute:** It cannot be divided into subparts. E.g. cust_no, rollno
- **Composite attribute**: It can be divided into subparts. E.g. address
- **Single valued attribute:** It has single data value. E.g. enrollmentno, birthdate
- **Multi valued attribute**: It has multiple data value. E.g. phone no (may have multiple phones)
- **Stored attribute:** It's value is stored in database. E.g. birthdate
- **Derived attribute**: It's value is derived or calculated from other attributes. E.g. age (can be calculated using current date and birthdate)

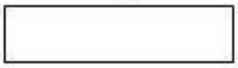
Example of entity-relationship model

The overall logical structure of a database can be expressed graphically by E-R diagram. The following components are useful for E-R diagram for

1. **Rectangle** It represents entity set.
2. **Ellipse** It represents attributes.
3. **Diamond** It represents relationship among entity set
4. **Line** It link attribute to entity set and entity set to relationship.

Symbols in ER Diagram

1. Rectangle: -
   - It represents entity set.

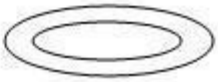


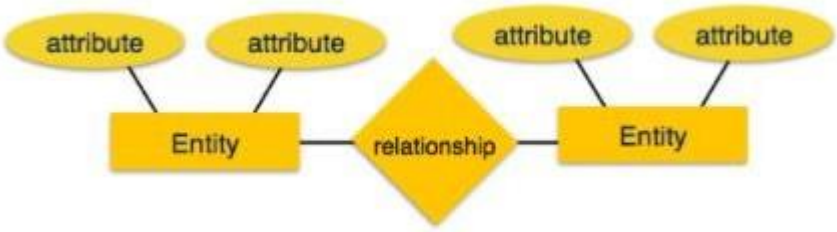
2. Ellipse: -
   - It represents attributes.

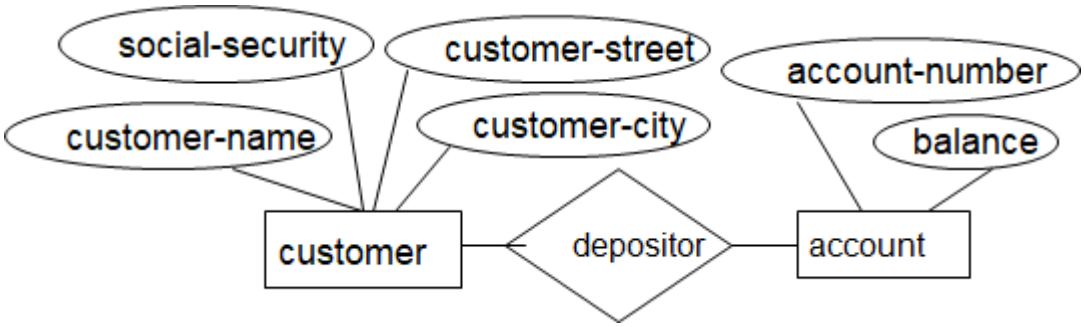
3. Diamond: -
   - It represents relationship among entity set

4. Line: -
   - It link attribute to entity set and entity set to relationship.

5. Double Elipse :
   - It represent multi valued attributes.





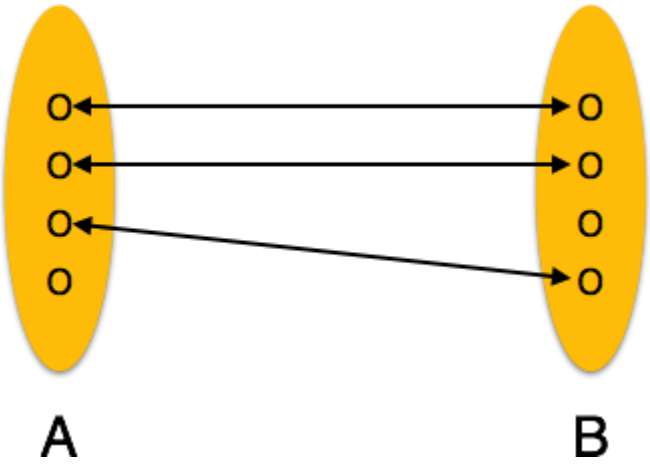


Mapping cardinalities

Mapping cardinalities define the number of association between two entities.It represents the number of entities of another entity set which are connected to an entity using Relationship.

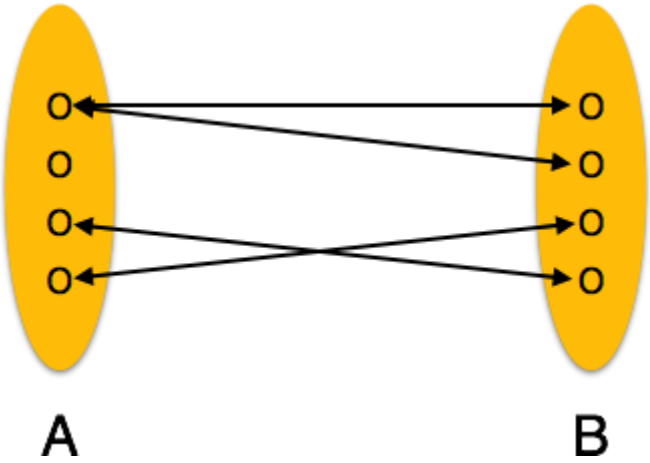
Cardinality defines the number of entities in one entity set, which can be associated with the number of entities of other set via relationship set.

- **One-to-one** − One entity from entity set A can be associated with at most one entity of entity set B and vice versa.



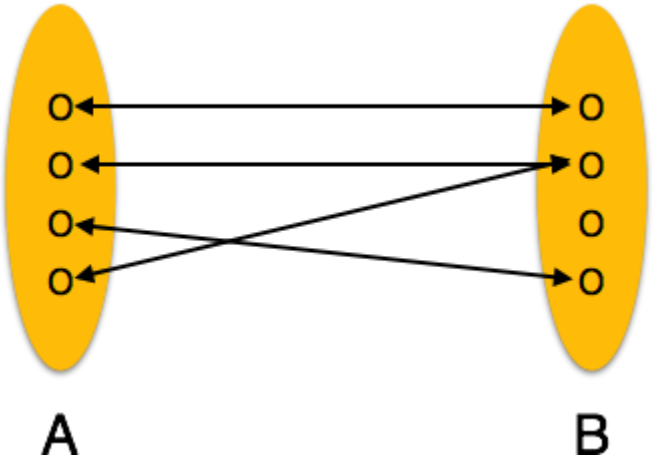
Example: A customer is connected with only one loan using the relationship borrower and a loan is connected with only one customer using borrower

- **One-to-many** − One entity from entity set A can be associated with more than one entities of entity set B however an entity from entity set B, can be associated with at most one entity.



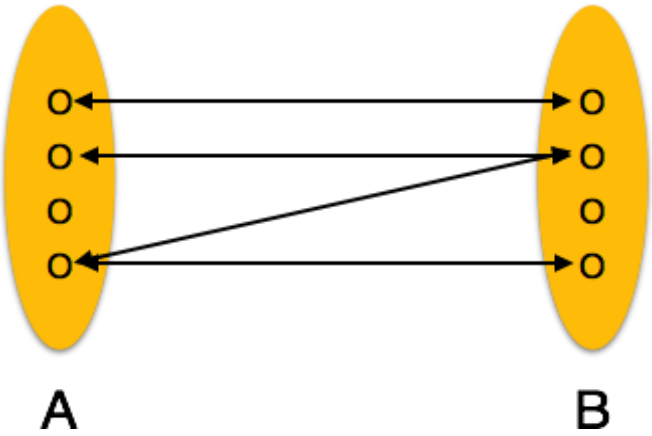
Example : one professor teach more than one students at a time

- **Many-to-one** − More than one entities from entity set A can be associated with at most one entity of entity set B, however an entity from entity set B can be associated with more than one entity from entity set A.



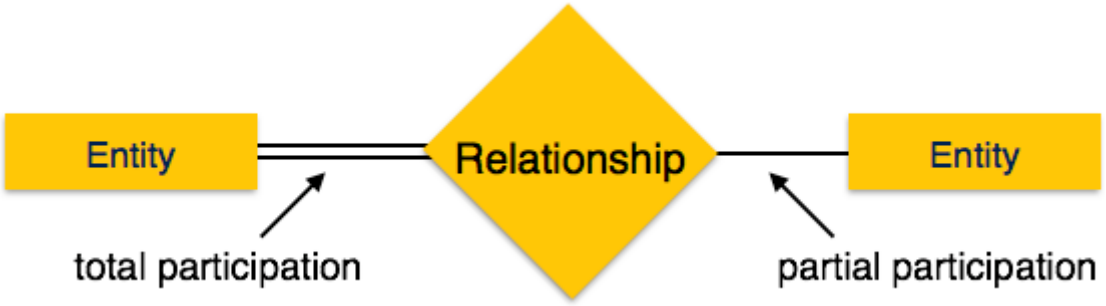
Example : Many employees working one company at a time.

- **Many-to-many** − One entity from A can be associated with more than one entity from B and vice versa.



Example :A customer is connected with more than one loan using borrower and a loan is connected with more than one customer using borrower

Participation Constraints:



1. **Total Participation** − Each entity is involved in the relationship. Total participation is represented by double lines.

Example-

Here,

- Double line between the entity set "Student" and relationship set "Enrolled in" signifies total participation.
- It specifies that each student must be enrolled in at least one course.

2. **Partial participation** − Not all entities are involved in the relationship. Partial participation is represented by single lines.

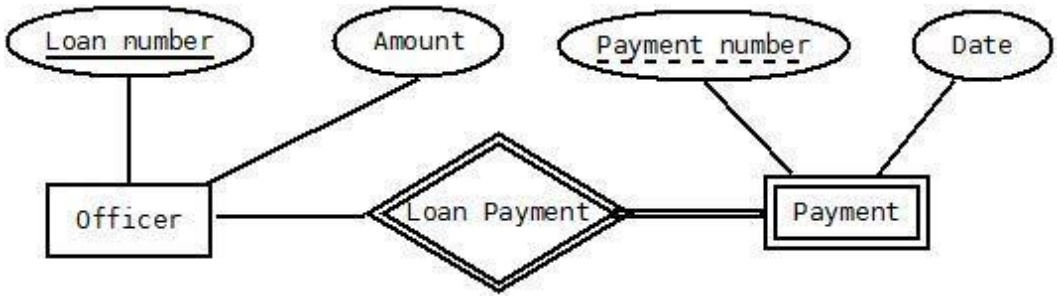
Here,

- Single line between the entity set "Course" and relationship set "Enrolled in" signifies partial participation.
- It specifies that there might exist some courses for which no enrollments are made.

Strong Entity:
A strong entity is not dependent of any other entity in the schema. A strong entity will always have a primary key. Strong entities are represented by a single rectangle. The relationship of two strong entities is represented by a single diamond.

Weak Entity:
A weak entity is dependent on a strong entity to ensure the existence. Unlike a strong entity, a weak entity does not have any primary key. It instead has a partial discriminator key (partial key or discriminator key). A weak entity is represented by a double rectangle. The primary key of a

Weak entity set is created by the primary key of the strong entity set on which the weak entity set is existence dependent and the weak entity set's discriminator.
We underline the discriminator attribute of a weak entity set with a dashed line. The relation between one strong and one weak entity is represented by a double diamond.

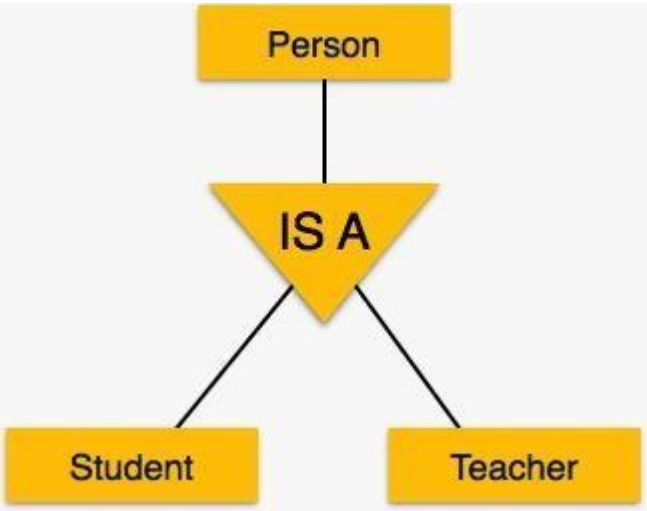


Difference between Strong and Weak Entity:

S.NO	STRONG ENTITY	WEAK ENTITY
1.	Strong entity always has primary key.	While weak entity has partial discriminator key.
2.	Strong entity is not dependent of any other entity.	Weak entity is depend on strong entity.
3.	Strong entity is represented by single rectangle.	Weak entity is represented by double rectangle.
4.	Two strong entity's relationship is represented by single diamond.	While the relation between one strong and one weak entity is represented by double diamond.
5.	Strong entity have either total participation or not.	While weak entity always has total participation.

