Lesson#02

Data Manipulation

Data plays an important role in making personal or business decisions. Companies extract data from multiple sources stored in different formats, including structured, semi-structured, and unstructured. It is vital to ensure that the data is clean and organized enough to get accurate insights to make better business decisions.

Data Manipulation is the process of **modifying** or changing data to make it more clean, organized, and readable for data analysis.

Skills covered in Lesson:

- **Skill 2.1:** Import, store, and export data
- **Skill 2.2:** Clean data
- **Skill 2.3:** Organize data
- **Skill 2.4:** Aggregate data

Extract, Transform, and Load (ETL) Introduction

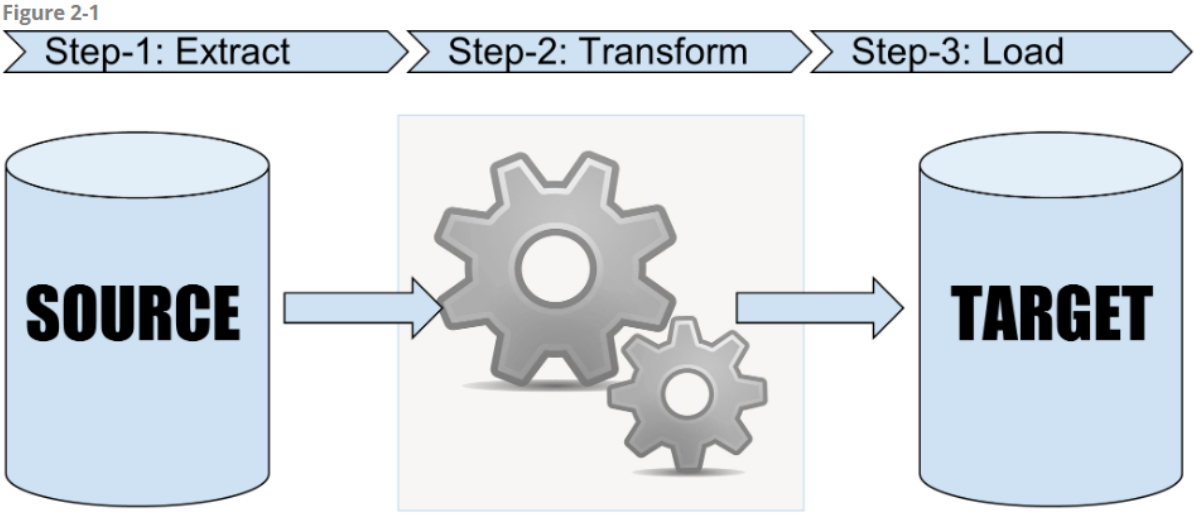
In any organization, **data** is **stored** in **multiple storage** systems and in **different formats**. For the purpose of data analysis, an **analyst needs to bring** data from different storage systems to **a common data repository in a common format**. *The process of extracting data from one or more source data storage systems, transforming it to a required format, and then loading it into another target data storage system for further analysis is known as* ***Extract, Transform, and Load (ETL) processing***.

This skill covers how to:

- Describe ETL processing
- Perform ETL with relational data
- Perform ETL with data stored in delimited files
- Perform ETL with data stored in XML files
- Perform ETL with data stored in JSON files

ETL stands for **Extract**, **Transform**, and **Load**. It is a three-step process in which you first extract data from a source system, perform transformations on that data, and then load it into a target system.

Figure 2-1



The ETL process involves three steps: Extract, Transform, and Load.

Extract

During the data extraction phase, **raw data** is **extracted** from one or more **source** systems **to a staging area.** The raw data **can be structured, semi-structured, or unstructured**. **Possible sources** include, but are not limited to the following:

- **Relational or non-relational databases**
- **Flat files like CSV, JSON, or XML**
- **Sensors, email, or web pages**

Transform

During the data transformation phase, the **raw data** extracted from the source system is **processed** and **transformed** for **its intended analytical use** case. Some of the tasks performed during this phase are as follows:

- **Filtering, cleansing, and deduplicating** (i.e., eliminating duplicate or redundant data) data
- **Removal, encryption, decryption**, or hashing of critical data as per industry or government data regulations
- Performing necessary **calculations or translations like currency conversion**, measurement units, or standardizing text formats
- Changing the data format to **match the schema** of the target system

Some methods of filtering, cleaning, and formatting data are discussed later in this lesson.

Load

During the data load phase, the transformed data is loaded into the target system.

The data loading process can be one of the following types:

1. **Full data load** where all data is loaded into the target system

2. **Incremental data load** where periodic loading of incremental data changes is done after the initial full loading of data
3. **Full refresh data load** where the old data is fully replaced by the new data in the target system

ETL Process Video

ETL with relational data

As discussed in Lesson 1, an RDBMS (Relational Database Management System) is used to store and manage data in a relational database. **In a relational database, the tables are related to each other.** There are many varieties of RDBMS available such as *Microsoft SQL Server, MySQL, Oracle database*, etc.

Consider the following two tables, the EMPLOYEE table and the DEPARTMENT table where both tables are related to each other through a common field named dept_id.

Table 2-1

EMPLOYEE table			
emp_id	**emp_name**	**dept_id**	**mgr_id**
100	Sam	null	null
101	Tom	1	100
102	Nir	1	101
103	Rick	3	100
104	Maria	3	103

Table 2-2

DEPARTMENT table	
dept_id	**dept_name**
1	Finance
2	Sales
3	HR

A **PRIMARY KEY** is a column or a collection of columns in a table that is used to uniquely identify each record in the table. For example, emp_id is the primary key of the EMPLOYEE table, and dept_id is the primary key of the DEPARTMENT table.

A **FOREIGN KEY** is a collection of one or more columns in a table that references the PRIMARY KEY in another table, thereby establishing a relationship between the two tables. For example, the column dept_id in the EMPLOYEE table is a foreign key because it references the primary key of the DEPARTMENT table.

Introduction to relational databases video

What is SQL?

SQL stands for Structured Query Language. It is used **for creating**, **manipulating**, **and retrieving** data stored in a relational database.

The following section briefly describes different categories of SQL statements.

- Data Definition Language (**DDL**) is used to create, modify, or drop a table or another database object.
- Data Query Language (**DQL**) is used to retrieve records from one or more tables.
- Data Manipulation Language (**DML**) is used to insert, update, or delete table records.
- Data Control Language (**DCL**) is used to grant or revoke privileges to users.

Table 2-3

SQL statements		**Description**
DDL	**CREATE**	used to create a new table or another database object
	ALTER	used to modify an existing table or another database object
	DROP	used to delete an existing table or another database object
DQL	**SELECT**	used to retrieve records from one or more tables
DML	**INSERT**	used to insert one or more new records into a table
	UPDATE	used to modify existing records in a table
	DELETE	used to delete existing records in a table
DCL	**GRANT**	used to grant privileges to users
	REVOKE	used to remove privileges granted to users

SELECT Statements

This section will focus on using SELECT statements to retrieve records from one or more tables.

SELECT statements are used to select records from one or more database tables.

Syntax of a SELECT statement:

1. The following syntax selects all the columns from a table. The **asterisk (*)** stands for all.

```
SELECT * FROM table_name;
```

2. The following syntax selects only **col1 and col2** from a table.

```
SELECT col1, col2 FROM table_name;
```

3. The **SELECT DISTINCT** statement is used to return only rows that have distinct values in the listed columns. The **'…'** means that you can list **any number of columns** in the list.

```
SELECT DISTINCT col1,col2,... FROM table_name;
```

4. The **WHERE** clause is used to filter records returned by the query based on a specified condition.

```
SELECT col1, col2,... FROM table_name WHERE condition;
```

5. The **LIMIT** clause is used by some RDBMSs, such as MySQL, to select only the specified number N of records from the table.

```
SELECT col1, col2,... FROM table_name LIMIT N;
```

6. *Note: In Microsoft SQL Server, you must use the TOP clause instead of LIMIT*. For example,

```
SELECT TOP N col1, col2,... FROM table_name;
```

Examples of SELECT statements:

1. The following query will select only emp_id and emp_name from the EMPLOYEE table.

```
SELECT emp_id, emp_name FROM employee;
```

Table 2-4

emp_id	**emp_name**
100	Sam
101	Tom
102	Nir
103	Rick
104	Maria

2. The following query will select all the columns from the EMPLOYEE table.

```
SELECT * FROM employee;
```

Table 2-5

emp_id	**emp_name**	**dept_id**	**mgr_id**
100	Sam	null	null
101	Tom	1	100
102	Nir	1	101
103	Rick	3	100
104	Maria	3	103

3. The following query will select only one instance of each distinct dept_id from the EMPLOYEE table.

```
SELECT DISTINCT dept_id FROM employee;
```

Table 2-6

dept_id
null
1
3

4. The following query will select all records that have dept_id of 3 from the EMPLOYEE table.

```
SELECT * FROM employee WHERE dept_id=3;
```

Table 2-7

emp_id	emp_name	dept_id
103	Rick	3
104	Maria	3

5. The following query will select the emp_id and emp_name of the first three records from the EMPLOYEE table.

```
SELECT emp_id, emp_name FROM employee LIMIT 3;
```

Table 2-8

emp_id	emp_name
100	Sam
101	Tom
102	Nir

SQL Alias

A column **alias** is **an alternate name** used in a query to better describe the contents of the column in the query results. The AS keyword is used to create a column alias.

The following query is using the alias employee_id for the column emp_id and the alias employee_name for the column emp_name to make it more readable.

```
SELECT emp_id AS employee_id, emp_name AS employee_name  FROM employee;
```

Table aliases are frequently used to make table names shorter or more readable in complex queries involving multiple tables. The following query is using the alias e for the EMPLOYEE table and d for the DEPARTMENT table.

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

The above query is using SQL JOIN which will be explained in the next section.

Watch the following videos by Ben Forta to learn more about SQL SELECT statements.

Syntax of the SQL SELECT statement video

Retrieve individual columns in a table using the SQL SELECT statement video

Retrieve multiple columns in a table using the SQL SELECT statement video

Retrieve all columns in a table using the SQL SELECT statement video

Retrieve distinct rows in a SQL table video

Limit retrieved data in SQL

SQL joins

In any relational database, the data is stored in multiple tables that are linked together by a common key value. To address business needs, it is often necessary to combine data from one and more tables to get the desired output based on some conditions. Such combined data can easily be achieved using the SQL JOIN clause.

Joins in SQL are used to get combined rows from two or more tables based on a related column between them.

Types of joins:

There are different types of joins that are used depending on the need.

These joins are:

- Inner join
- Left Outer join
- Right Outer join
- Cross join
- Full outer join

- Self join

Inner Joins

An inner join selects **all rows from both tables** as long as the common field matches up in both tables. An inner join is frequently used because it allows you to generate a result set that shows data that is spread across multiple tables while excluding data that is missing from some of the joined tables.

Syntax of an inner join:

```
SELECT
table1.column1, table1.column2, table2.column1, table2.column2,...
FROM table1
INNER JOIN table2
ON table1.matching_column = table2.matching_column;
```

Example of an inner join:

Consider the EMPLOYEE and DEPARTMENT tables as demonstrated in the previous examples. Suppose you want to generate a list of employees and their departments. You do not want to include any departments that do not have employees or any employees that are not assigned to a department.

The following query returns the result of the inner join between the EMPLOYEE table and the DEPARTMENT table based on the matching department IDs.

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

Table 2-8

emp_id	emp_name	dept_name
101	Tom	Finance
102	Nir	Finance
103	Rick	HR
104	Maria	HR

Notice that the rows in the DEPARTMENT table and EMPLOYEE table that do not have a matching dept_id are not returned by this query.

Left Outer Joins

A left join or left outer join returns **all the rows from the left table** and only the matching rows from the right table. It returns null for the non-matching rows from the right table.

The syntax for a left join:

```
SELECT
table1.column1,table1.column2,table2.column3,...
FROM table1
LEFT JOIN table2
ON table1.matching_column = table2.matching_column;
Or
SELECT table1.column1,table1.column2,table2.column3,...
FROM table1
LEFT OUTER JOIN table2
ON table1.matching_column = table2.matching_column;
```

Example of a left join:

Consider the EMPLOYEE and DEPARTMENT tables as demonstrated in the previous examples. Suppose you want to generate a list of all the employees and the department they work in. You do not want to include any newly set up departments that do not have employees yet.

To get the above result, the following query will be used to perform a left join between the EMPLOYEE table and the DEPARTMENT table based on the matching department IDs.

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

Table 2-9

emp_id	emp_name	dept_name
100	Sam	null
101	Tom	Finance
102	Nir	Finance
103	Rick	HR
104	Maria	HR

In the above result set, all the employees are included along with the department they work in. It is worth noting that:

- The row in the EMPLOYEE table with emp_id of 100 is included in the result, but its dept_name is null because it has no matching department in the DEPARTMENT table.
- The row in the DEPARTMENT table with dept_name of Sales and dept_id of 2 is not included in the result as it has no employees yet.

Right Outer Join:

A right join or right outer join **returns all the rows from the right table** and only the matching rows from the left table. It returns null for the non-matching rows from the left table, being the opposite of the left join.

Syntax of a right join:

```
SELECT
table1.column1,table1.column2,table2.column1,table2.column2,...
FROM table1
RIGHT JOIN table2
ON table1.matching_column = table2.matching_column;
Or
SELECT table1.column1,table1.column2,table2.column1,table2.column2,...
FROM table1
RIGHT OUTER JOIN table2
ON table1.matching_column = table2.matching_column;
```

Example of a right join:

Consider the EMPLOYEE and DEPARTMENT tables as demonstrated in the previous examples. Suppose you want to generate a list of the employees and the departments they work in. You do not want to include any employees that are not assigned to a department, but you want to include all departments even if they have no employees yet.

To get the above result, the following query will be used to perform a right join between the EMPLOYEE table and the DEPARTMENT table based on the matching department IDs.

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

Table 2-10

emp_id	emp_name	dept_name
101	**Tom**	**Finance**
102	**Nir**	**Finance**
103	**Rick**	**HR**
104	**Maria**	**HR**
null	**null**	**Sales**

In the above result set, the employees from each department are included. It is worth noting that dept_name Sales with dept_id of 2 is also included in the result even though it has no employees.

Full Outer Join:

A full outer join or full join **returns all the rows from the left and right tables**. **It returns null for the non-matching rows from both the left and right tables.**

The syntax for a full join:

```
SELECT
table1.column1,table1.column2,table2.column1,table2.column2,...
FROM table1
FULL JOIN table2
ON table1.matching_column = table2.matching_column;

Or

SELECT table1.column1,table1.column2,table2.column1,table2.column2,...
FROM table1
FULL OUTER JOIN table2
ON table1.matching_column = table2.matching_column;
```

Example of a full join:

Consider the EMPLOYEE and DEPARTMENT tables demonstrated in the previous examples. Suppose you want to generate a list of all employees along with the department they work in. You want to include all departments even if they do not have any employees as well as all the employees even if they are not assigned to any department.

To get the above result, the following query will be used to perform a full join between the EMPLOYEE table and the DEPARTMENT table based on the matching department IDs.

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

Table 2-11

emp_id	emp_name	dept_name
100	Sam	null
101	Tom	Finance
102	Nir	Finance
103	Rick	HR

104	Maria	HR
null	null	Sales

In the above result set, all the employees of different departments are included. Note that even those departments that do not have any employees are included. Also, even those employees are included that have not been assigned to any department.

Cross Joins

A cross join **generates a paired combination of each row of the first table with each row of the second table.** A cross-join is also known as a cartesian join.

The syntax of a cross join:

```
SELECT column1, column2,...
FROM table1 CROSS JOIN table2;
```

Example of a cross join:

Consider the Shirts and Sizes tables given below.

Table 2-12

Shirts table	
id	shirt
1	T-shirt
2	Polo shirt
3	Collar shirt

Table 2-13

Sizes table	
id	size
1	Large
2	Medium
3	Small

Suppose you want to generate a list of all combinations of shirts with different sizes.

To get the above result, the following query will be used to perform the cross-join between the shirts table and the sizes table. The resulting table is sometimes called a cartesian product.

```
SELECT
   shirt, size
FROM shirts CROSS JOIN sizes;
```

Table 2-14

shirt	**size**

T-shirt	Large
T-shirt	Medium
T-shirt	Small
Polo shirt	Large
Polo shirt	Medium
Polo shirt	Small
Collar shirt	Large
Collar shirt	Medium
Collar shirt	Small

Self Joins

A self join allows you **to join a table to itself**. This allows you to explore relationships between rows in the same table.

The syntax of a self join:

```
SELECT
t1.column1,t1.column2,...,t1.columnN
t2.column1,t2.column2,...,t2.columnN
FROM tablex t1
[INNER|LEFT|RIGHT|FULL] JOIN tablex t2
ON t1.matching_column = t2.matching_column;
```

The query references the table TABLEX two times. The aliases t1 and t2 are used to assign the TABLEX table different names during the query.

Example of a self-join:

Consider the EMPLOYEE table as demonstrated in the previous examples. Suppose you want to generate a list of all employees and their manager's name if they have a manager. It is worth noting that the employee with emp_id as 100 and name as Sam is the CEO of the company, so he doesn't have a manager.

To get the above result, the following query will be used to perform an inner join using the emp_id of the EMPLOYEE table with the mgr_id of the same EMPLOYEE table. It is a self-join because the EMPLOYEE table is joined with itself.

```
SELECT
  e1.emp_id,
  e1.emp_name,
  e1.mgr_id,
  e2.emp_name as mgr_name
FROM employee e1
INNER JOIN employee e2
ON e1.mgr_id = e2.emp_id;
```


Table 2-15

emp_id	emp_name	mgr_id	mgr_name
101	Tom	100	Sam
102	Nir	101	Tom
103	Rick	100	Sam
104	Maria	103	Rick

Here, the record with emp_id 100 does not have a manager, so it is excluded from the result.

Watch the following videos by Ben Forta to learn about joining two or more tables.

Joining two tables in SQL video

Joining multiple tables in SQL video

ETL with delimited files;

A delimited file is a text file used to store data, in which each line represents a row or a record, and each row has fields that are separated by a delimiter.

A delimiter is a sequence of one or more characters that is used as a separator for the fields in a row of a delimited file. The following characters are common delimiters:

- Comma (,)
- Semicolon (;)
- Tab
- Space

Types of delimited files;

There are many types of delimited files, but the following are used very frequently.

- CSV file: CSV stands for Comma-Separated Values. It is a delimited file in which a comma (,) is used as a field delimiter.
- TSV file: TSV stands for Tab-Separated Values. It is a delimited file in which the TAB character is used as a field delimiter.

The two most common file extensions for delimited files are .csv and .tsv.

CSV Example

The content of a CSV file *users.csv* is given below:

```
id,name,age
1,John,35
2,San,30
3,Mary,31
```

The first line in the file is the header that contains the name of the fields.

There are four rows, including the header, in this CSV file. Each row has three fields that are separated by the delimiter comma (,).