Generalization

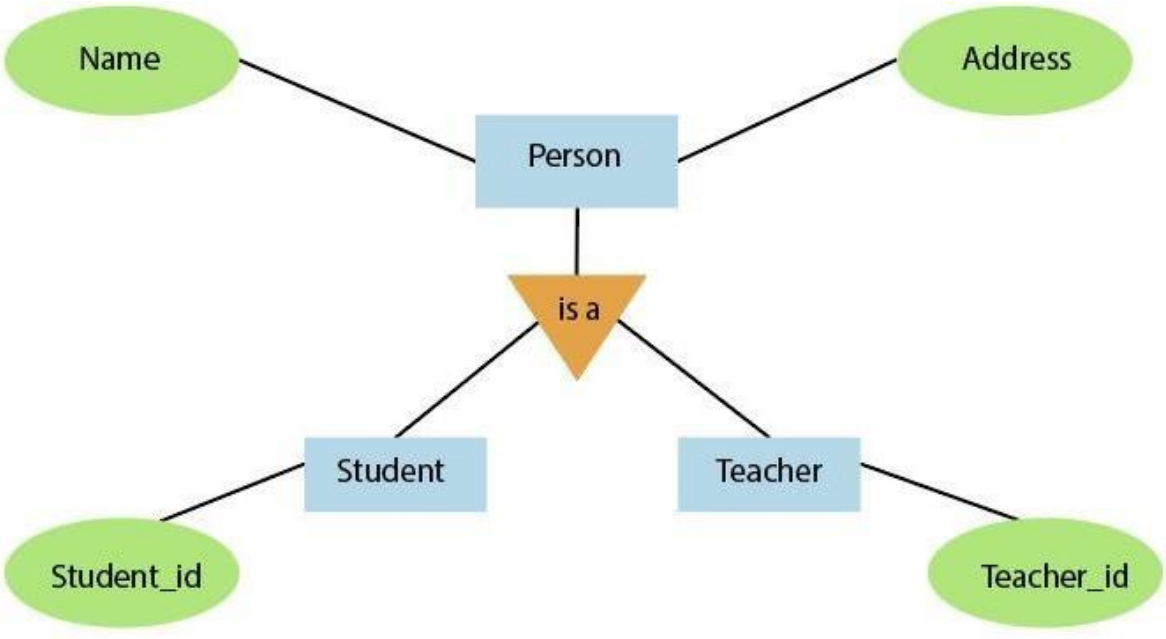
Generalization is the process of extracting common properties from a set of entities and create a generalized entity from it. It is a bottom-up approach in which two or more entities can be generalized to a higher level entity if they have some attributes in common. In generalization, a number of entities are brought together into one generalized entity based on their similar characteristics. For Example, STUDENT and FACULTY can be generalized to a higher level entity called PERSON. In this case, common attributes like P_NAME, P_ADD become part of higher entity (PERSON) and specialized attributes like S_FEE become part of specialized entity (STUDENT).



Specialization

Specialization is the opposite of generalization. In specialization, a group of entities is divided into sub-groups based on their characteristics. In specialization, an entity is divided into sub-entities based on their characteristics. It is a top-down approach where higher level entity is

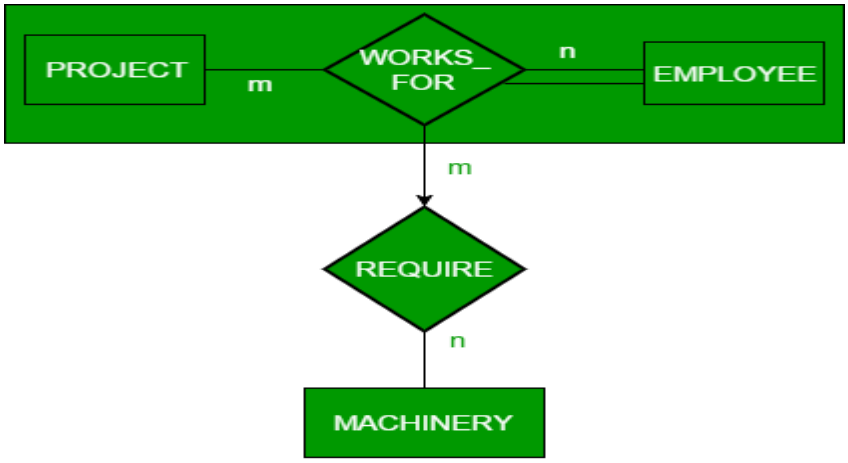
specialized into two or more lower level entities. For Example, EMPLOYEE entity in an Employee management system can be specialized into DEVELOPER, TESTER etc. In this case, common attributes like E_NAME, E_SAL etc. become part of higher entity (EMPLOYEE) and specialized attributes like TES_TYPE become part of specialized entity (TESTER).



Generalisation

Aggregation

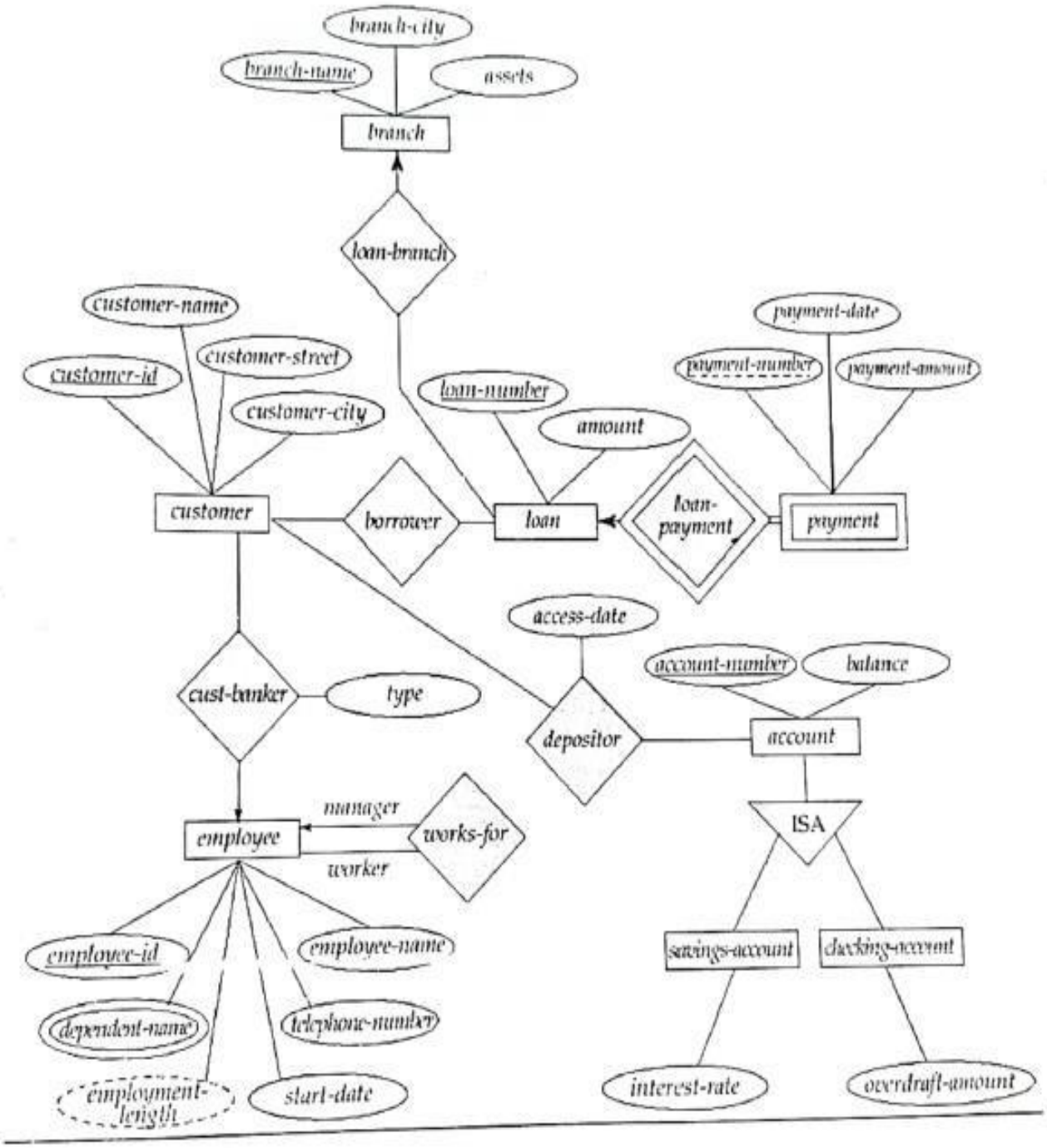
In aggregation, the relation between two entities is treated as a single entity. In aggregation, relationship with its corresponding entities is aggregated into a higher level entity.
An aggregation is an abstraction through which relationships are treated as higher level entity sets and can participate in relationships.
An ER diagram is not capable of representing relationship between an entity and a relationship which may be required in some scenarios. In those cases, a relationship with its corresponding entities is aggregated into a higher level entity. For Example, Employee working for a project may require some machinery. So, REQUIRE relationship is needed between relationship WORKS_FOR and entity MACHINERY. Using aggregation, WORKS_FOR relationship with its entities EMPLOYEE and PROJECT is aggregated into single entity and relationship REQUIRE is created between aggregated entity and MACHINERY.

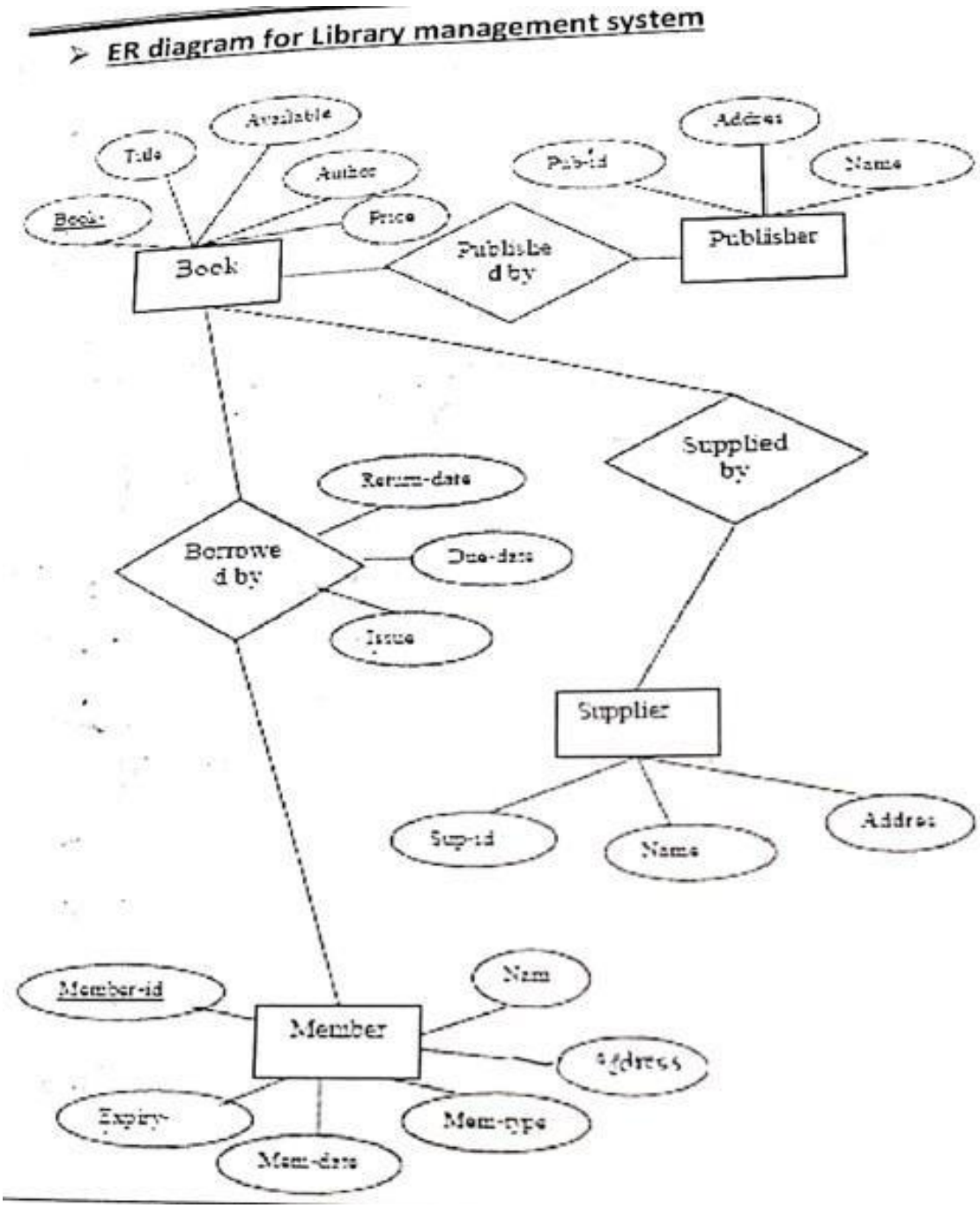


Aggregation

ER DIAGRAM

- **ER diagram for bank management system.**





Types of keys

Key is an attribute or collection of attributes that uniquely identifies an entity among entity set.

For example, the roll_number of a student makes him/her identifiable among students.

1. Super Key

- A Super key for an entity set is a set of one or more attribute whose combined value uniquely identified entity from entity set.
- Ex: An Entity set employee is defined which contain attributes employee no, employee name, address. For this entity set combination of employee no, employee name is super key for entity set employee. Here employee no can also uniquely identify each employee record.so employee no is another super key.