TSV Example

The content of a TSV file *users.tsv* is as below:

id	name	age
1	John	35
2	San	30
3	Mary	31

There are four rows in this TSV file, including the header. Each row has three fields that are separated by the delimiter TAB. The first line in the file is the header that contains the name of the fields.

What is a text qualifier?

A text qualifier is a character used to specify the point at which the contents of a text field should begin and end. Consider that you need to import or read a comma-delimited file, but one of the fields contains a comma. In such a scenario, a text qualifier can be used to show that the comma should be included within the field and not be used as a delimiter.

The following is the content of a CSV file that contains the id, name, age, and skills. A user can have one or more skills that are again separated by a comma.

```
id,name,age,skills
1,John,35,Java
2,San,30,Java,Python
3,Mary,31,Python
```

The above file will be parsed out like this:

Table 2-16

id	**name**	**age**	**skills**	
1	John	35	Java	
2	San	30	Java	Python
3	Mary	31	Python	

After parsing the above CSV file using a comma delimiter, the skills of the user having id 2 are separated into two different fields.

To avoid this, we can use a text qualifier such as double quotes for skills as below.

```
id,name,age,skills
1,John,35,"Java"
2,San,30,"Java,Python"
3,Mary,31,"Python"
```

Now, the above file will be parsed out like this:

Table 2-17

id	**name**	**age**	**skills**

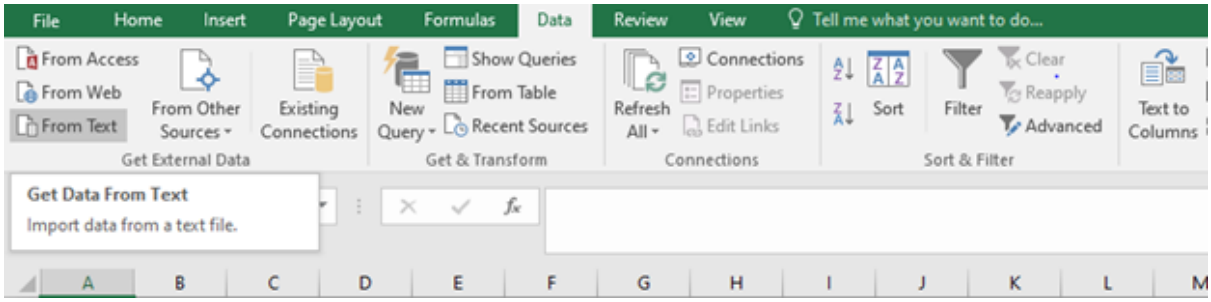
1	John	35	Java
2	San	30	Java, Python
3	Mary	31	Python

Importing Delimited Files to Excel

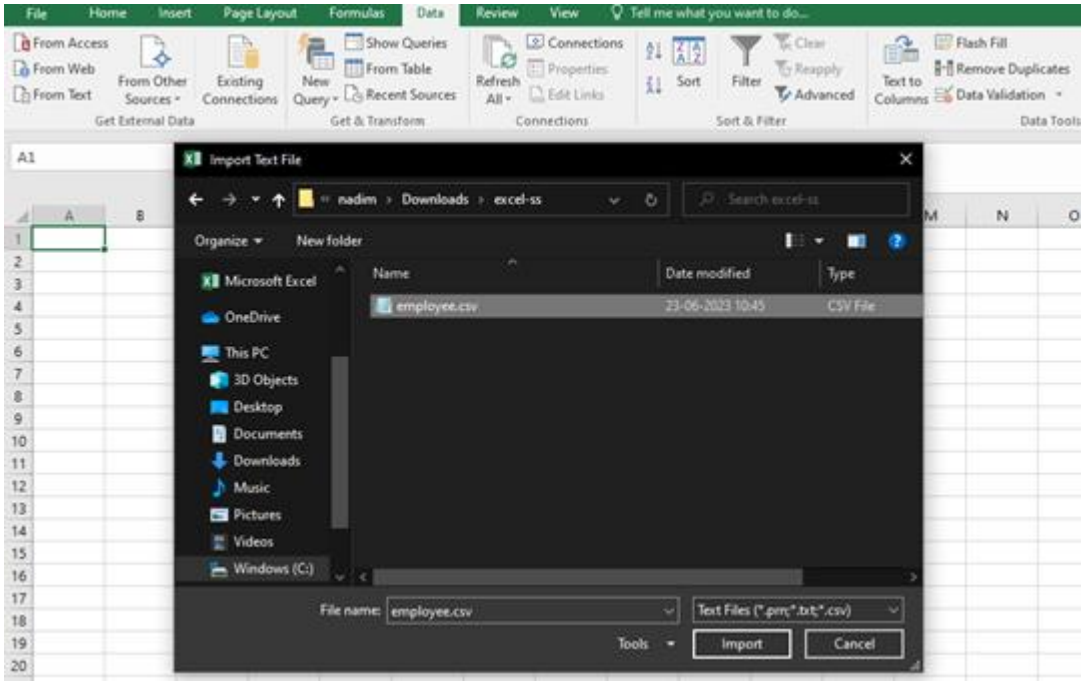
The following are the steps to import a delimited file (for example, a CSV file) to Excel.

1. Open Excel and click on Data -> From Text as below.

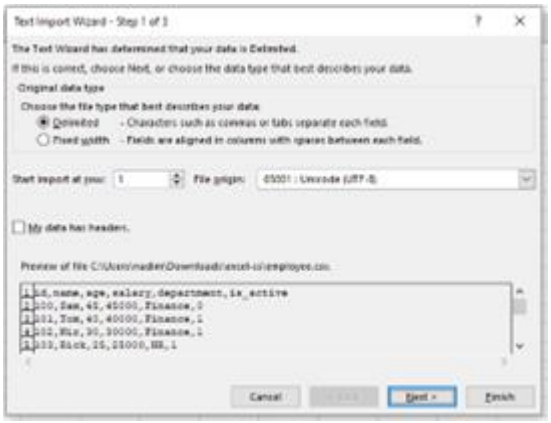
Figure 2-2



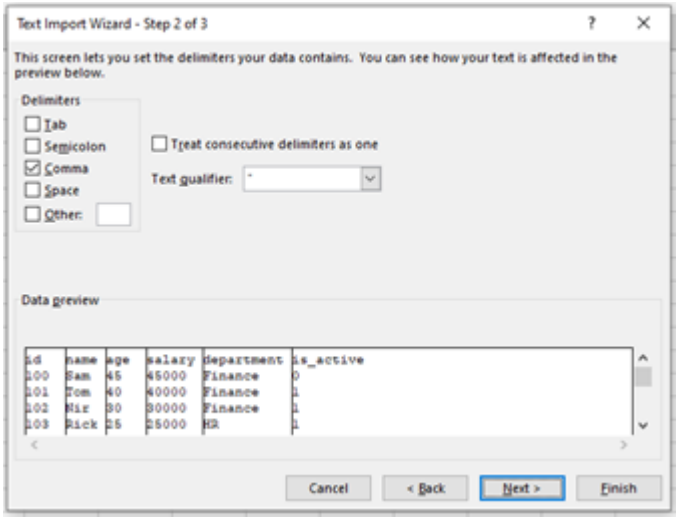
2. A dialog box will be opened to allow you to select the file. Select your delimited file and click on Import.



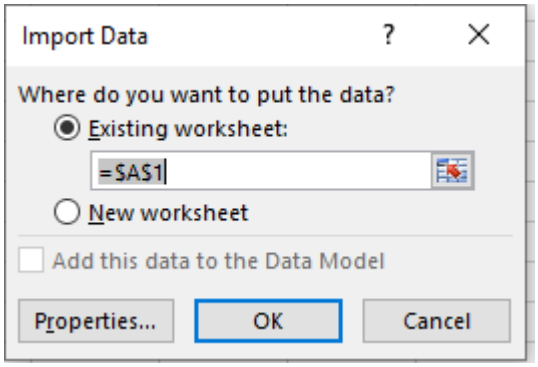
3. Excel will open a preview of the data in the selected file.



Choose the delimiter (for example, comma for a CSV file) and click **on Finish. Figure**



A new Import Data dialog box will be opened. Choose either Existing worksheet or New



File importation to Excel Video

Reading and Writing Delimited Files Using Python

There are various ways to read and write a delimited file using Python. The Python Pandas library provides methods to read and write a delimited file easily.

- read_csv() method is used to read a delimited file. It imports data in the form of a data frame. Refer to the official documentation of read_csv().

- to_csv() method is used to write a Pandas data frame to a delimited file. Refer to the official documentation to_csv().

A data frame is a two-dimensional data structure in which the data is organized in a tabular fashion having rows and columns.
Consider a CSV file *users.csv* with the following contents:

```
id,name,age
1,John,35
2,San,30
3,Mary,31
```

Use the following Python code example to read the *users.csv* file as a data frame and display its contents.

Table 2-18

Python code to read CSV file	**Output data frame**

import pandas
 	
# Read the CSV file 	
user_df = pandas.read_csv('users.csv')
 	
# Print the contents of the CSV file 	
print(user_df)	

	Id	**name**	**age**
0	1	John	35
1	2	San	30
2	3	Mary	31

Use the following Python code example to write the data frame to a file *output_users.csv*.