2. Candidate Key

- A minimal super key is called a candidate key. An entity set may have more than one candidate key.
- Candidate keys are selected from the set of super keys, the only thing we take care while selecting candidate key is: It should not have any redundant attribute. That's the reason they are also termed as minimal super key.
- Ex: Roll no and combination of name and address are two candidate key for the entity set of student.

Table: Employee

Emp_SSN	Emp_Number	Emp_Name
123456789	226	Steve
999999321	227	Ajeet
888997212	228	Chaitanya
777778888	229	Robert

Super keys: The above table has following super keys. All of the following sets of super key are able to uniquely identify a row of the employee table.

- {Emp_SSN}
- {Emp_Number}
- {Emp_SSN, Emp_Number}
- {Emp_SSN, Emp_Name}
- {Emp_SSN, Emp_Number, Emp_Name}
- {Emp_Number, Emp_Name}

Candidate Keys: As I mentioned in the beginning, a candidate key is a minimal super key with no redundant attributes. The following two set of super keys are chosen from the above sets as there are no redundant attributes in these sets.

- {Emp_SSN}
- {Emp_Number}

3. Primary Key

- A primary key is one of the candidate keys chosen by the database designer to uniquely identify the entity set.
- A **primary key** is a minimal set of attributes (columns) in a table that uniquely identifies tuples (rows) in that table.
- We can say for the above example that either {Emp_SSN} or {Emp_Number} can be chosen as a primary key for the table Employee.

Primary Key Example in DBMS

Lets take an example to understand the concept of primary key. In the following table, there are three attributes: Stu_ID, Stu_Name & Stu_Age. Out of these three attributes, one attribute or a set of more than one attributes can be a primary key. Attribute Stu_Name alone cannot be a primary key as more than one students can have same name. Attribute Stu_Age alone cannot be a primary key as more than one students can have same age. Attribute Stu_Id alone is a primary key as each student has a unique id that can identify the student record in the table.

Note: In some cases an attribute alone cannot uniquely identify a record in a table, in that case we try to find a set of attributes that can uniquely identify a row in table

Table Name: STUDENT

Stu_Id	Stu_Name	Stu_Age
101	Steve	23
102	John	24
103	Robert	28
104	Steve	29
105	Carl	29

Points to Note regarding Primary Key

- We denote usually denote it by underlining the attribute name (column name).
- The value of primary key should be unique for each row of the table. The column(s) that makes the key cannot contain duplicate values.
- The attribute(s) that is marked as primary key is not allowed to have null values.
- Primary keys are not necessarily to be a single attribute (column). It can be a set of more than one attributes (columns). For example {Stu_Id, Stu_Name} collectively can identify the tuple in the above table, but we do not choose it as primary key because Stu_Id alone is enough to uniquely identifies rows in a table and we always go for minimal set. Having that said, we should choose more than one columns as primary key only when there is no single column that can uniquely identify the tuple in table.

4) Foreign Key

- A FOREIGN KEY is a key used to link two tables together.
- A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table.
- The table containing the foreign key is called the child table, and the table containing the candidate key is called the referenced or parent table.
- A foreign key is different from a super key, candidate key or primary key because a foreign key is the one that is used to link two tables together or create connectivity between the two.

“Persons” Table

PersonID	LastName	FirstName	Age
1	Hansen	Ola	30
2	Svendson	Tove	23
3	Pettersen	Kari	20

"Orders" table:

OrderID	OrderNumber	PersonID
1	77895	3
2	44678	3
3	22456	2
4	24562	1

- Notice that the "PersonID" column in the "Orders" table points to the "PersonID" column in the "Persons" table.
- The "PersonID" column in the "Persons" table is the PRIMARY KEY in the "Persons" table.
- The "PersonID" column in the "Orders" table is a FOREIGN KEY in the "Orders" table

.

The following SQL creates a FOREIGN KEY on the "PersonID" column when the "Orders" table is created:

create table "Persons" ("PersonID" numeric(5) primary key, "LastName" varchar(20), "FirstName" varchar(20),"Age" numeric(5));

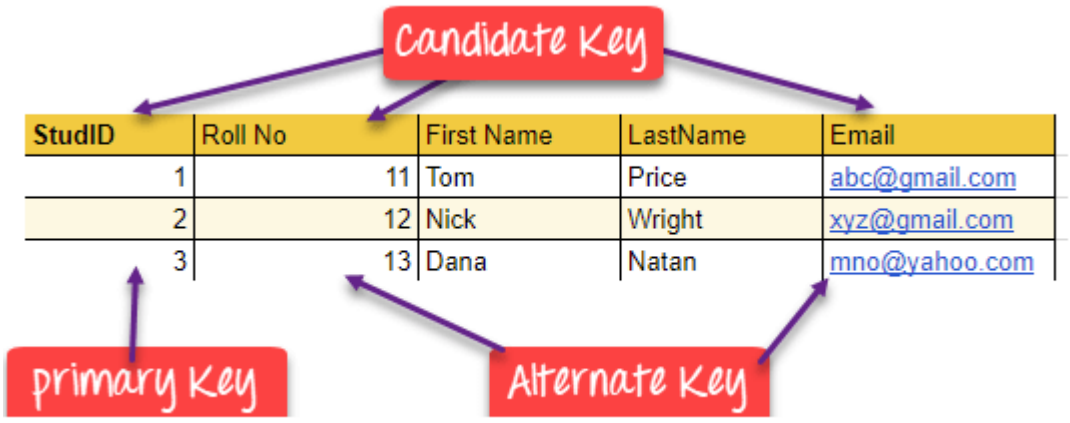
create table "Orders"("OrderID" numeric(5) primary key,"OrderNumber" numeric(5),"PersonID" numeric(5) , foreign key ("PersonID") references "Persons" ("PersonID") on delete cascade on update cascade);

5) Alternate Key

- An alternate is a secondary candidate key that is capable of identifying a row uniquely. However, such a key is not used as a primary key because, as we have discussed in our previous section that out of all the generated candidate keys, only one key is selected as the primary key. Thus, the other remaining keys are known as **Alternate Keys** or **Secondary Keys**.

Example:

In this table, StudID, Roll No, Email are qualified to become a primary key. But since StudID is the primary key, Roll No, Email becomes the alternative key.



6) Composite Key

- It is a combination of two or more columns that uniquely identify rows in a table. The combination of columns guarantees uniqueness, though individually uniqueness is not guaranteed. Hence, they are combined to uniquely identify records in a table.
- A 'combination of two or more' better describes the word 'composite'. Thus, a composite key in DBMS is a candidate key that is composed of two or more attributes and is capable of uniquely identifying a table or a relation.

Example of Composite Key

Below is an example to understand the working of a composite key.

Consider table **A** having three columns or attributes, which are as follows:

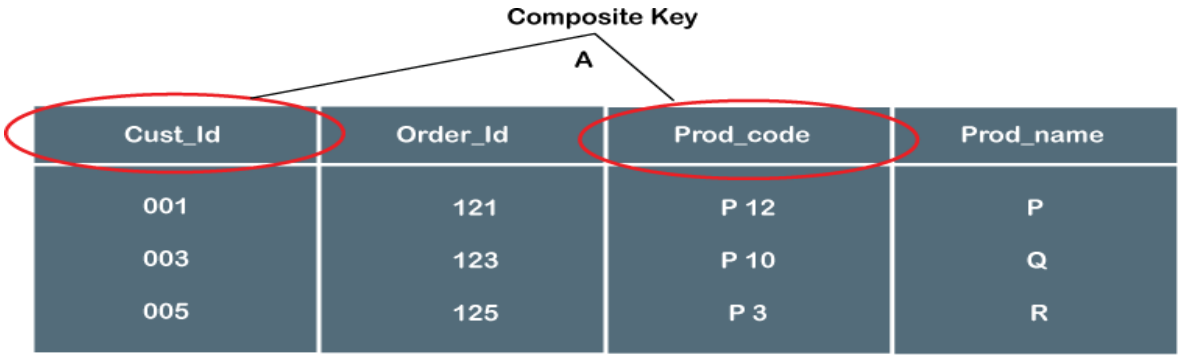
Cust_Id: A customer id is provided to each customer who visits and is stored in this field.

Order_Id: Each order placed by the customer is given an order id, which is stored in this field.

Prod_code: It holds the code value for the products available.

Prod_name: An attribute holding the name of the product on the specified product code.

Below is the table that shows the diagram for the above table:



From the table, we found that no attribute is available that alone can identify a record in the table and can become a primary key. However, the combination of some attributes can form a key and can identify a tuple in the table. In the above example, **Cust_Id** and **Prod_code** can together form a primary key because they alone are not able to identify a tuple, but together they can do so.

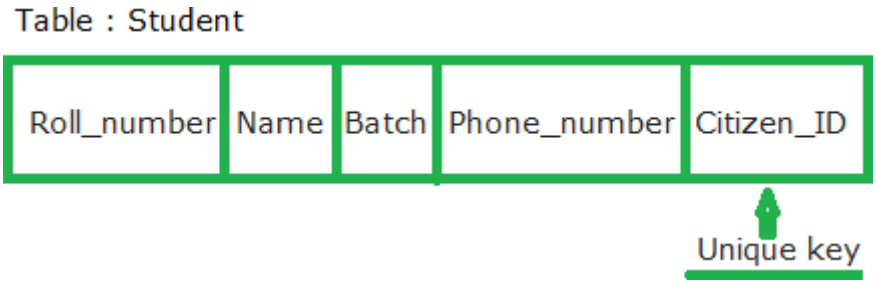
Points to be noted:

- The attribute Cust_Id can work as a primary key alone if the other attributes which are identifiable through it are related to the customer's details only, such as customer's phone number, name, address, etc.
- Similarly, in the case of Prod_code, it fails to be a primary key alone for this table because it can identify the Prod_name but cannot identify the Cust_Id and Order_Id. Thus, we cannot make Prod_code as the primary key.
- Both Cust_Id and Prod_code together can identify all the records of the table because using such a combination, and we can identify the Order_Id of the customer and also can identify the Prod_name of the particular Prod_code.
- Therefore, {Cust_Id, Prod_code} is the composite key for the table.

7) Unique Key

- A unique key is a set of one or more than one fields/columns of a table that uniquely identify a record in a database table.
- You can say that it is little like primary key but it can accept only one null value and it cannot have duplicate values.
- The unique key and primary key both provide a guarantee for uniqueness for a column or a set of columns.
- There is an automatically defined unique key constraint within a primary key constraint.
- There may be many unique key constraints for one table, but only one PRIMARY KEY constraint for one table.

For better understanding of unique key we take Student table with Roll_number, Name, Batch, Phone_number and Citizen_ID attributes.

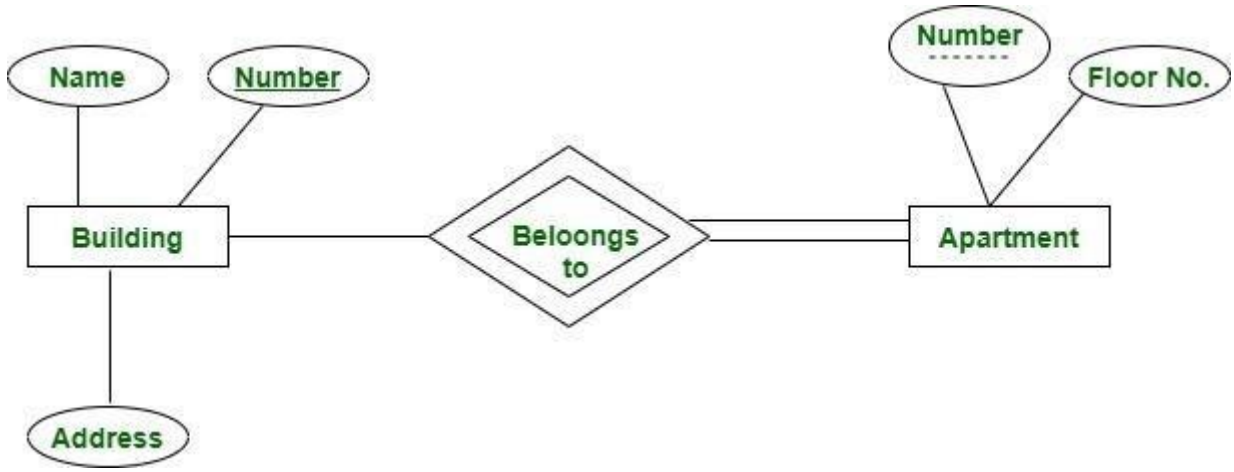


Roll number attribute is already assigned with the primary key and Citizen_ID can have unique constraints where each entry in a Citizen_ID column should be unique because each citizen of a country must have his or her Unique identification number like Aadhaar Number. But if student is migrated to another country in that case, he or she would not have any Citizen_ID and the entry could have a NULL value as only one NULL is allowed in the unique constraint.

8) Partial Key :

The set of attributes that are used to uniquely identify a weak entity set is called the Partial key. Only a bunch of the tuples can be identified using the partial keys. The partial Key of the weak entity set is also known as a discriminator.

It is just a part of the key as only a subset of the attributes can be identified using it. It is partially unique and can be combined with other strong entity set to uniquely identify the tuples.



Partial Key apartment number is shown with a dashed line.

Here we have an apartment as a weak entity and building as a strong entity type connected via ‘belongs to’ relationship set. Apartment number is not globally unique i.e. more than one apartment may have same number globally but it is unique for a particular building since a building may not have same apartment number. Thus apartment number cannot be primary key of entity Apartment but it is a partial key shown with a dashed line.

Comparison Chart:

Paramenter	PRIMARY KEY	UNIQUE KEY
Basic	Used to serve as a unique identifier for each row in a table.	Uniquely determines a row which isn’t primary key.
NULL value acceptance	Cannot accept NULL values.	Can accept one NULL value.
Number of keys that can be defined in the table	Only one primary key	More than one unique key
Index	Creates clustered index	Creates non-clustered index

UNIT 4: Normalization and Concepts of SQL

Why normalization (Insertion, Updating, Deletion anomalies)

Normalization is the process of splitting relations into well-structured relations that allow users to insert, delete, and update tuples without introducing database. Without normalization many problems can occur when trying to load an integrated conceptual model into the DBSM. These problems arise from relations that are generated directly from user views are called anomalies.

Anomalies in DBMS

There are three types of anomalies that occur when the database is not normalized. These are – Insertion, update and deletion anomaly. Let's take an example to understand this.

Employee_ID	Name	Department	Student_Group
123	Rosemol	Accounting	Alpha org
234	Naveen	Marketing	Marketing Club
234	Naveen	Marketing	Management Club
456	Thomas	PRODUCTION	Technology Org.
456	Thomas	PRODUCTION	Alpha org

Example: 4.1

An **update anomaly** is a data inconsistency that results from data redundancy and a partial update. For above example 4.1, each employee in a company has a department associated with them as well as the student group they participate in. If Thomas' department is an error it must be updated at least 2 times or there will be inconsistent data in the database. If the user performing the update does not realize the data is stored redundantly the update will not be done properly.

Example 4.2: For example Jones moving address-you need to update all instances of jones's address.

StudentNum	CourseNum	Student Name	Address	Course
S21	9201	Jones	Edlinburgh	Accounts
S21	9267	Jones	Edlinburgh	Accounts
S24	9267	Smith	Glasgow	Physics
S30	9201	Richards	Manchester	Computing
S30	9322	Richards	Manchester	Maths

A **deletion anomaly** is the unintended loss of data due to deletion of other data. For example 4.1, if the student group Alpha org disbanded and was deleted from the table above, Rosemol and the Accounting department would stop to exist. This results in database inconsistencies and is an example of how combining information that does not really belong together into one table can cause problems.

Example 4.3: Consider what happens if Student S30 is the last student to leave the course. All information about the course is lost.

StudentNum	CourseNum	Student Name	Address	Course
S21	9201	Jones	Edlinburgh	Accounts
S21	9267	Jones	Edlinburgh	Accounts
S24	9267	Smith	Glasgow	Physics
S30	9201	Richards	Manchester	Computing
S30	9322	Richards	Manchester	Maths

An **insertion anomaly** is the inability to add data to the database due to absence of other data. For example 4.1, assume Student_Group is defined so that null values are not allowed. If a new employee is hired but not immediately assigned to a Student_Group then this employee could not be entered into the database. This results in database inconsistencies due to omission.

Example 4.4: We can't add a new course unless we have at least one student enrolled on the course.

StudentNum	CourseNum	Student Name	Address	Course
S21	9201	Jones	Edlinburgh	Accounts
S21	9267	Jones	Edlinburgh	Accounts
S24	9267	Smith	Glasgow	Physics
S30	9201	Richards	Manchester	Computing
S30	9322	Richards	Manchester	Maths

Update, deletion, and insertion anomalies are very undesirable in any database. Anomalies are avoided by the process of normalization.

Normalization Rules:

Concepts of Dependency, Transitive Dependency

What is Functional Dependency

Functional dependency in DBMS, as the name suggests is a relationship between attributes of a table dependent on each other. Introduced by E. F. Codd, it helps in preventing data redundancy and gets to know about bad designs.

The attributes of a table is said to be dependent on each other when an attribute of a table uniquely identifies another attribute of the same table.

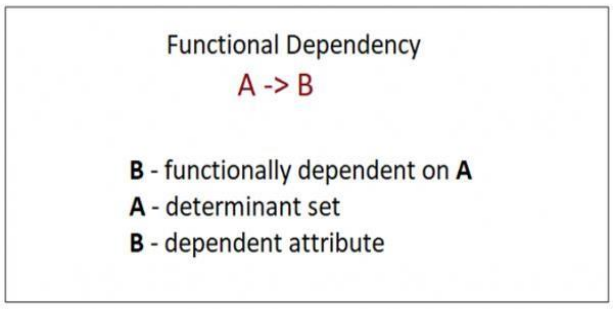
Functional dependency is represented by an arrow sign (→) that is, X→Y, where X functionally determines Y. The left-hand side attributes determine the values of attributes on the right-hand side.

To understand the concept thoroughly, let us consider P is a relation with attributes A and B. Functional Dependency is represented by -> (arrow sign)

Then the following will represent the functional dependency between attributes with an arrow sign –

A -> B

Above suggests the following:



Example

The following is an example that would make it easier to understand functional dependency.

We have a **<Department>** table with two attributes **DeptId** and **DeptName**.

DeptId =DepartmentID
DeptName = Department Name

The **DeptId** is our primary key. Here, **DeptId** uniquely identifies the **DeptName** attribute. This is because if you want to know the department name, then at first you need to have the **DeptId**.

DeptId	DeptName
001	Finance
002	Marketing
003	HR

Therefore, the above functional dependency between **DeptId** and **DeptName** can be determined as **DeptName** is functionally dependent on **DeptId**−

DeptId -> DeptName

Transitive dependency

A functional dependency is said to be transitive if it is indirectly formed by two functional dependencies.
If A->B and B->C are two FDs then A->C is called transitive dependency.

Example 1 – In relation STUDENT,
FD set: {STUD_NO -> STUD_NAME,

STUD_NO -> STUD_STATE,

STUD_STATE -> STUD_COUNTRY,

STUD_NO -> STUD_AGE}

Candidate Key: {STUD_NO}

For the relation, STUD_NO -> STUD_STATE and STUD_STATE -> STUD_COUNTRY are true. So STUD_COUNTRY is transitively dependent on STUD_NO.

Prime and non-prime attributes
Attributes which are parts of any candidate key of relation are called as prime attribute, others are non-prime attributes. For example for the relation {rollno, name, city, age},{rollno} is a candidate key and hence rollno is a prime attribute and name, city, age are non prime attributes.

Armstrong's axioms (or Inference Rules)

The term Armstrong axioms refer to the sound and complete set of inference rules or axioms, introduced by William W. Armstrong, that is used to test the logical implication of **functional dependencies**. If F is a set of functional dependencies then the closure of F, denoted as F^+, is the set of all functional dependencies logically implied by F. Armstrong's Axioms are a set of rules, that when applied repeatedly, generates a closure of functional dependencies.

We can use the following three rules to find logically implied functional dependencies. By applying these rules *repeatedly*, we can find all of *F*+, given *F*. This collection of rules is called Armstrong's axioms in honor of the person who first proposed it.

• **Reflexivity rule.** If α is a set of attributes and $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$ holds.
• **Augmentation rule**. If $\alpha \rightarrow \beta$ holds and γ is a set of attributes, then $\gamma\alpha \rightarrow \gamma\beta$ holds.
• **Transitivity rule**. If $\alpha \rightarrow \beta$ holds and $\beta \rightarrow \gamma$ holds, then $\alpha \rightarrow \gamma$ holds

Although Armstrong's axioms are complete, it is tiresome to use them directly for the Computation of *F*+.
To simplify matters further, we list additional rules

• **Union rule.** If $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta\gamma$ holds.
• **Decomposition rule.** If $\alpha \rightarrow \beta\gamma$ holds, then $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds.
• **Pseudotransitivity rule.** If $\alpha \rightarrow \beta$ holds and $\gamma\beta \rightarrow \delta$ holds, then $\alpha\gamma \rightarrow \delta$ holds.

Let us apply our rules to the example of schema $R = (A, B, C, G, H, I)$ and the set F of functional dependencies $\{A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H\}$. We list several members of *F*+ here:

- $A \rightarrow H$. Since $A \rightarrow B$ and $B \rightarrow H$ hold, we apply the transitivity rule. Observe that it was much easier to use Armstrong's axioms to show that $A \rightarrow H$ holds than it was to argue directly from the definitions, as we did earlier in this section.
- $CG \rightarrow HI$. Since $CG \rightarrow H$ and $CG \rightarrow I$, the union rule implies that $CG \rightarrow HI$.
- $AG \rightarrow I$. Since $A \rightarrow C$ and $CG \rightarrow I$, the pseudotransitivity rule implies that $AG \rightarrow I$ holds.

Another way of finding that $AG \rightarrow I$ holds is as follows: We use the augmentation rule on $A \rightarrow C$ to infer $AG \rightarrow CG$. Applying the transitivity rule to this dependency and $CG \rightarrow I$, we infer $AG \rightarrow I$

1st Normal Form, 2nd Normal Form, 3rd Normal Form, B.C.N.F.

Normalization is a database design technique that reduces data redundancy and eliminates undesirable characteristics like Insertion, Update and Deletion Anomalies. Normalization rules divides larger tables into smaller tables and links them using relationships. The purpose of Normalization in SQL is to eliminate redundant (repetitive) data and ensure data is stored logically.

1st Normal Form (1NF)
If a relation contain composite or multi-valued attribute, it violates first normal form or a relation is in first normal form if it does not contain any composite or multi-valued attribute. A relation is in first normal form if every attribute in that relation is **singled valued attribute**.

- **Example 1 –** Relation STUDENT in table 1 is not in 1NF because of multi-valued attribute STUD_PHONE. Its decomposition into 1NF has been shown in table 2.

Example 2 –

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY
1	RAM	9716271721, 9871717178	HARYANA	INDIA
2	RAM	9898297281	PUNJAB	INDIA
3	SURESH		PUNJAB	INDIA

Table 1

Conversion to first normal form

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY
1	RAM	9716271721	HARYANA	INDIA
1	RAM	9871717178	HARYANA	INDIA
2	RAM	9898297281	PUNJAB	INDIA
3	SURESH		PUNJAB	INDIA

Table 2

ID Name Courses

---

1 A c1, c2

2 E c3

3 M C2, c3

In the above table Course is a multi valued attribute so it is not in 1NF.

Below Table is in 1NF as there is no multi valued attribute

ID Name Course

---

1 A c1

1 A c2

2 E c3

3 M c2

3 M c3

2nd Normal Form(2NF)

To be in second normal form, a relation must be in first normal form and relation must not contain any partial dependency. A relation is in 2NF if it has **No Partial Dependency,** i.e.**,** no non-prime attribute (attributes which are not part of any candidate key) is dependent on any proper subset of any candidate key of the table.

Partial Dependency – If the proper subset of candidate key determines non-prime attribute, it is called partial dependency.

Example 1 – Consider table-3 as following below.

STUD_NO	COURSE_NO	COURSE_FEE
1	C1	1000
2	C2	1500
1	C4	2000
4	C3	1000
4	C1	1000
2	C5	2000

{Note that, there are many courses having the same course fee. }

Here,
COURSE_FEE cannot alone decide the value of COURSE_NO or STUD_NO;
COURSE_FEE together with STUD_NO cannot decide the value of COURSE_NO;
COURSE_FEE together with COURSE_NO cannot decide the value of STUD_NO;

Hence,
COURSE_FEE would be a non-prime attribute, as it does not belong to the one only candidate key {STUD_NO, COURSE_NO} ;

But, COURSE_NO -> COURSE_FEE , i.e., COURSE_FEE is dependent on COURSE_NO, which is a proper subset of the candidate key. Non-prime attribute COURSE_FEE is dependent on a proper subset of the candidate key, which is a partial dependency and so this relation is not in 2NF.

To convert the above relation to 2NF,
we need to split the table into two tables such as :
Table 1: STUD_NO, COURSE_NO
Table 2: COURSE_NO, COURSE_FEE

Table 1

STUD_NO	COURSE_NO
1	C1
2	C2
1	C4
4	C3
4	C1

Table 2

COURSE_NO	COURSE_FEE
C1	1000
C2	1500
C3	1000
C4	2000
C5	2000

3rd Normal Form(3NF)

A table or a relation is considered to be in **Third Normal Form** only if the table is already in **2NF** and there is no **non-prime** attribute **transitively** dependent on the **candidate key** of a relation (i.e. **no transitive dependency)**.
A relation is in 3NF if **at least one of the following condition holds** in every non-trivial function dependency X –> Y

1. X is a super key.
2. Y is a prime attribute (each element of Y is part of some candidate key).

STUD_NO	STUD_NAME	STUD_STATE	STUD_COUNTRY	STUD_AGE
1	RAM	HARYANA	INDIA	20
2	RAM	PUNJAB	INDIA	19
3	SURESH	PUNJAB	INDIA	21

Table 4

Transitive dependency – If A->B and B->C are two FDs then A->C is called transitive dependency.

- **Example –** In relation STUDENT given in Table 4,
  FD set: {STUD_NO -> STUD_NAME, STUD_NO -> STUD_STATE, STUD_STATE -> STUD_COUNTRY, STUD_NO -> STUD_AGE}
  Candidate Key: {STUD_NO}

  For this relation in table 4, STUD_NO -> STUD_STATE and STUD_STATE -> STUD_COUNTRY are true. So STUD_COUNTRY is transitively dependent on STUD_NO. It violates the third normal form.