Table 2-19

Python code to write a CSV file

import pandas
 	
# Create a data frame. 	
users = {'id': [1, 2, 3], 'name': ['John', 'San', 'Mary'], 'age': [35, 30, 31]}
users_df = pandas.DataFrame(users)
 	
# Write to a CSV file 	
users_df.to_csv('output_users.csv', index=False)

Both Pandas read_csv() and to_csv() methods consider comma ',' to be the delimiter by default. However, you can specify any delimiter explicitly using the sep parameter. The following example uses the pipe '|' as a delimiter.

```
pandas.read_csv('users.txt', sep='|')
```

```
df.to_csv('out_users.csv', sep='|', index=False)
```

id,name,age
1,John,35
2,San,30
3,Mary,31

Use the following R code example to read the *users.csv* file and display its contents.

Table 2-20

R code to read CSV file

```
# Read the CSV file
csvFile =read.csv("users.csv")

# Print the contents of the CSV file
print(csvFile)
```

Output data frame

	Id	name	age
0	1	John	35
1	2	San	30
2	3	Mary	31

Use the following R code example to write a data frame to the *output_users.csv* file.

Table 2-21

R code to write a CSV file

```
# Create a data frame
users_df <- data.frame(id = c(1, 2, 3), name = c("John", "San", "Mary"), age = c(35,30,31))

# Write to a CSV file
write.csv(users_df, "output_users.csv", row.names=FALSE)
```

Both read.csv() and write.csv() methods in R consider comma ',' to be the default delimiter. However, you can specify any delimiter explicitly using the sep parameter in read.csv(), but write.csv() does not allow any other separator except comma. You can use the method write.table() instead of write.csv() to change the separator to something else.

The following example uses the pipe '|' as a delimiter and HEADER set to FALSE. Here are the contents of the users.txt file:

id|name|age

1|John|35

2|San|30

3|Mary|31

```
users_df = read.csv("users.txt", header = FALSE, sep='|')
```

```
write.table(users_df, "output_users.csv", row.names=FALSE, sep='|')
```

Reading and Writing Delimited Files Using R

The R language also provides methods to read and write delimited files.

- The read.csv() method is used to read a delimited file. It also imports data in a data frame similar to the Python pandas.read_csv() method.
- The write.csv() method is used to write a data frame to a delimited file.

Both methods consider comma ',' as a delimiter by default. But you can also specify any delimiter explicitly using the sep parameter.

Consider a CSV file *users.csv* with the following content.

File importation to R Video

ETL with XML files

An **XML file** is a plain text file that contains XML data. It usually has .xml as a file extension.

XML stands for eXtensible Markup Language. It is designed to store and transport structured data.

XML Structure

XML follows a tree structure that must contain a root element. The root element is the parent of all other elements in an XML document. An **XML element** is the basic building block of an XML document. An element must have a **start tag** and an **end tag**. Each XML element may contain text, attributes, or sub-child elements. All attribute values must be quoted with either single or double quotes.

In the following example, root is the parent of all other elements where <student> is the start tag and </student> is the end tag. The root element has two sub-elements named student. Each student element has one attribute: gender, and two sub-child elements: id and name. Note that the value of the attribute gender is quoted, but the values of the elements id and name are not quoted.

```
<root>
 <student gender="male">
  <id>10</id>
  <name>John</name>
 </student>
 <student gender="female">
  <id>20</id>
  <name>Mary</name>
 </student>
</root>
```

Reading an XML file using Python

There are various ways to read an XML file using Python. The Python Pandas library provides methods to read and write an XML file easily.

- read_xml() method is used to read an XML file. It imports data in the form of a data frame. Refer to the official documentation of the read_xml() method.
- to_xml() method is used to write a Pandas data frame to an XML file. Refer to the official documentation of the to_xml() method.

The lxml library must be installed to use the above methods using the following command.

```
pip install lxml
```

These methods are best designed to import shallow XML documents in the following format, which is the ideal fit for the two dimensions of a data frame having rows and columns.

The following is the content of a simple XML file.

```
<root>
 <row>
  <column1>data</column1>
  <column2>data</column2>
  <column3>data</column3>
  ...
```

```
  </row>
  <row>
    ...
  </row>
</root>
```

Consider a *users.xml* file with the following content:

```
<data>
  <row>
    <id>1</id>
    <name>John</name>
    <age>35</age>
  </row>
  <row>
    <id>2</id>
    <name>San</name>
    <age>30</age>
  </row>
  <row>
    <id>3</id>
    <name>Mary</name>
    <age>31</age>
  </row>
</data>
```

Use the following Python code example to read the *users.xml* file and display its contents.

Table 2-22

Python code	**Output data frame**
import pandas # Read the XML file as a data frame	<table><tr><th></th><th>id</th><th>name</th><th>age</th></tr><tr><td>0</td><td>1</td><td>John</td><td>35</td></tr></table>

xmlFile = pandas.read_xml('users.xml')
 	
# Print the contents of the data frame 	
print(xmlFile)
	<table><tr><td>1</td><td>2</td><td>San</td><td>30</td></tr><tr><td>2</td><td>3</td><td>Mary</td><td>31</td></tr></table>

Use the following Python code example to write the data frame to a file *output_users.xml*.

Table 2-23

Python code to write an XML file

import pandas
 	
# Create a data frame. 	
users = {'id': [1, 2, 3], 'name': ['John', 'San', 'Mary'], 'age': [35, 30, 31]}
users_df = pandas.DataFrame(users)
 	
# Write to an XML file 	
users_df.to_xml('output_users.xml', index=False)

ETL with JSON files

A **JSON file** is a plaintext file that contains JSON data. It usually has .json as the file extension.

JSON stands for JavaScript Object Notation. It is designed to store and transport structured data. A JSON file contains the data in a key-value mapping format within { } where the keys are always unique and must be a string. Each key and its value is separated by a colon(:). Every value in a JSON file has a data type. The values in a JSON file can be one of the following types.

- String
- Number
- JSON object
- Array
- Boolean
- Null

A string must be enclosed in quotes, therefore all the keys and all the values of the type string are enclosed in quotes.

The JSON object below contains three key-value pairs.

```
{
 "id":1,
 "name":"John",
 "age":35
}
```

Reading and writing JSON using Python

The Python Pandas library provides methods to read and write a JSON file.

- read_json() method is used to read a JSON file. It imports data in the form of a data frame. Refer to the official documentation of read_json().
- to_json() method is used to write a Pandas data frame to a JSON file. Refer to the official documentation to_json().

Consider a users.json file whose contents look like the below.

```
{"id":1,"name":"John","age": 35}
{"id":2,"name":"San","age": 30}
{"id":3,"name":"Mary","age" : 31}
```

Use the following Python code example to read the *users.json* file as a data frame and display its contents.

Table 2-24

Python code to read JSON file	**Output data frame**

```
import pandas

# Read the JSON file
user_df =pandas.read_json('users.json', lines=True)

# Print the contents of the JSON file
print(user_df)
```

	id	name	age
0	1	John	35
1	2	San	30
2	3	Mary	31

Here, the parameter lines=True is used because the JSON file contains multiple lines of JSON objects.

Use the following Python code example to write the data frame to a JSON file *output_users.json*.

Table 2-25

Python code to write a JSON file

import pandas
 	
# Create a data frame. 	
users = {'id': [1, 2, 3], 'name': ['John', 'San', 'Mary'], 'age': [35, 30, 31]}
users_df = pandas.DataFrame(users)
 	
# Write to a JSON file 	
users_df.to_json('output_users.json', orient='records')

Cleaning and validating data

Real-world data is not always clean and prepared enough for data analysis. The data can also be messy which can lead to false insights and poor decision-making. In a world of data-driven decision-making, it is very vital to ensure that your data is clean and prepared enough to get accurate insights for making better business decisions. Data cleaning improves data quality and overall productivity.

This skill covers how to:

- Perform data cleaning common practices
- Perform truncation
- Describe data validation

Data cleaning common practices

Data cleaning is a process of identifying and fixing incorrect, incomplete, duplicate, and erroneous data in a data set to make it correct, consistent, and usable for data analysis.

A small dataset can be cleaned manually, but it is not possible to manually clean a large dataset with millions of rows. Data can be cleaned using scripting languages, such as R and Python. Data stored in a relational database can be cleaned using SQL.

The process of data cleaning depends on the nature and use case of the dataset. The following are some common practices used in the data cleaning process.

1. Remove irrelevant data
2. Remove duplicate data
3. Remove unnecessary spaces
4. Handle inconsistent capitalization
5. Data type conversion

6. Handle missing or null values using imputation
7. Deal with outliers
8. Standardize data

Removing or filtering out irrelevant data

In most cases, only part of the dataset is relevant to our data analysis. In such cases, we either filter out or delete the irrelevant data and select only the part of the data that is relevant to us.

Consider the table EMPLOYEE that has six columns - id, name, age, salary, department, and is_active.

Table 2-26

EMPLOYEE Table					
id	**name**	**age**	**salary**	**department**	**is_active**
100	Sam	45	45000	Finance	0
101	Tom	40	40000	Finance	1
102	Nir	30	30000	Finance	1
103	Rick	25	25000	HR	1
104	Maria	35	35000	HR	1
105	Lara	50	50000	Sales	1
106	Mary	35	35000	HR	0

In this table, the value of the field is_active can be either 1 or 0. The value 1 denotes the employee is active, which means the employee is currently working in the company. The value 0 denotes the employee is inactive, which means the employee has left the company.

There are seven records in this table, out of which two are inactive. In such a scenario, we have the following two options.

- Filter out all the inactive employees during any calculation. The following query will filter out the inactive employees.

```
SELECT * FROM employee2 WHERE is_active != 0;
```

- Delete all inactive employees permanently from the table. The following query will delete all inactive employees permanently from the table.

```
DELETE FROM employee2 WHERE is_active = 0;
```

After performing any of the above operations, the resultant dataset will only contain the active employees as shown below.