  To convert it in third normal form, we will decompose the relation STUDENT (STUD_NO, STUD_NAME, STUD_PHONE, STUD_STATE, STUD_COUNTRY_STUD_AGE) as:

  1) STUDENT (STUD_NO, STUD_NAME, STUD_PHONE, STUD_STATE, STUD_AGE)
  2) STATE_COUNTRY (STATE, COUNTRY)

Boyce-Codd Normal Form (BCNF)

It is an advance version of 3NF that's why it is also referred as 3.5NF. BCNF is stricter than 3NF. A table complies with BCNF if it is in 3NF and for every functional dependency X->Y, X should be the super key of the table.

Example: Suppose there is a company wherein employees work in **more than one department**. They store the data like this:

emp_id	emp_nationality	emp_dept	dept_type	dept_no_of_emp
1001	Austrian	Production and planning	D001	200
1001	Austrian	stores	D001	250
1002	American	design and technical support	D134	100
1002	American	Purchasing department	D134	600

Functional dependencies in the table above:
emp_id -> emp_nationality
emp_dept -> {dept_type, dept_no_of_emp}

Candidate key: {emp_id, emp_dept}

The table is not in BCNF as neither emp_id nor emp_dept alone are keys.To make the table comply with BCNF we can break the table in three tables like this:

emp_dept table:

emp_dept	dept_type	dept_no_of_emp
Production and planning	D001	200
stores	D001	250
design and technical support	D134	100
Purchasing department	D134	600

emp_nationality table:

emp_id	emp_nationality
1001	Austrian
1002	American

emp_dept_mapping table:

emp_id	emp_dept
1001	Production and planning
1001	stores
1002	design and technical support
1002	Purchasing department

Functional dependencies:
emp_id -> emp_nationality
emp_dept -> {dept_type, dept_no_of_emp}

Candidate keys:
For first table: emp_id
For second table: emp_dept
For third table: {emp_id, emp_dept}

This is now in BCNF as in both the functional dependencies left side part is a key.

Key Differences Between 3NF and BCNF

S.NO.	3NF	BCNF
1.	In 3NF there should be no transitive dependency that is no non prime attribute should be transitively dependent on the candidate key.	In BCNF for any relation A->B, A should be a super key of relation.
2.	It is less stronger than BCNF.	It is comparatively more stronger than 3NF.
3.	In 3NF the functional dependencies are already in 1NF and 2NF.	In BCNF the functional dependencies are already in 1NF, 2NF and 3NF.
4.	The redundancy is high in 3NF.	The redundancy is comparatively low in BCNF.

S.NO.	3NF	BCNF
5.	In 3NF there is preservation of all functional dependencies.	In BCNF there may or may not be preservation of all functional dependencies.
6.	It is comparatively easier to achieve.	It is difficult to achieve.
7.	Lossless decomposition can be achieved by 3NF.	Lossless decomposition is hard to achieve in BCNF.

Concepts of Structure Query Language (SQL)

SQL is Structured Query Language, which is a computer language for storing, manipulating and retrieving data stored in a relational database. SQL is a language to operate databases; it includes database creation, deletion, fetching rows, modifying rows, etc.

SQL is the standard language for Relational Database System. All the Relational Database Management Systems (RDMS) like MySQL, MS Access, Oracle, Sybase, Informix, Postgres and SQL Server use SQL as their standard database language.

What Can SQL do?

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database
- SQL can create new databases
- SQL can create new tables in a database
- SQL can create stored procedures in a database
- SQL can create views in a database
- SQL can set permissions on tables, procedures, and views

SQL datatypes : int, float, double, char, varchar, number,varchar2, Text, date

CHAR(size): This data type is used to store character strings values of fixed length. The size in brackets determines the number of character the cell can hold. CHAR takes up 1 byte per character. The data held is right-padded with spaces to whatever length specified.

For Example: In case of Name CHAR(60),if the data held in the variable Name is only 20 characters in length, then the entry will be padded with 40 characters worth of spaces. These spaces will be removed when the value is retrieved through.

VARCHAR(size): This data type is used to store variable length alphanumeric data. It is used to specify a variable length string that can contain numbers, letters, and special characters. It is a more flexible form of the CHAR data type. VARCHAR can store up to 2000 bytes of characters. If we declare datatype as VARCHAR then it will occupy space for NULL values. Name Varchar(10) - If you enter value less than 10, it utilize total 10 spaces. One difference between this

data type and the CHAR data type is, VARCHAR uses non-padded comparison semantics i.e. the inserted values will not be padded with spaces.

Lets understand this with an example, If you have an student name column with size 10; sname CHAR(10) and If a column value 'RAMA' is inserted, **6 empty spaces** will be inserted to the right of the value. If this was a VARCHAR column; sname VARCHAR2(10). Then Varchar will take 4 spaces out of 10 possible and free the next 6 for other usage.

VARCHAR2 Datatype

The VARCHAR2 datatype stores variable-length character strings. VARCHAR2 can store up to 4000 bytes of characters. In case of VARCHAR2 datatype it will not occupy any space. Name Varchar2(10) - If you enter value less than 10 then remaining space is not utilize. For example, assume you declare a column VARCHAR2 with a maximum size of 50 characters. In a single-byte character set, if only 10 characters are given for the VARCHAR2 column value in a particular row, the column in the row's row piece only stores the 10 characters (10 bytes), not 50.

Name	Data type	No. of Bytes	Range
INTEGER	Integer	4	-2^{31} to $2^{31}-1$
FLOAT	Float	4	2^{-1074} to $(2-2^{-52})*2^{1023}$
DOUBLE	Double	4	2^{-1074} to $(2-2^{-52})*2^{1023}$
Numeric	Number	No limit	(Max scale, Max precision) Max scale=unlimited Max precision=$e^{(+/-)2^{31}}$
Text	Text	2GB of text data	
DATE	Date	Stores month, day and year information. Format: YYYY-MM-DD. e.g. '1999-01-01'	

DDL Statements

- **DDL** stands for **D**ata **D**efinition **L**anguage. It is used to define database structure or pattern.
- It is used to create schema, tables, indexes, constraints, etc. in the database.
- Using the DDL statements, you can create the skeleton of the database.
- Data definition language is used to store the information of metadata like the number of tables and schemas, their names, indexes, columns in each table, constraints, etc.

Here are some tasks that come under DDL:

- **Create:** It is used to create objects in the database.
- **Alter:** It is used to alter the structure of the database.
- **Drop:** It is used to delete objects from the database.
- **Truncate:** It is used to remove all records from a table.
- **Rename:** It is used to rename an object.

These commands are used to update the database schema that's why they come under Data definition language.

DML and DQL(Data Query Language) Statements

DML stands for **D**ata **M**anipulation **L**anguage. It is used for accessing and manipulating data in a database. It handles user requests.

Here are some tasks that come under DML:

- **Insert:** It is used to insert data into a table.
- **Update:** It is used to update existing data within a table.
- **Delete:** It is used to delete all records from a table.

DML statements are used for performing queries on the data within schema objects. The purpose of DQL Command is to get some schema relation based on the query passed to it.

Example of DQL:

- SELECT – is used to retrieve data from the database.

UNIT 5: Queries (Single Table only)

Using where clause and operators with where clause

The SQL **WHERE** clause is used to specify a condition while fetching the data from a single table or by joining with multiple tables. If the given condition is satisfied, then only it returns a specific value from the table. You should use the WHERE clause to filter the records and fetching only the necessary records.

The WHERE clause is not only used in the SELECT statement, but it is also used in the UPDATE, DELETE statement, etc., which we would examine in the subsequent chapters.

Syntax

The basic syntax of the SELECT statement with the WHERE clause is as shown below.

```
SELECT column1, column2, columnN
FROM table_name
WHERE [condition]
```

You can specify a condition using the comparison or logical operators like >, <, =, **LIKE, NOT**, etc.

Example

Consider the CUSTOMERS table having the following records −

```
+----------+----------------------+-----------+-----------------------+----------------------+
| ID | NAME      | AGE | ADDRESS    | SALARY    |
+----------+----------------------+-----------+-----------------------+----------------------+
|  1 | Ramesh    |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan    |  25 | Delhi     |  1500.00 |
|  3 | kaushik   |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik    |  27 | Bhopal    |  8500.00 |
|  6 | Komal     |  22 | MP        |  4500.00 |
|  7 | Muffy     |  24 | Indore    | 10000.00 |
+----------+----------------------+-----------+-----------------------+----------------------+
```

The following code is an example which would fetch the ID, Name and Salary fields from the CUSTOMERS table, where the salary is greater than 2000

```
SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE SALARY > 2000;
```

This would produce the following result

```
+----------+----------------------+----------------------+
| ID | NAME      | SALARY    |
+----------+----------------------+----------------------+
|  4 | Chaitali |  6500.00 |
|  5 | Hardik    |  8500.00 |
|  6 | Komal     |  4500.00 |
|  7 | Muffy     | 10000.00 |
```

In, between , like, not in, =, !=, >, <, >=, <=, wildcard operatorsIn

Operator

The IN operator allows you to specify multiple values in a WHERE clause.

The IN operator is a shorthand for multiple OR conditions.

IN Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, ...);
```

Example

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	France
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

The following SQL statement selects all customers that are located in "Germany", "France" or "UK":

```
SELECT * FROM Customers
WHERE Country IN ('Germany', 'France', 'UK');
```

OUTPUT

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	France
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK

NOT IN Operator

The following SQL statement selects all customers that are NOT located in "Germany", "France" or "UK":

```
SELECT * FROM Customers
WHERE Country NOT IN ('Germany', 'France', 'UK');
```

OUTPUT

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

Between Operator

The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.

The BETWEEN operator is inclusive: begin and end values are included.

BETWEEN Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

Example

Below is a selection from the "Products" table in the Northwind sample database:

ProductID	ProductName	SupplierID	CategoryID	Unit	Price
1	Chais	1	1	10 boxes x 20 bags	18
2	Chang	1	1	24 - 12 oz bottles	19
3	Aniseed Syrup	1	2	12 - 550 ml bottles	10
4	Chef Anton's Cajun Seasoning	1	2	48 - 6 oz jars	22
5	Chef Anton's Gumbo Mix	1	2	36 boxes	21.35

The following SQL statement selects all products with a price BETWEEN 10 and 20:

```
SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20;
```

OUTPUT

ProductID	ProductName	SupplierID	CategoryID	Unit	Price
1	Chais	1	1	10 boxes x 20 bags	18
2	Chang	1	1	24 - 12 oz bottles	19
3	Aniseed Syrup	1	2	12 - 550 ml bottles	10

NOT BETWEEN Example

To display the products outside the range of the previous example, use NOT BETWEEN:

Example

```
SELECT * FROM Products
WHERE Price NOT BETWEEN 10 AND 20;
```

OUTPUT

ProductID	ProductName	SupplierID	CategoryID	Unit	Price
4	Chef Anton's Cajun Seasoning	1	2	48 - 6 oz jars	22
5	Chef Anton's Gumbo Mix	1	2	36 boxes	21.35

Like Operator

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

There are two wildcards often used in conjunction with the LIKE operator:

- * or % The asterisk sign or percent sign represents zero, one, or multiple characters
- ? or _ The question mark sign or underscore sign represents a single character

The asterisk sign and the question mark sign can also be used in combinations! You can also combine any number of conditions using AND or OR operators.

Here are some examples showing different LIKE operators with '*' and '?' wildcards:

LIKE Operator	Description
WHERE CustomerName LIKE 'a*'	Finds any values that start with "a"
WHERE CustomerName LIKE '*a'	Finds any values that end with "a"
WHERE CustomerName LIKE '*or*'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '?r*'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a?*'	Finds any values that start with "a" and are at least 2 characters in length
WHERE CustomerName LIKE 'a??*'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a*o'	Finds any values that start with "a" and ends with "o"

Example

Student				
s_id	**s_name**	**s_addr**	**s_city**	**s_phno**
1	Rahul	kamrej	surat	1234567895
2	Raj	vesu	surat	1296385895
3	Mit	ramgadh	valsad	7418529632
4	Fenil	jodhpur	vapi	9638527415
5	Hardik	mahabaleshvar	bhavanagar	9685623471

The following SQL statement selects all students with a studentname starting with "R":