Table 2-27

id	**name**	**age**	**salary**	**department**	**is_active**

101	Tom	40	40000	Finance	1
102	Nir	30	30000	Finance	1
103	Rick	25	25000	HR	1
104	Maria	35	35000	HR	1
105	Lara	50	50000	Sales	1

Removing duplicate records

t is very common to have duplicate records. Data can be collected and gathered from multiple different sources. It is very normal to have duplicates in the raw and unprocessed data.

Consider the below RAW_EMPLOYEE table that has two entries for Tom with the same id. In this table, the primary key is not enforced for id, so it can have duplicate values in the id column.

Table 2-28

RAW_EMPLOYEE Table				
id	**name**	**age**	**salary**	**department**
101	Tom	40	40000	Finance
101	Tom	40	40000	Finance
102	Nir	30	30000	Finance
103	Rick	25	25000	HR
104	Maria	35	35000	HR
105	Lara	50	50000	Sales

Such duplicate records should be removed to keep only one record for Tom. The following query will select only distinct records from the RAW_EMPLOYEE table.

```
SELECT DISTINCT * FROM raw_employee;
```

Removing unnecessary spaces

The unnecessary leading and/or trailing space can cause the same data to be considered different. For example, the values “male”, “ male”, “male ” and “ male ” are the same, but are considered different by a string comparison due to leading and/or trailing spaces. Extra spaces can be handled using the following SQL functions.

- TRIM() removes both leading and trailing spaces.
- LTRIM() removes only leading spaces.
- RTRIM() removes only trailing spaces.

Suppose the department column of the EMPLOYEE table contains values that have leading and/or trailing spaces, then the following query will remove the unnecessary spaces from the department values.

```
SELECT TRIM(department) FROM employee;
```

Handling inconsistent capitalization

Sometimes, the same data looks different to an algorithm or function that uses string comparison due to inconsistent capitalization. For example, in the EMPLOYEE table above, the department name can be entered as "Finance", “finance”, or “FINANCE”. While these are the same, they will be considered different entries due to differences in capitalization. This can be fixed by turning text into all uppercase or lowercase using UPPER() and LOWER() functions in SQL as below.

```
SELECT UPPER(department) FROM employee;
```

Data type conversion

Suppose the data type of the column age in the EMPLOYEE table is text and not numeric. In such a case, you need to convert the data type of the age column from text to numeric in order to do any numeric calculation on age such as finding the average age of an employee in the company.

The following query will convert the data type of age from text to INT.

```
SELECT CAST(age AS INT) AS age FROM employee;
```

Handling missing data or null values using imputation

In the real world, there may be some datasets that contain missing values. Missing values are generally represented as NULL, N/A, blanks, etc. Missing values are one of the most common problems in data analysis.

There are various ways to handle missing values.

- One way is to discard the records having missing values. But doing so may result in the loss of valuable information.
- A better way is to replace missing data with some substituted value. This technique is known as **imputation**.

The substituted value that is used to replace missing data is known as **imputed data**. The imputed data is derived from the existing part of the data.

The following are some of the popular methods of data imputation:

- **Imputation Using Mean or Median Values**: In this technique, the missing values in a column are replaced by the mean or median value of non-missing values in the same column. This method can be used only with numeric data.

    - **Mean** or **average**: It is the sum of all the numbers divided by the total number of numbers. For example, the mean of 5 numbers [9, 12, 8, 14, 7] is (9+12+8+14+7)/5 , i.e., 10.
    - **Median**: It is the middle number in the sorted list of numbers in ascending or descending order. For example, to find the median of 5 numbers [9, 12, 8, 14, 7], first sort the numbers in ascending order [7, 8, 9, 12, 14] and find the middle number, i.e., 9. Therefore 9 is the median value.

- **Imputation Using Most Frequent Values**: In this technique, the missing values in a column are replaced by the most frequent value of non-missing values in the same column. This method can be used for both numeric and non-numeric data.
- **Imputation Using Zero or Constant Values**: In this technique, the missing values of a column are replaced by zero or any other constant value. This method can be used for both numeric and non-numeric data.

Dealing with Outliers

An outlier is an extremely high or extremely low data value compared to the other data values in the dataset.
The outliers are considered abnormal data values but they should be investigated before eliminating them because they may be valuable to the data and the analysis. To investigate them, ask questions such as:

- Why did such data values appear?
- Is it a rare case, or is it likely to appear again?

Based on the investigation, a data analyst may either eliminate those data points or perform data imputation.

Standardizing data format

tandardizing data is a process of changing data into a consistent format. This is required in various scenarios like these:

- Changing temperature to either Fahrenheit or Celsius to have a consistent unit.
- Ensuring that all instances of a length measurement are given in the same unit (meters or kilometers).

For example, the following query will convert the length in kilometers to meters.

```
SELECT (length_km * 1000) AS length_meter FROM lengths;
```

Do you know how to remove non-printable characters?

Data is extracted from different applications. Sometimes non-printable characters are also included in the worksheet when data is imported or copied to a worksheet. Such unwanted characters are very frustrating and must be removed or cleaned from the worksheet for further analysis. But do you know how to remove such non-printable characters from your worksheet?

Hint: You can use the CLEAN function to remove non-printable characters from Excel.

Truncating

In data analytics, **truncation** is used to simplify data, for example, truncating a number to a specified number of decimal places.

Both SQL and Excel provide functions to truncate a number to a specified number of decimal places.

The syntax and example of the truncate function in Excel are as follows.

```
=TRUNC(N, D)
```

The TRUNC function takes two parameters where N is the number to be truncated and D is the number of decimal places up to which the number N is to be truncated.

For example, =TRUNC(12.18965, 2) will return 12.18.

The syntax and example of the truncate function in MySQL are as follows.

```
TRUNCATE(N, D)
```

The TRUNCATE function takes two parameters where N is the number to be truncated and D is the number of decimal places up to which the number N is to be truncated.

For example, TRUNCATE(12.18965, 2) will return 12.18

Note that truncating is different from rounding. The truncate function TRUNCATE(N, D) simply truncates the number N up to D number of decimal places but does not round it. There is another function ROUND(N, D) that rounds the number N to D number of decimal places.
For example, SELECT TRUNCATE(12.18965, 2) will return 12.18, but SELECT ROUND(12.18965, 2) will return 12.19.

Data validation

Data validation is performed after the data cleaning process to validate the data. During data validation, you take steps to ensure that the data is accurate, complete, consistent, and uniform.

Every organization has its own unique ways of storing, maintaining, and validating data. Some common examples of data validation rules followed across different organizations to maintain integrity and clarity are as shown below:

- Data completeness check to ensure that the required records are not missing
- Data type validation to verify that each field has the correct data type (For example integer, float, string)
- Range validation to ensure the values are in the correct range (eg., a number between 1-100)
- Uniqueness check (For example, in a relational database, the uniqueness can be ensured at the time of table creation by creating the primary key constraints for fields like employee_id in the EMPLOYEE table and department_id in DEPARTMENT table.)
- Consistent expressions (For example, the same department name should not have different values like “HR”, “H.R”, and “Hr”)
- No null values (For example, the field name in the EMPLOYEE table should not have null values).

Organizing data introduction

Data organization is the practice of categorizing and classifying data to make it easy to use. Consider the organization of books in a library where the books are arranged in different categories in the most logical and orderly fashion so that anyone can easily find the book that they're looking for.

This skill covers how to:

- Describe data organization
- Perform sorting
- Perform filtering
- Perform appending and slicing
- Perform pivoting
- Perform transposition

Data organization plays a vital role in managing and accessing data. When data is well organized in an Excel worksheet or in a database table, it allows users to access and process data easily and efficiently. It is very difficult to access and process data that is not well organized.

Data organization helps in categorizing and classifying data to make it more usable. Some of the following processes are used to organize data.

- Sorting data
- Filtering data
- Appending data
- Slicing data

It is easier to analyze data if it is in sorted order. Data can be previewed in sorted order, either in ascending or descending order, depending on the requirement. For example, if you are looking for year-over-year growth trends, you might sort the data in ascending order by date so that the oldest data is displayed first.

Data can be sorted easily whether it is in an Excel or a database table.

Sorting data in Excel

1. Select the data to sort. Select tabular data containing multiple rows, as well as single or multiple columns.
2. Select **Data** and look for **[A->Z]** and **[Z->A]** icons near Sort and Filter

Figure 2-5

3. Click **[A->Z]** to perform an ascending sort and **[Z->A]** to perform a descending order sort.

Sorting in Excel Video

Sorting Data in SQL

Syntax of ORDER BY:

```
SELECT column1, column2, ...
FROM table_name ORDER BY column_name [ASC|DESC];
```

The ASC keyword is used to return the result in ascending order, but it is optional as the ORDER BY returns results in ascending order by default. The DESC keyword is used to return the result in descending order.

The following query will return all records from the EMPLOYEE table in ascending order of the employee's name.

```
SELECT * FROM employee ORDER BY name;
```

Table 2-29

id	name	age	salary	department
105	Lara	50	50000	Sales
104	Maria	35	35000	HR
102	Nir	30	30000	Finance
103	Rick	25	25000	HR
101	Tom	40	40000	Finance

Watch the following videos by Ben Forta to learn about sorting in SQL.