```
SELECT * FROM student
WHERE s_name LIKE 'R*';
```

OUTPUT

s_id	**s_name**	**s_addr**	**s_city**	**s_phno**
1	Rahul	kamrej	surat	1234567895
2	Raj	vesu	surat	1296385895

SQL Comparison Operators

Operator	**Description**	**Example**
=	Equal to	SELECT * FROM “employee” WHERE “salary”=1500;
!=	Not equal to	SELECT * FROM “employee” WHERE “salary” !=1500;
>	Greater than	SELECT * FROM “employee” WHERE “salary” >1500;
<	Less than	SELECT * FROM “employee” WHERE “salary” <1500;
>=	Greater than or equal to	SELECT * FROM “employee” WHERE “salary” >=1500;
<=	Less than or equal to	SELECT * FROM “employee” WHERE “salary”<=1500;

Wildcard operators

A wildcard character is used to substitute one or more characters in a string.

Wildcard characters are used with the SQL LIKE operator. The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

Symbol	Description	Example
* or %	Represents zero or more characters	bl* finds bl, black, blue, and blob
? or _	Represents a single character	h?t finds hot, hat, and hit

Order by, Group by, Distinct

Order by clause

The ORDER BY statement in sql is used to sort the fetched data in either ascending or descending according to one or more columns.

- By default ORDER BY sorts the data in **ascending order.**
- We can use the keyword DESC to sort the data in descending order and the keyword ASC to sort in ascending order.

Syntax of all ways of using ORDER BY is shown below:

Sort according to one column: To sort in ascending or descending order we can use the keywords ASC or DESC respectively.

Syntax:

SELECT “column1”,”column2”,...
FROM “table_name”
ORDER BY “column_name” ASC|DESC

table_name: name of the table.
column_name: name of the column according to which the data is needed to be arranged.
ASC: to sort the data in ascending order.
DESC: to sort the data in descending order.
| : use either ASC or DESC to sort in ascending or descending order

Sort according to multiple columns: To sort in ascending or descending order we can use the keywords ASC or DESC respectively. To sort according to multiple columns, separate the names of columns by (,) operator.

Syntax:

SELECT "*column1*","*column2*",...
FROM *table_name*
ORDER BY "*column1" ASC/DESC," column2" ASC/DESC* ... ASC|DESC;

Example:

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	HARSH	DELHI	XXXXXXXXXX	18
2	PRATIK	BIHAR	XXXXXXXXXX	19
3	RIYANKA	SILIGURI	XXXXXXXXXX	20
4	DEEP	RAMNAGAR	XXXXXXXXXX	18
5	SAPTARHI	KOLKATA	XXXXXXXXXX	19
6	DHANRAJ	BARABAJAR	XXXXXXXXXX	20
7	ROHIT	BALURGHAT	XXXXXXXXXX	18
8	NIRAJ	ALIPUR	XXXXXXXXXX	19

Queries:
Sort according to single column: In this example we will fetch all data from the table **Student** and sort the result in descending order according to the column **ROLL_NO**.

SELECT * FROM "Student" ORDER BY "ROLL_NO" ;

Output:

ROLL_NO	NAME	ADDRESS	PHONE	Age
8	NIRAJ	ALIPUR	XXXXXXXXXX	19
7	ROHIT	BALURGHAT	XXXXXXXXXX	18
6	DHANRAJ	BARABAJAR	XXXXXXXXXX	20
5	SAPTARHI	KOLKATA	XXXXXXXXXX	19
4	DEEP	RAMNAGAR	XXXXXXXXXX	18
3	RIYANKA	SILIGURI	XXXXXXXXXX	20
2	PRATIK	BIHAR	XXXXXXXXXX	19
1	HARSH	DELHI	XXXXXXXXXX	18

To sort in ascending order we have to use ASC in place of DESC.

Sort according to multiple columns: In this example we will fetch all data from the table **Student** and then sort the result in ascending order first according to the column **Age**.and then in descending order according to the column **ROLL_NO**.

SELECT * FROM "Student" ORDER BY "Age" ASC , "ROLL_NO" DESC;

Output:

ROLL_NO	NAME	ADDRESS	PHONE	Age
7	ROHIT	BALURGHAT	XXXXXXXXXX	18
4	DEEP	RAMNAGAR	XXXXXXXXXX	18
1	HARSH	DELHI	XXXXXXXXXX	18
8	NIRAJ	ALIPUR	XXXXXXXXXX	19
5	SAPTARHI	KOLKATA	XXXXXXXXXX	19
2	PRATIK	BIHAR	XXXXXXXXXX	19
6	DHANRAJ	BARABAJAR	XXXXXXXXXX	20
3	RIYANKA	SILIGURI	XXXXXXXXXX	20

In the above output you can see that first the result is sorted in ascending order according to Age. There are multiple rows having same Age. Now, sorting further this result-set according to ROLL_NO will sort the rows with same Age according to ROLL_NO in descending order.

Note that: ASC is the default value for ORDER BY clause. So, if you don't specify anything after column name in ORDER BY clause, the output will be sorted in ascending order by default.
Example: The following query will give similar output as the above:

```
SELECT * FROM "Student" ORDER BY "Age" , "ROLL_NO" DESC;
```

Output:

ROLL_NO	NAME	ADDRESS	PHONE	Age
7	ROHIT	BALURGHAT	XXXXXXXXXX	18
4	DEEP	RAMNAGAR	XXXXXXXXXX	18
1	HARSH	DELHI	XXXXXXXXXX	18
8	NIRAJ	ALIPUR	XXXXXXXXXX	19
5	SAPTARHI	KOLKATA	XXXXXXXXXX	19
2	PRATIK	BIHAR	XXXXXXXXXX	19
6	DHANRAJ	BARABAJAR	XXXXXXXXXX	20
3	RIYANKA	SILIGURI	XXXXXXXXXX	20

Group by clause

The GROUP BY Statement in SQL is used to arrange identical data into groups with the help of some functions. i.e if a particular column has same values in different rows then it will arrange these rows in a group.

Important Points:

- GROUP BY clause is used with the SELECT statement.
- In the query, GROUP BY clause is placed after the WHERE clause.

- In the query, GROUP BY clause is placed before ORDER BY clause if used any.

Syntax:
SELECT column(s), function_name(column2)

FROM table_name

WHERE condition

GROUP BY column1, column2

ORDER BY column1, column2;

function_name: Name of the function used for example, SUM() , AVG().
table_name: Name of the table.
condition: Condition used.
Sample Table:

Student

SUBJECT	YEAR	NAME
English	1	Harsh
English	1	Pratik
English	1	Ramesh
English	2	Ashish
English	2	Suresh
Mathematics	1	Deepak
Mathematics	1	Sayan

Example:
Group By single column: Group By single column means, to place all the rows with same value of only that particular column in one group. Consider the query as shown below:

SELECT NAME, SUM(SALARY) FROM Employee

GROUP BY NAME;

The above query will produce the below output:

NAME	SALARY
Ashish	3000
Dhanraj	3000
Harsh	5500

As you can see in the above output, the rows with duplicate NAMEs are grouped under same NAME and their corresponding SALARY is the sum of the SALARY of duplicate rows. The SUM() function of SQL is used here to calculate the sum.

Group By multiple columns: Group by multiple column is say for example, **GROUP BY column1, column2**. This means to place all the rows with same values of both the columns **column1** and **column2** in one group. Consider the below query:

```
SELECT SUBJECT, YEAR, Count(*)
FROM Student
GROUP BY SUBJECT, YEAR;
```

Output:

SUBJECT	YEAR	Count
English	1	3
English	2	2
Mathematics	1	2

As you can see in the above output the students with both same SUBJECT and YEAR are placed in same group. And those whose only SUBJECT is same but not YEAR belongs to different groups. So here we have grouped the table according to two columns or more than one column.

HAVING Clause

We know that WHERE clause is used to place conditions on columns but what if we want to place conditions on groups?

This is where HAVING clause comes into use. We can use HAVING clause to place conditions to decide which group will be the part of final result-set. Also we can not use the aggregate functions like SUM(), COUNT() etc. with WHERE clause. So we have to use HAVING clause if we want to use any of these functions in the conditions.

Syntax:

```
SELECT column(s), function_name(column2)

FROM table_name

WHERE condition

GROUP BY column1, column2

HAVING condition

ORDER BY column1, column2;
```

function_name: Name of the function used for example, SUM() , AVG().
table_name: Name of the table.
condition: Condition used.

Employee

SI NO	NAME	SALARY	AGE
1	Harsh	2000	19
2	Dhanraj	3000	20
3	Ashish	1500	19
4	Harsh	3500	19
5	Ashish	1500	19

Example:

SELECT NAME, SUM(SALARY) FROM Employee

GROUP BY NAME

HAVING SUM(SALARY)>3000;

Output:

NAME	SUM(SALARY)
HARSH	5500

As you can see in the above output only one group out of the three groups appears in the result-set as it is the only group where sum of SALARY is greater than 3000. So we have used HAVING clause here to place this condition as the condition is required to be placed on groups not columns.

Distinct

The SELECT DISTINCT statement is used to return only distinct (different) values.

Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values.

SELECT DISTINCT Syntax

SELECT DISTINCT *column1, column2, ...*
FROM *table_name*;

Example

Consider the CUSTOMERS table having the following records −

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
```

```
+----------+-----------------------+------------+-------------------------+-----------------------+
```

First, let us see how the following SELECT query returns the duplicate salary records.

```
SELECT SALARY FROM CUSTOMERS
   ORDER BY SALARY;
```

This would produce the following result, where the salary (2000) is coming twice which is a duplicate record from the original table.

```
+----------------------+
| SALARY   |
+----------------------+
|  1500.00 |
|  2000.00 |
|  2000.00 |
|  4500.00 |
|  6500.00 |
|  8500.00 |
| 10000.00 |
+----------------------+
```

Now, let us use the DISTINCT keyword with the above SELECT query and then see the result.

```
SELECT DISTINCT SALARY FROM CUSTOMERS
   ORDER BY SALARY;
```

This would produce the following result where we do not have any duplicate entry.

```
+----------------------+
| SALARY   |
+----------------------+
|  1500.00 |
|  2000.00 |
|  4500.00 |
|  6500.00 |
|  8500.00 |
| 10000.00 |
+----------------------+
```

SQL Logical Operators (AND, OR operators, Exists and not Exists)AND, OR operators

In SQL, the AND & OR operators are used for filtering the data and getting precise result based on conditions.

- The AND and OR operators are used with the WHERE clause.
- These two operators are called conjunctive operators.

The AND and OR operators are used to filter records based on more than one condition:

- The AND operator displays a record if all the conditions separated by AND are TRUE.
- The OR operator displays a record if any of the conditions separated by OR is TRUE.

Student

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	xxxxxxxxxx	18
2	RAMESH	GURGAON	xxxxxxxxxx	18
3	SUJIT	ROHTAK	xxxxxxxxxx	20
4	SURESH	Delhi	xxxxxxxxxx	18
3	SUJIT	ROHTAK	xxxxxxxxxx	20
2	RAMESH	GURGAON	xxxxxxxxxx	18

AND Operator

Basic Syntax:

SELECT * FROM "table_name" WHERE condition1 AND condition2 and ...conditionN;

table_name: name of the table
condition1,2,..N : first condition, second condition and so on

Sample Queries:

1. To fetch all the records from Student table where Age is 18 and ADDRESS is Delhi.
   SELECT * FROM Student WHERE Age = 18 AND ADDRESS = 'Delhi';

   Output:

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	XXXXXXXXXX	18
4	SURESH	Delhi	XXXXXXXXXX	18

2. To fetch all the records from Student table where NAME is Ram and Age is 18.
   SELECT * FROM "Student" WHERE "Age" = 18 AND "NAME" = 'Ram';
   Output:

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	XXXXXXXXXX	18

OR Operator

Basic Syntax:
SELECT * FROM "table_name" WHERE condition1 OR condition2 OR... conditionN;

table_name: name of the table
condition1,2,..N : first condition, second condition and so on
Sample Queries:

1. To fetch all the records from Student table where NAME is Ram or NAME is SUJIT.
   SELECT * FROM Student WHERE NAME = 'Ram' OR NAME = 'SUJIT';
   Output:

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	XXXXXXXXXX	18
3	SUJIT	ROHTAK	XXXXXXXXXX	20
3	SUJIT	ROHTAK	XXXXXXXXXX	20

2. To fetch all the records from Student table where NAME is Ram or Age is 20.
   SELECT * FROM Student WHERE NAME = 'Ram' OR Age = 20;
   Output:

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	XXXXXXXXXX	18
3	SUJIT	ROHTAK	XXXXXXXXXX	20
3	SUJIT	ROHTAK	XXXXXXXXXX	20

Combining AND and OR

You can combine AND and OR operators in below manner to write complex queries.

Sample Queries:
1. To fetch all the records from Student table where Age is 18 NAME is Ram or RAMESH.
SELECT * FROM Student WHERE Age = 18 AND (NAME = 'Ram' OR NAME = 'RAMESH');

Output:

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	XXXXXXXXXX	18
2	RAMESH	GURGAON	XXXXXXXXXX	18

Exists and not Exists

The EXISTS condition in SQL is used to check whether the result of a correlated nested query is empty (contains no tuples) or not. The result of EXISTS is a boolean value True or False. It can be used in a SELECT, UPDATE, INSERT or DELETE statement.

Syntax:
SELECT column_name(s)
FROM table_name
WHERE EXISTS
(**SELECT** column_name(s) **FROM** table_name **WHERE** condition);

Examples:
Consider the following two relation "Customers" and "Orders".

Customers

customer_id	lname	fname	website
401	Singh	Dolly	abc.com
402	Chauhan	Anuj	def.com
403	Kumar	Niteesh	ghi.com
404	Gupta	Shubham	jkl.com
405	Walecha	Divya	abc.com
406	Jain	Sandeep	jkl.com
407	Mehta	Rajiv	abc.com
408	Mehra	Anand	abc.com

Orders

order_id	c_id	order_date
1	407	2017-03-03
2	405	2017-03-05
3	408	2017-01-18
4	404	2017-02-05

Queries

1. **Using EXISTS condition with SELECT statement**
   To fetch the first and last name of the customers who placed atleast one order.

```
SELECT fname, lname
FROM Customers
WHERE EXISTS (SELECT *
        FROM Orders
        WHERE Customers.customer_id = Orders.c_id);
```

Output:

fname	lname
Shubham	Gupta
Divya	Walecha
Rajiv	Mehta
Anand	Mehra

2. Using NOT with EXISTS

Fetch last and first name of the customers who has not placed any order.

```
SELECT lname, fname
FROM Customer
WHERE NOT EXISTS (SELECT *
        FROM Orders
        WHERE Customers.customer_id = Orders.c_id);
```

Output:

lname	fname
Singh	Dolly
Chauhan	Anuj
Kumar	Niteesh
Jain	Sandeep

3. Using EXISTS condition with DELETE statement

Delete the record of all the customer from Order Table whose last name is ‘Mehra’.

```
DELETE
FROM Orders
WHERE EXISTS (SELECT *
        FROM customers
        WHERE Customers.customer_id = Orders.cid
        AND Customers.lname = 'Mehra');
SELECT * FROM Orders;
```

Output:

order_id	c_id	order_date
1	407	2017-03-03
2	405	2017-03-05
4	404	2017-02-05

4. **Using EXISTS condition with UPDATE statement**
   Update the lname as 'Kumari' of customer in Customer Table whose customer_id is 401.

```
UPDATE Customers
SET lname = 'Kumari'
WHERE EXISTS (SELECT *
        FROM Customers
        WHERE customer_id = 401);

SELECT * FROM Customers;
```

Output:

customer_id	lname	fname	website
401	Kumari	Dolly	abc.com
402	Chauhan	Anuj	def.com
403	Kumar	Niteesh	ghi.com
404	Gupta	Shubham	jkl.com
405	Walecha	Divya	abc.com
406	Jain	Sandeep	jkl.com
407	Mehta	Rajiv	abc.com
408	Mehra	Anand	abc.com

Use of Alias

Aliases are the temporary names given to table or column for the purpose of a particular SQL query. It is used when name of column or table is used other than their original names, but the modified name is only temporary.

- Aliases are created to make table or column names more readable.
- The renaming is just a temporary change and table name does not change in the original database.
- Aliases are useful when table or column names are big or not very readable.
- These are preferred when there are more than one table involved in a query.

Basic Syntax:

For column alias:

SELECT column as alias_name FROM table_name;

column: fields in the table
alias_name: temporary alias name to be used in replacement of original column name
table_name: name of table
For table alias:

SELECT column FROM table_name as alias_name;
column: fields in the table
table_name: name of table

alias_name: temporary alias name to be used in replacement of original table name

Student_Details

ROLL_NO	Branch	Grade
1	Information Technology	O
2	Computer Science	E
3	Computer Science	O
4	Mechanical Engineering	A

Queries for illustrating column alias

To fetch ROLL_NO from Student table using CODE as alias name.

SELECT ROLL_NO AS CODE FROM Student;

Output:

CODE
1
2
3
4

To fetch Branch using Stream as alias name and Grade as CGPA from table Student_Details.

SELECT Branch as Stream, Grade as CGPA FROM Student_Details;

Output:

Stream	CGPA
Information Technology	O
Computer Science	E
Computer Science	O
Mechanical Engineering	A

Queries for illustrating table alias

Student

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	XXXXXXXXXX	18
2	RAMESH	GURGAON	XXXXXXXXXX	18
3	SUJIT	ROHTAK	XXXXXXXXXX	20
4	SURESH	Delhi	XXXXXXXXXX	18

Generally table aliases are used to fetch the data from more than just single table and connect them through the field relations.
To fetch Grade and NAME of Student with Age = 20.

SELECT s.NAME, d.Grade FROM Student s, Student_Details d WHERE s.Age=20 AND s.ROLL_NO=d.ROLL_NO;

Output:

NAME	Grade
SUJIT	O

Constraints (Table level and Attribute Level)

SQL constraints are used to specify rules for the data in a table.Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.

Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.
The following constraints are commonly used in SQL:
- **NOT NULL** - Ensures that a column cannot have a NULL value
- **CHECK** - Ensures that all values in a column satisfies a specific condition
- **DEFAULT** - Sets a default value for a column when no value is specified
- **UNIQUE** - Ensures that all values in a column are different
- **PRIMARY KEY** - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
- **FOREIGN KEY** - Uniquely identifies a row/record in another table

NOT NULL, CHECK, DEFAULT<u>NOT NULL</u>

By default, a column can hold NULL values.The NOT NULL constraint enforces a column to NOT accept NULL values.This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.

SQL NOT NULL on CREATE TABLE
The following SQL ensures that the "ID", "LastName", and "FirstName" columns will NOT accept NULL values when the "Persons" table is created:

Example
```
CREATE TABLE Persons (
ID int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255) NOT NULL,
Age int
);
```

SQL NOT NULL on ALTER TABLE
To create a NOT NULL constraint on the "Age" column when the "Persons" table is already created, use the following SQL:

```
ALTER TABLE Persons ALTER Age int NOT NULL;
```

SQL CHECK Constraint
The CHECK constraint is used to limit the value range that can be placed in a column. If you define a CHECK constraint on a single column it allows only certain values for this column. If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

SQL CHECK on CREATE TABLE
The following SQL creates a CHECK constraint on the "Age" column when the "Persons" table is created. The CHECK constraint ensures that the age of a person must be 18, or older:

```
CREATE TABLE Persons (
ID int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Age int, CHECK (Age>=18)
);
```

To allow naming of a CHECK constraint, and for defining a CHECK constraint on multiple columns, use the following SQL syntax:

```
CREATE TABLE Persons (
ID int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Age int,
City varchar(255),
CONSTRAINT CHK_Person CHECK (Age>=18 AND City='Sandnes')
);
```

SQL CHECK on ALTER TABLE
To create a CHECK constraint on the "Age" column when the table is already created, use the following SQL:

ALTER TABLE Persons ADD CHECK (Age>=18);

To allow naming of a CHECK constraint, and for defining a CHECK constraint on multiple columns, use the following SQL syntax:

ALTER TABLE Persons ADD CONSTRAINT CHK_PersonAge CHECK (Age>=18 AND City='Sandnes');

DROP a CHECK Constraint
To drop a CHECK constraint, use the following SQL:
ALTER TABLE Persons DROP CONSTRAINT CHK_PersonAge;

SQL DEFAULT Constraint
The DEFAULT constraint is used to provide a default value for a column. The default value will be added to all new records IF no other value is specified.

SQL DEFAULT on CREATE TABLE
The following SQL sets a DEFAULT value for the "City" column when the "Persons" table is created:

```
CREATE TABLE Persons(
ID int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Age int,
City varchar(255) DEFAULT 'Sandnes'
 );
```

SQL DEFAULT on ALTER TABLE
To create a DEFAULT constraint on the "City" column when the table is already created, use the following SQL:

ALTER TABLE Persons ALTER City SET DEFAULT 'Sandnes';
OR
ALTER TABLE Persons ALTER COLUMN City SET DEFAULT 'Sandnes';

DROP a DEFAULT Constraint
To drop a DEFAULT constraint, use the following SQL:

ALTER TABLE Persons ALTER City DROP DEFAULT;
OR
ALTER TABLE Persons ALTER COLUMN City DROP DEFAULT;

UNIQUE, Primary Key, Foreign KeySQL
UNIQUE Constraint
The UNIQUE constraint ensures that all values in a column are different.Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns.A PRIMARY KEY constraint automatically has a UNIQUE constraint.
However, you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

SQL UNIQUE Constraint on CREATE TABLE
The following SQL creates a UNIQUE constraint on the "ID" column when the "Persons" table is created.
CREATE TABLE Persons (

```
ID int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Age int, UNIQUE (ID)
);
```

To name a UNIQUE constraint, and to define a UNIQUE constraint on multiple columns, use the following SQL syntax:

```
CREATE TABLE Persons (
ID int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Age int,
CONSTRAINT UC_Person UNIQUE (ID,LastName)
);
```

SQL UNIQUE Constraint on ALTER TABLE

To create a UNIQUE constraint on the "ID" column when the table is already created, use the following SQL:

```
ALTER TABLE Persons ADD UNIQUE (ID);
```

To name a UNIQUE constraint, and to define a UNIQUE constraint on multiple columns, use the following SQL syntax:

```
ALTER TABLE Persons ADD CONSTRAINT UC_Person UNIQUE (ID,LastName);
```

DROP a UNIQUE Constraint

To drop a UNIQUE constraint, use the following SQL:

```
ALTER TABLE Persons DROP CONSTRAINT UC_Person;
```

SQL PRIMARY KEY Constraint

The PRIMARY KEY constraint uniquely identifies each record in a table.Primary keys must contain UNIQUE values, and cannot contain NULL values.A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).

SQL PRIMARY KEY on CREATE TABLE

The following SQL creates a PRIMARY KEY on the "ID" column when the "Persons" table is created:

```
CREATE TABLE Persons (
ID int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Age int, PRIMARY KEY (ID)
);
```

```
CREATE TABLE Persons (
ID int NOT NULL PRIMARY KEY,
LastName varchar(255) NOT NULL,
```

```
FirstName varchar(255), Age int
);
```

To allow naming of a PRIMARY KEY constraint, and for defining a PRIMARY KEY constraint on multiple columns, use the following SQL syntax:

```
CREATE TABLE Persons (
ID int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Age int,
CONSTRAINT PK_Person PRIMARY KEY (ID,LastName)
);
```

Note: In the example above there is only ONE PRIMARY KEY (PK_Person). However, the VALUE of the primary key is made up of TWO COLUMNS (ID + LastName).

SQL PRIMARY KEY on ALTER TABLE

To create a PRIMARY KEY constraint on the "ID" column when the table is already created, use the following SQL:

```
ALTER TABLE Persons ADD PRIMARY KEY (ID);
```

To allow naming of a PRIMARY KEY constraint, and for defining a PRIMARY KEY constraint on multiple columns, use the following SQL syntax:

```
ALTER TABLE Persons ADD CONSTRAINT PK_Person PRIMARY KEY (ID,LastName);
```

DROP a PRIMARY KEY Constraint

To drop a PRIMARY KEY constraint, use the following SQL:

```
ALTER TABLE Persons DROP PRIMARY KEY;

ALTER TABLE Persons DROP CONSTRAINT PK_Person;
```

SQL FOREIGN KEY on CREATE TABLE

The following SQL creates a FOREIGN KEY on the "PersonID" column when the "Orders" table is created:

```
CREATE TABLE Orders (
OrderID int NOT NULL,
OrderNumber int NOT NULL,
PersonID int, PRIMARY KEY (OrderID),
FOREIGN KEY (PersonID) REFERENCES Persons(PersonID) );
```

To allow naming of a FOREIGN KEY constraint, and for defining a FOREIGN KEY constraint on multiple columns, use the following SQL syntax:

```
CREATE TABLE Orders (
OrderID int NOT NULL,
OrderNumber int NOT NULL,
PersonID int,
PRIMARY KEY (OrderID),
CONSTRAINT FK_PersonOrder FOREIGN KEY (PersonID) REFERENCES Persons (PersonID)
```

);

SQL FOREIGN KEY on ALTER TABLE
To create a FOREIGN KEY constraint on the "PersonID" column when the "Orders" table is already created, use the following SQL:

ALTER TABLE Orders ADD FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);

To allow naming of a FOREIGN KEY constraint, and for defining a FOREIGN KEY constraint on multiple columns, use the following SQL syntax:

ALTER TABLE Orders ADD CONSTRAINT FK_PersonOrder FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);

DROP a FOREIGN KEY Constraint
To drop a FOREIGN KEY constraint, use the following SQL:

ALTER TABLE Orders DROP CONSTRAINT FK_PersonOrder;

On Delete Cascade
When we create a foreign key using this option, it deletes the referencing rows in the child table when the referenced row is deleted in the parent table which has a primary key.

Suppose that we have two tables: buildings and rooms. In this database model, each building has one or many rooms. However, each room belongs to one only one building. A room would not exist without a building.

The relationship between the buildings and rooms tables is one-to-many (1:N).
When you delete a row from the buildings table, you also want to delete all rows in the rooms table that references to the row in the buildings table. For example, when you delete a row with building no. 2 in the buildings table you also want the rows in the rooms table that refers to building number 2 will be also removed.

The following are steps that demonstrate how the ON DELETE CASCADE referential action works.
Step 1. Create the buildings table:
CREATE TABLE buildings (
building_no INT PRIMARY **KEY** ,
building_name VARCHAR(255) **NOT** NULL,
address VARCHAR(255) **NOT** NULL
);

	building_no	building_name	address
▶	1	ACME Headquaters	3950 North 1st Street CA 95134
	2	ACME Sales	5000 North 1st Street CA 95134

Step 2. Create the rooms table:
CREATE TABLE rooms (
room_no INT PRIMARY **KEY** ,
room_name VARCHAR(255) **NOT** NULL,
building_no INT **NOT** NULL,
FOREIGN KEY (building_no)
REFERENCES buildings (building_no)

ON DELETE CASCADE
);

room_no	room_name	building_no
1	Amazon	1
2	War Room	1
3	Office of CEO	1
4	Marketing	2
5	Showroom	2

Notice that the ON DELETE CASCADE clause at the end of the foreign key constraint definition.now when you delete building no 2 record from parent table, all the rows that reference to building_no 2 were automatically deleted.

room_no	room_name	building_no
1	Amazon	1
2	War Room	1
3	Office of CEO	1

ON UPDATE CASCADE: When we create a foreign key using UPDATE CASCADE the referencing rows are updated in the child table when the referenced row is updated in the parent table which has a primary key.