Sorting in SQL using ORDER BY Video

Sorting by one column in SQL Video

Sorting by multiple columns in SQL Video

Specify sort direction in SQL Video

Filtering

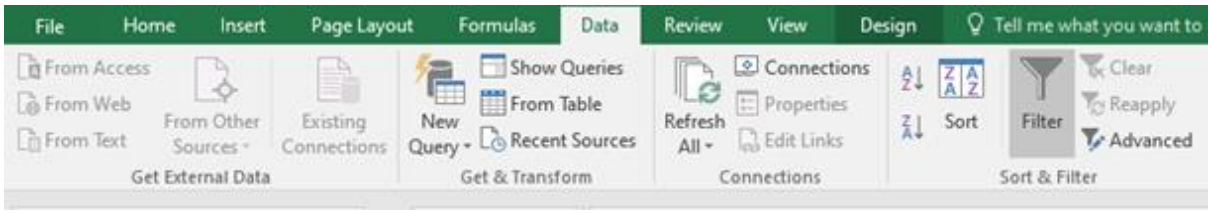
Filtering is a way to extract only those records that satisfy a specified condition. Consider that an employee's records are stored in a table or an Excel sheet and you want to view those records that belong to the HR department. In such cases, you can apply the filter to extract only the relevant records.

Filtering data in Excel

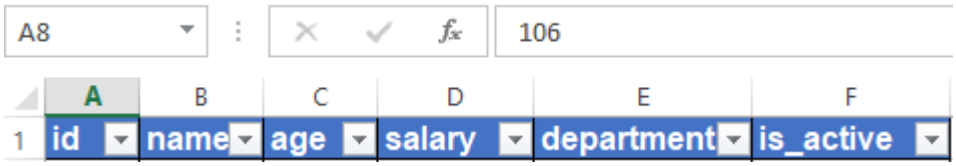
Follow these steps to filter data in Excel.

1. Select any cell within the range of your dataset.

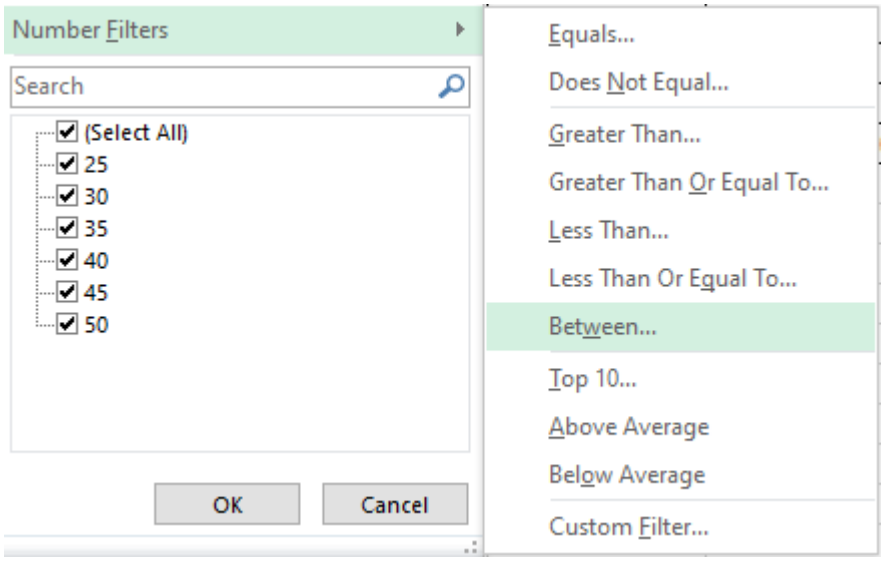
2. Select **Data** and then click on **Filter**.



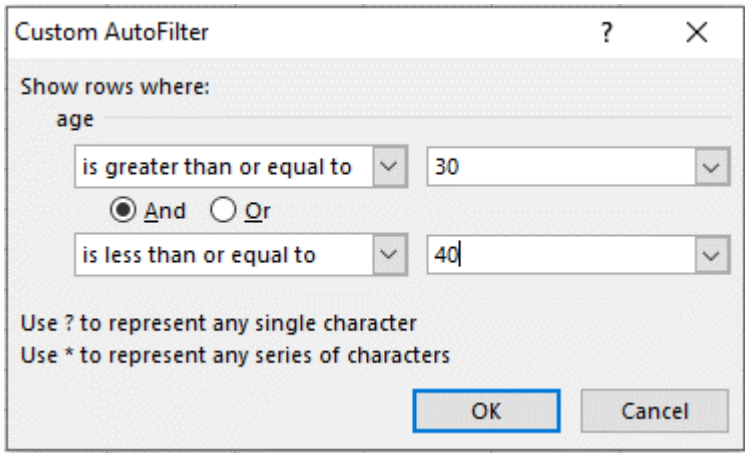
3. After clicking on **Filter**, the column header will show arrow icons as shown below. Select any of the column header arrows to filter the data on that column.



4. Select **Text Filters** or **Number Filters**, and then select a comparison, like **Between**.



5. Enter the filter criteria and click on OK.



Watch the following videos by Bill Jelen to learn more about filtering in Excel.

Filtering in Excel Video

Filter by selection in Excel Video

Filtering data in SQL

In SQL, the WHERE clause is used to filter records.

Syntax of WHERE clause

```
SELECT * FROM table_name WHERE condition(s);
```

Consider the table EMPLOYEE as shown below.

Table 2-30

id	name	department
101	Tom	Finance
102	Nir	Finance
103	Rick	HR
104	Maria	HR
105	Lara	Sales

The following query will return all records from the EMPLOYEE2 table where the department is HR.

```
SELECT * FROM employee2 WHERE department='HR';
```

Table 2-31

id	name	department
103	Rick	HR
104	Maria	HR

Data operators in WHERE clause

In the above example, we used the = operator to check if the department is equal to HR. Following is the list of operators that can be used in the WHERE clause.

Operator	Meaning
=	Equal to
!= or <>	Not equal to
>	Greater than

<	Less than
<=	Less than or equal to
>=	Equal to or greater than
BETWEEN	To select items within a specified range where the start and end items are inclusive
LIKE	To search for a pattern
IN	To specify multiple possible values for a column to include
NOT IN	To specify multiple possible values for a column to exclude

The following query will pull all records from the EMPLOYEE table where the name begins with the character 'R'. Here, the wildcard character % is being used.

Wildcard characters are used with the LIKE operator to substitute one or more characters in a string. The character underscore _ is used to substitute a single character and the character % is used to substitute zero or more characters.

```
SELECT * FROM employee2 WHERE name LIKE 'R%';
```

Table 2-33

id	name	department
103	Rick	HR

The following query will return all records from the EMPLOYEE table where the id is between 103 and 105, inclusive of the beginning and end values.

```
SELECT * FROM employee2 WHERE id BETWEEN 103 AND 105;
```

Table 2-34

id	name	department
103	Rick	HR
104	Maria	HR
105	Lara	Sales

The following query will return all records from the EMPLOYEE table where the id is either 103, 104, or 105.

```
SELECT * FROM employee2 WHERE id IN (103,104,105);
```

Table 2-35

id	name	department
103	Rick	HR
104	Maria	HR
105	Lara	Sales

The following query will return all records from the EMPLOYEE table where the id is not 103, 104, or 105.

```
SELECT * FROM employee2 WHERE id NOT IN (103,104,105);
```

Table 2-36

id	name	department
101	Tom	Finance
102	Nir	Finance

AND and OR operators:

The AND and OR operators are used to filter records based on more than one condition in the WHERE clause.

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

The following query will return all records from the EMPLOYEE table where the id is greater than 103 and the department is HR.

```
SELECT * FROM employee2 WHERE id>103 AND department='HR';
```

Table 2-37

id	department	
104	Maria	HR

The following query will return all records from the EMPLOYEE table where the id is greater than 103 or the department is HR.

```
SELECT * FROM employee2 WHERE id>103 OR department='HR';
```

Table 2-38

id	name	department
103	Rick	HR
104	Maria	HR
105	Lara	Sales

Watch the following videos by Ben Forta to learn about filtering data in SQL.

Use WHERE to filter data in SQL Video

WHERE clause operators in SQL Video

Appending and slicing strings

Appending is used to combine two or more strings together. In SQL, the + operator as well as the CONCAT() function are used to combine strings.

The following queries combine the two strings 'Data' and ' Analyst' together.

Table 2-39

Query	Output
SELECT CONCAT('Data', ' Analyst');	Data Analyst
SELECT 'Data' + ' Analyst';	Data Analyst

Slicing strings

Slicing is used to extract a subset of elements from a string. In SQL, the SUBSTRING() function is used for slicing.
Syntax of the SUBSTRING() function in SQL:

```
SUBSTRING(string, start, length)
```

It takes three required parameters as follows.

- string - It is the string to extract from.
- start - It is the start position. The first position in the string is 1.
- length - It is the number of characters required to extract.

The following query extracts the first four characters from the string 'Data Analyst'.

Table 2-40

Query	Output
SELECT SUBSTRING('Data Analyst', 1, 4);	Data

Slicing a dataset

In data analytics, **slicing** refers to extracting a small portion of data from a larger data set to analyze and have a deeper understanding of data.

Consider there are thousands of rows in a table EMPLOYEE and you want to know how the data looks for your analysis or better understanding of data. So, instead of fetching all the rows from the table, you can simply extract a portion of rows using the WHERE condition, TOP, or LIMIT clause according to your requirements.

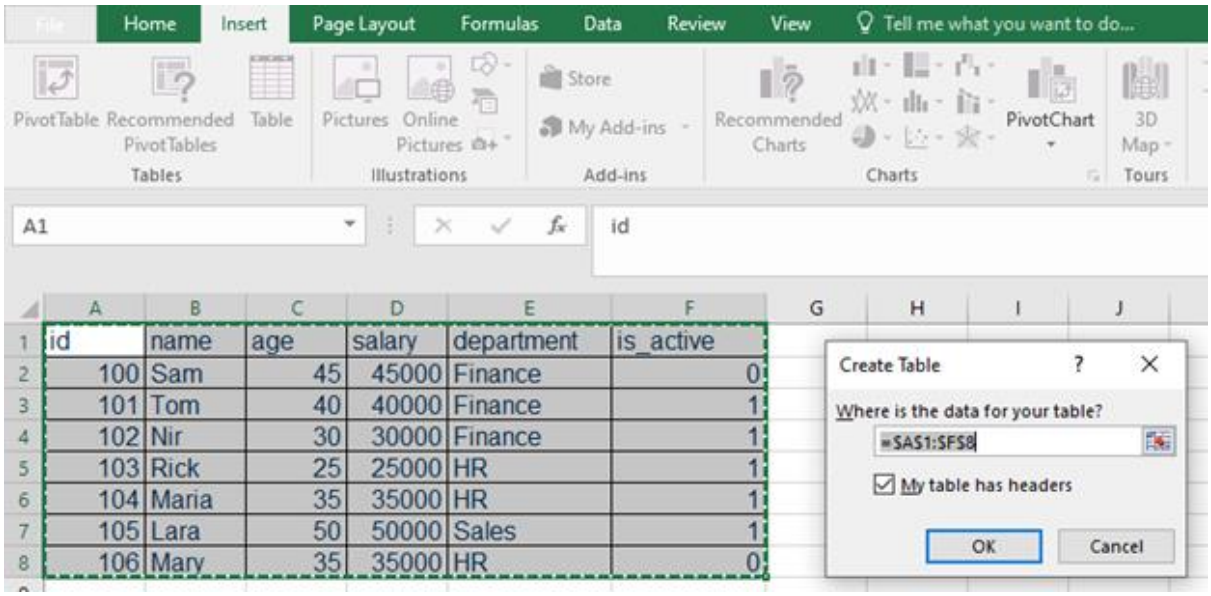
The following MySQL query will return a maximum of 10 employee records whose dept_id is 3.

```
SELECT * FROM employee WHERE dept_id = 3 LIMIT 10;
```

Slicing data in Excel

Microsoft Excel also provides slicers to filter data in a table. The following steps are used to filter data using a slicer in a table in Excel.

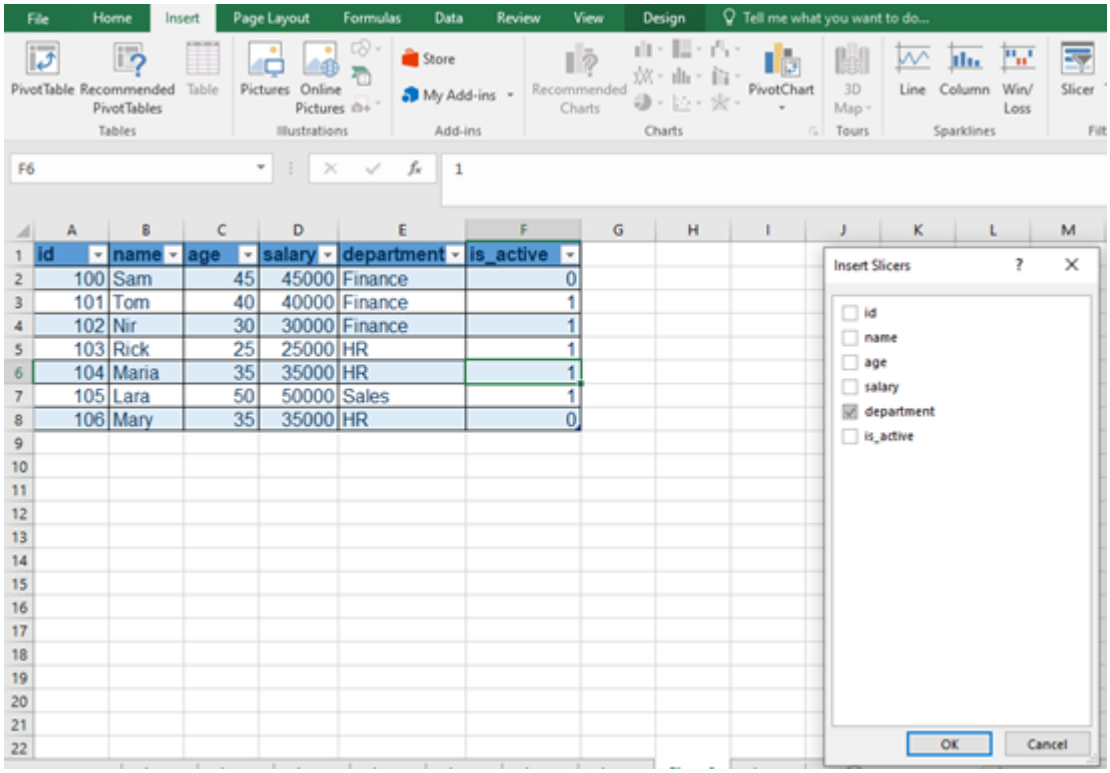
1. Select all the data in Excel and format it as a table **(Insert -> Table)**. Check **My Table has Headers** and click **OK**,



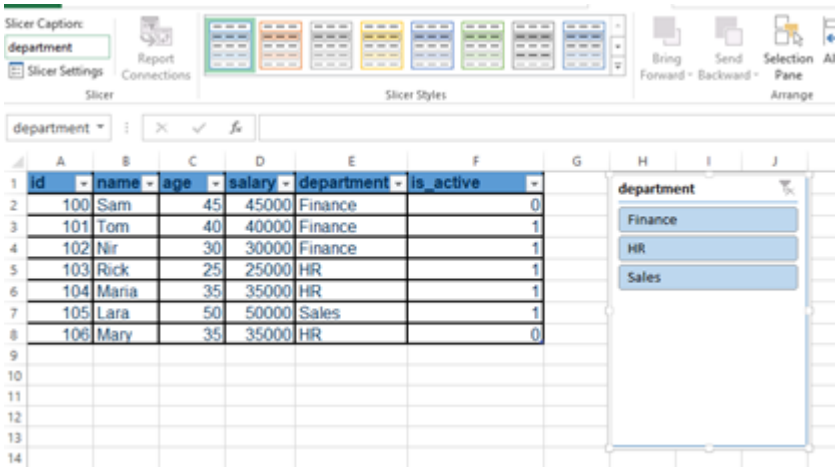
id	name	age	salary	department	is_active
100	Sam	45	45000	Finance	0
101	Tom	40	40000	Finance	1
102	Nir	30	30000	Finance	1
103	Rick	25	25000	HR	1
104	Maria	35	35000	HR	1
105	Lara	50	50000	Sales	1
106	Mary	35	35000	HR	0

2. Click anywhere in the table and select **Insert** -> **Slicer**

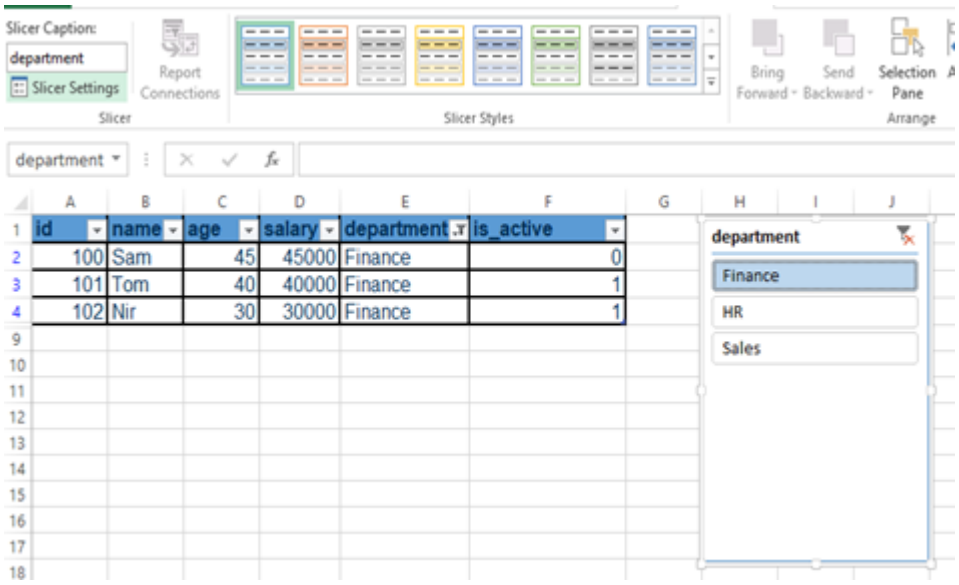
3. A dialog box for **Insert Slicers** will open in which you need to select the fields that you want to use to slice data and then select **OK**.



4. For each of the selected fields, a slicer will be created and each slicer will have buttons corresponding to the distinct values in the selected field.



5. When any of the slicer buttons is clicked, then only the matching rows in the linked table will be shown.



Transposition

In data analytics, transposition is the process of changing the data layout from rows to columns or columns to rows for making observations from a different perspective.

Look at the following two tables **Table 2-41** and **Table 2-42**. Both tables show the same report i.e, total, min, max and average salary in each department, but in a different view.

Table 2-41 is the transposition of **Table 2-42** and vice versa.

- The department names (Finance, HR, and Sales) are rows in **Table 2-41**, but columns in **Table 2-42**.
- The total_salary, max_salary, min_salary, and average_salary are column names in **Table 2-41**, but rows in **Table 2-42**.