SQL Functions :

Aggregate Functions: avg(), max(), min(), sum(), count(), first(),last().

PRODUCT	COMPANY	QTY	RATE	COST
Item1	Com1	2	10	20
Item2	Com2	3	25	75
Item3	Com1	2	30	60
Item4	Com3	5	10	50
Item5	Com2	2	20	40
Item6	Cpm1	3	25	75
Item7	Com1	5	30	150
Item8	Com1	3	10	30

Item9	Com2	2	25	50
Item10	Com3	4	30	120

AVG function

The AVG function is used to calculate the average value of the numeric type. AVG function returns the average of all non-Null values.

Syntax

AVG()

or

AVG([ALL|DISTINCT] expression)

Example:

```
SELECT AVG(COST)
FROM PRODUCT_MAST;
```

Output:

```
67.00
```

MAX Function

MAX function is used to find the maximum value of a certain column. This function determines the largest value of all selected values of a column.

Syntax

MAX()

or

MAX([ALL|DISTINCT] expression)

Example:

```
SELECT MAX(RATE)
FROM PRODUCT_MAST;
```

```
30
```

MIN Function

MIN function is used to find the minimum value of a certain column. This function determines the smallest value of all selected values of a column.

Syntax

MIN()

or

MIN([ALL|DISTINCT] expression)

Example:

```
SELECT MIN(RATE)
FROM PRODUCT_MAST;
```

Output:

```
10
```

SUM Function

Sum function is used to calculate the sum of all selected columns. It works on numeric fields only.

Syntax

SUM()

or

SUM([ALL|DISTINCT] expression)

Example: SUM()

```
SELECT SUM(COST)
FROM PRODUCT_MAST;
```

Output:

```
670
```

Example: SUM() with WHERE

```
SELECT SUM(COST)
FROM PRODUCT_MAST
WHERE QTY>3;
```

Output:

```
320
```

Example: SUM() with GROUP BY

```
SELECT SUM(COST)
FROM PRODUCT_MAST
WHERE QTY>3
GROUP BY COMPANY;
```

Output:

```
Com1    150
Com3    170
```

Example: SUM() with HAVING

```
SELECT COMPANY, SUM(COST)
FROM PRODUCT_MAST
GROUP BY COMPANY
HAVING SUM(COST)>=170;
```

Output:

```
Com1    335
Com3    170
```

COUNT FUNCTION

COUNT function is used to Count the number of rows in a database table. It can work on both numeric and non-numeric data types.

COUNT function uses the COUNT(*) that returns the count of all the rows in a specified table. COUNT(*) considers duplicate and Null.

Example: COUNT()

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

Output:

```
10
```

Example: COUNT with WHERE

```
SELECT COUNT(*)
FROM PRODUCT_MAST;
WHERE RATE>=20;
```

Output:

```
7
```

Example: COUNT() with DISTINCT

```
SELECT COUNT(DISTINCT COMPANY)
FROM PRODUCT_MAST;
```

Output: 3

Example: COUNT() with GROUP BY

```
SELECT COMPANY, COUNT(*)
FROM PRODUCT_MAST
GROUP BY COMPANY;
```

Output:

```
Com1     5
Com2     3
Com3     2
```

Example: COUNT() with HAVING

```
SELECT COMPANY, COUNT(*)
FROM PRODUCT_MAST
GROUP  BY  COMPANY
HAVING COUNT(*)>2;
```

Output:

```
Com1     5
Com2     3
```

FIRST Function

SQL SELECT FIRST() function returns the first value of selected column.

Syntax

```
SELECT FIRST(Column_name) FROM table_name;
OR
SELECT FIRST(Column_name) AS First_Name FROM table_name;
```

Example :
Query using FIRST() Function
Consider the following table titled as **'Stationary',** which contains the information of products.

Write a query to display a first name from table 'Stationary'.

ID	Name	Quantity	Price
1	Pen	10	200
2	Ink	15	300
3	Notebook	20	400

4	Pencil	30	150

SELECT FIRST(Name) AS First_name FROM Stationary;

The result is shown in the following table.

First_name
Pen

Note: The FIRST() function is only supported in MS Access.

SQL LAST() Function

SQL SELECT LAST() function returns the last value of selected column.

Syntax:

```
SELECT LAST(Column_name) FROM table_name;
OR
SELECT LAST(Column_name) AS Last_Name FROM table_name;
```

Example : Query using LAST() Function.
Consider the following table titled as **'Stationary',** which contains the information of products.

Write a query to display a last name from table 'Stationary'.

ID	Name	Quantity	Price
1	Pen	10	200
2	Ink	15	300
3	Notebook	20	400
4	Pencil	30	150

SELECT Last(Name) AS Last_name FROM Stationary;

The result is shown in the following table.

Last_name
Pencil

Scalar Functions: ucase(), lcase(), round(), mid().

UCASE(): It converts the value of a field to uppercase.

Syntax:
SELECT UCASE(column_name) FROM table_name;

Queries:
Converting names of students from the table Students to uppercase.

SELECT UCASE(NAME) FROM Students;

Output:

NAME
HARSH
SURESH
PRATIK
DHANRAJ
RAM

LCASE(): It converts the value of a field to lowercase.
Syntax:
SELECT LCASE(column_name) FROM table_name;

Queries:
Converting names of students from the table Students to lowercase.
SELECT LCASE(NAME) FROM Students;

Output:

NAME
harsh
suresh
pratik
dhanraj
ram

Round()
The ROUND() function rounds a number to a specified number of decimal places.
Syntax
ROUND(*number*, *decimals*)

Parameter Values **Parameter**	**Description**
number	Required. The number to be rounded
decimals	Optional. The number of decimal places to round *number* to. If omitted, it returns the integer (no decimals)

Example

Student_ID	Student_name	City	Marks
1	Raju	Chennai	90.34
2	Mani	Delhi	95
3	Thiyagarajan	Vizag	79.67
4	Surendren	Pune	82

SQL query:
SELECT ROUND(Marks, 1) from student;

SQL query output:

90.3
95
79.7
82

SQL query:
SELECT ROUND(Marks, 0) from student;

SQL query output:

90
95
80
82

MID()

The MID() function extracts a substring from a string (starting at any position).
Syntax
MID(*string*, *start*, *length*)

Parameter Values

Parameter	Description
string	Required. The string to extract from
start	Required. The start position. Can be both a positive or negative number. If it is a positive number, this function extracts from the beginning of the string. If it is a negative number, this function extracts from the end of the string
length	Required. The number of characters to extract

Example
Extract a substring from the text in a column (start at position 2, extract 5 characters):
SELECT MID(CustomerName, 2, 5) AS ExtractString FROM Customers;

Creating sequence

Sequence is a set of integers 1, 2, 3, … that are generated and supported by some database systems to produce unique values on demand.

- A sequence is a user defined schema bound object that generates a sequence of numeric values.
- Sequences are frequently used in many databases because many applications require each row in a table to contain a unique value and sequences provides an easy way to generate them.
- The sequence of numeric values is generated in an ascending or descending order at defined intervals and can be configured to restart when max_value exceeds.

Syntax:
CREATE SEQUENCE sequence_name

START WITH initial_value

INCREMENT BY increment_value

MINVALUE minimum value

MAXVALUE maximum value

CYCLE|NOCYCLE;

sequence_name: Name of the sequence.

initial_value: starting value from where the sequence starts.
Initial_value should be greater than or equal
to minimum value and less than equal to maximum value.

increment_value: Value by which sequence will increment itself.
Increment_value can be positive or negative.

minimum_value: Minimum value of the sequence.
maximum_value: Maximum value of the sequence.

cycle: When sequence reaches its set_limit
it starts from beginning.

nocycle: An exception will be thrown
if sequence exceeds its max_value.

Following is the sequence query creating sequence in ascending order.

Example 1:

```
CREATE SEQUENCE sequence_1

start with 1

increment by 1

minvalue 0

maxvalue 100

cycle;
```

Above query will create a sequence named *sequence_1*.Sequence will start from 1 and will be incremented by 1 having maximum value 100. Sequence will repeat itself from start value after exceeding 100.

Example to use sequence : create a table named students with columns as id and name.

```
CREATE TABLE students

(

ID number(10),

NAME char(20)

);
```

Now insert values into table

```
INSERT into students VALUES(sequence_1.nextval,'Ramesh');

INSERT into students VALUES(sequence_1.nextval,'Suresh');
```

where *sequence_1.nextval* will insert id's in id column in a sequence as defined in sequence_1.

Output:

```
| ID |    NAME    |

----------------

| 1 |   Ramesh   |

| 2 |   Suresh   |

 ---------------
```

Creating simple view, updating view, dropping view.

It provides a way to look at a number of fields without regard to the table to which any of the fields belong.
A View can consists of some of the fields on one table Or, it can consist of fields from more than one table.
A table has a set of definition, and it physically stores the data.
A view also has a set of definitions, which is build on top of table(s) or other view(s), and it does not physically store the data.

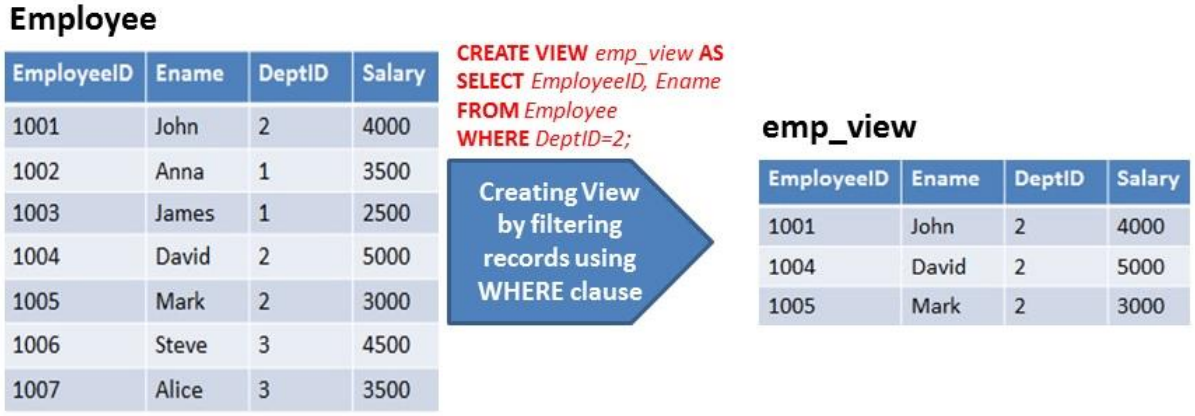
Views in SQL are considered as a virtual table. A view also contains rows and columns.

To create the view, we can select the fields from one or more tables present in the database.A view can either have specific rows based on certain condition or all the rows of a table.

The view has primarily two purposes:

- Simplify the complex SQL queries.
- Provide restriction to users from accessing sensitive data.

VIEW:



The syntax for creating a view is as follows:

CREATE VIEW "VIEW_NAME" AS "SQL Statement";

Let's use a simple example to illustrate. Say we have the following table:

Table ***Customer***

Column Name	Data Type
First_Name	char(50)
Last_Name	char(50)
-Address	char(50)
City	char(50)
Country	char(25)
Birth_Date	datetime

CREATE table Customer(cno numeric(20), First_Name varchar(20), Last_Name varchar(20), Country varchar(20));

Insert into customer values(1,'xyz','a','India');

Insert into customer values(2,'pqr','b','nepal');

Insert into customer values(3,'def','c','brazil');

Insert into customer values(2,'rst','d','brazil');

we want to create a view called ***V_Customer*** that contains only the First_Name, Last_Name, and Country columns from this table, we would type in,

```
CREATE VIEW V_Customer
AS SELECT First_Name, Last_Name, Country
FROM Customer;
```

Instead of sending the complex query to the database all the time, you can save the query as a view and then SELECT * FROM view

If we want to find out the content of this view, we type in,

```
SELECT * FROM V_Customer
```

Deleting View

A view can be deleted using the Drop View statement.

Syntax

DROP VIEW view_name;

Example:

If we want to delete the View **V_Customer**, we can do this as:

DROP VIEW **V_Customer**;

SQL Updating a View

A view can be updated with the CREATE OR REPLACE VIEW command.

SQL CREATE OR REPLACE VIEW Syntax

CREATE OR REPLACE VIEW *view_name* AS
SELECT *column1, column2, ...*
FROM *table_name*
WHERE *condition*;

The following SQL adds the "City" column to the "Brazil Customers" view:

Example

```
CREATE or replace VIEW V_Customer AS
SELECT First_Name, Last_Name, Country
FROM Customer
WHERE Country = 'brazil';
```

Difference between View and Table

View	Table
View is a virtual table based on the result-set of an SQL statement and that is Stored in the database with some name.	The table is structured with a set number of columns and a boundless number of columns
A view is additionally a database object which is utilized as a table and inquiry that can be connected to different tables.	The table is database's which are utilized to hold the information that is utilized in applications and reports.
View depends on the table	Table is independent data object in database
The view is utilized to query certain information which is contained in a few distinct tables	The table holds fundamental client information and holds cases of a characterized object.
In the view, you will get frequently queried information.	In the table, changing the information in the database likewise changes the information appeared in the view