Table 2-41

department	total_salary	min_salary	max_salary	average_salary
Finance	70000	30000	40000	35000
HR	60000	25000	35000	30000
Sales	50000	50000	50000	50000

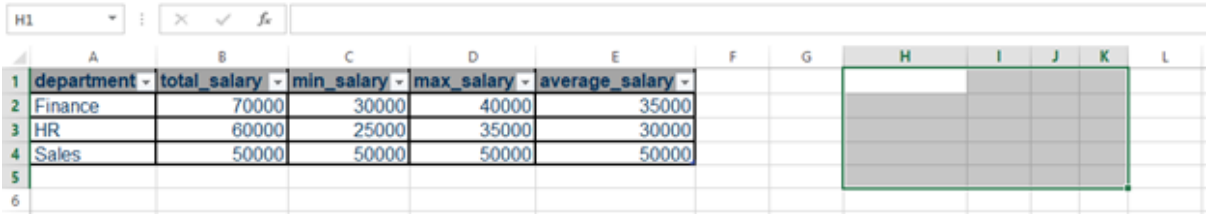
Table 2-42

department	Finance	HR	Sales
total_salary	70000	60000	50000
min_salary	30000	25000	50000
max_salary	40000	35000	50000
average_salary	35000	30000	50000

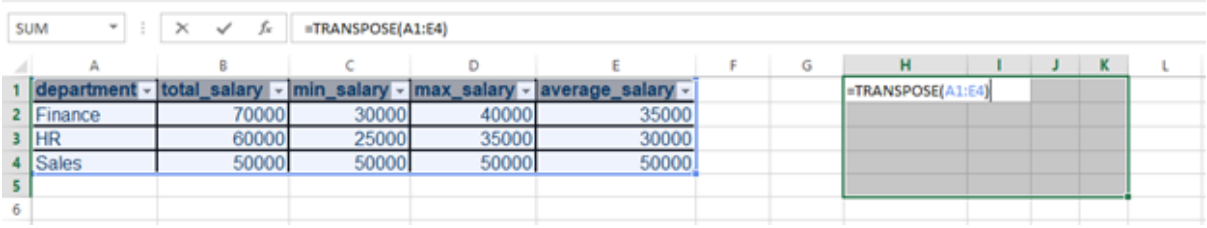
Excel provides the function TRANSPOSE to transpose a table. The following are the steps to create a transposition of a table.

1. Select blank cells where you would like the transposed table to be created.

Count the number of rows and columns in your original table. Suppose there are 4 rows and 5 columns in your table, then select blank cells with 5 rows and 4 columns.



2. After selecting blank cells, type the transpose formula =TRANSPOSE(A1:E4). Here, A1 is the first cell and E4 is the last cell of the original table.



3. After writing the transpose formula, press **ENTER**, which will generate a transposed table as below.

Aggregating data introduction

The aggregation of data is one of the most important aspects of data analytics. It is useful in knowing the summary of the data. Consider that you want to know the total number of employees as well as

the maximum, minimum, and average salary of employees working in your organization. In order to find these details, you need to apply the appropriate aggregation functions on the employee records stored in the database.

This skill covers how to:

- Describe the aggregation function
- Use aggregation functions like COUNT, SUM, MIN, MAX, and AVG in SQL
- Use GROUP BY and HAVING in SQL

Aggregate functions

An **aggregate function** is a function that performs a calculation on multiple values and returns a single value. Some of the most common and frequently used aggregation functions are as follows:

Table 2-43

COUNT	Returns the number of records.
SUM	Returns the total sum of values in a numeric column.
MIN	Returns the smallest value in a column.
MAX	Returns the largest value in a column.
AVG	Returns the average of all the values in a column.

NULL values are ignored when performing most aggregate functions. However, NULL values in the COUNT function are not ignored.

Using aggregate functions in SQL

An aggregate function in SQL returns a single value that is calculated from the multiple values in a column. The following are some of the frequently used aggregation functions in SQL:

- COUNT()
- SUM()
- MIN()
- MAX()
- AVG()

The COUNT() function returns the number of records that satisfy a specific condition.

Syntax of COUNT function:

SELECT COUNT(*column_name*)FROM *table_name* WHERE *condition*;

The SUM() function returns the sum of values in a numeric column.

Syntax of SUM function:

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

The MIN() function returns the smallest value in a column.

Syntax of MIN function:

```
SELECT MIN(column_name)FROM table_name WHERE condition;
```

The MAX() function returns the largest value in a column.

Syntax of MAX function:

```
SELECT MAX(column_name)FROM table_name WHERE condition;
```

The AVG() function returns the average of all the values in a column.

Syntax of AVG function:

```
SELECT AVG(column_name)FROM table_name WHERE condition;
```

Consider the table EMPLOYEE2 below that has five columns - id, name, age, salary, and department.

Table 2-44

id	name	age	salary	department
101	Tom	40	40000	Finance
102	Nir	30	30000	Finance
103	Rick	25	25000	HR
104	Maria	35	35000	HR
105	Lara	50	50000	Sales

The following query will return these values:

- The total number of employees
- The sum of the salaries of all the employees
- The minimum salary of the employees
- The maximum salary of the employees
- The average salary of the employees

```
SELECT
    count(*) as total_employee,
    sum(salary) as total_salary,
    min(salary) as min_salary,
    max(salary) as max_salary,
```

```
avg(salary) as average_salary
FROM employee2;
```

The above query will return the following result:

Table 2-45

total_employee	total_salary	min_salary	max_salary	average_salary
5	180000	25000	50000	36000

Using GROUP BY and HAVING in SQL

The GROUP BY statement groups rows in different categories where the aggregation functions can be applied on the rows of a category independently. For example, you can find the number of employees in each department using the GROUP BY statement on the department.

The HAVING clause is used to filter grouped data using conditions calculated with the aggregate functions. It is used along with the GROUP BY statement.

Syntax of GROUP BY:

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s);
```

Syntax of GROUP BY with HAVING:

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

One thing to keep in mind when using aggregates in SQL is that any column in the SELECT list must be either an aggregate or be listed in the GROUP BY clause.

Consider the table EMPLOYEE2 in the previous example.

- The following query will return the number of employees in each department.

```
SELECT department, COUNT(*) as total FROM employee2 GROUP BY department;
```

The above query will return the following result:

Table 2-46

department	total

Finance	2
HR	2
Sales	1

- The following query will return the number of employees in each department where the number of employees in the department is more than 1.

```
SELECT
  department,
  COUNT(*) as total
FROM employee2 GROUP BY department HAVING count(*)>1;
```

The above query will return the following result:

Table 2-47

department	**total**
Finance	2
HR	2

- The following query will return the total number of employees as well as the minimum, maximum, and average salary of the employees in each department.

```
SELECT
  department,
  COUNT(*) as total_employee,
  MIN(salary) as min_salary,
  MAX(salary) as max_salary,
  AVG(salary) as average_salary
FROM Employee2 GROUP BY department;
```

The above query will return the following result:

Table 2-48

department	**total_employee**	**min_salary**	**max_salary**	**average_salary**
Finance	2	30000	40000	35000
HR	2	25000	35000	30000
Sales	1	50000	50000	50000

Watch the following videos by Ben Forta to learn about aggregation in SQL.

Using GROUP BY to group data in SQL Video

Using HAVING to filter items in a group of data in SQL Video

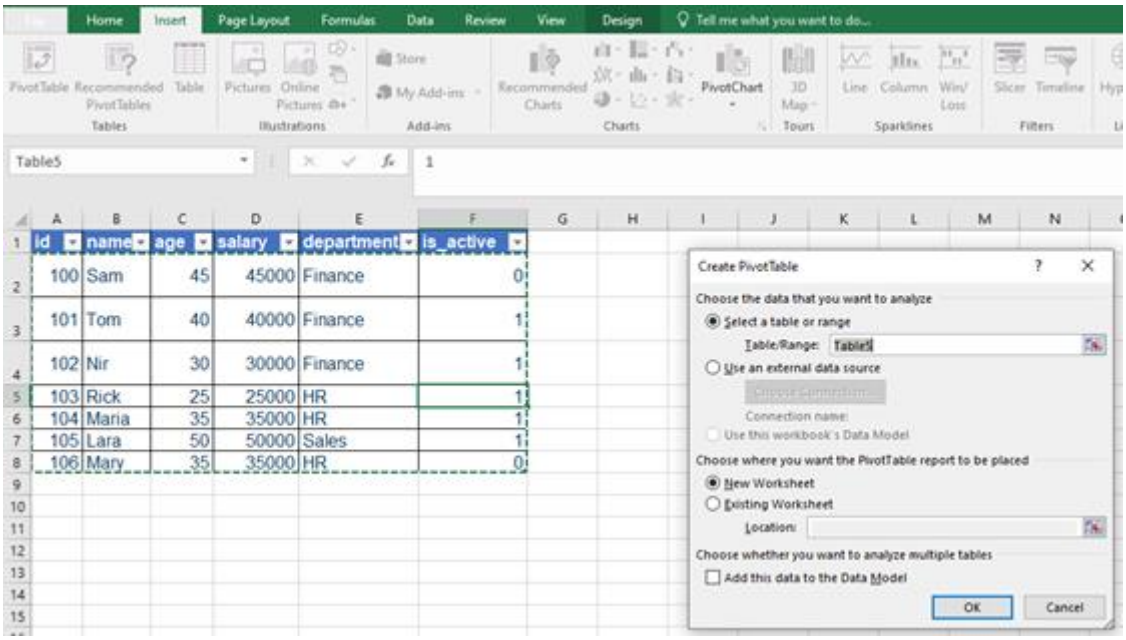
Pivoting

Consider there is a large dataset containing the employees records stored in a table and you want to find the number of employees in each department. In the aggregate data example above, **Table 2-46** shows the number of employees in each department is the summary information calculated from the employee table **Table 2-44**. This is called pivoting.

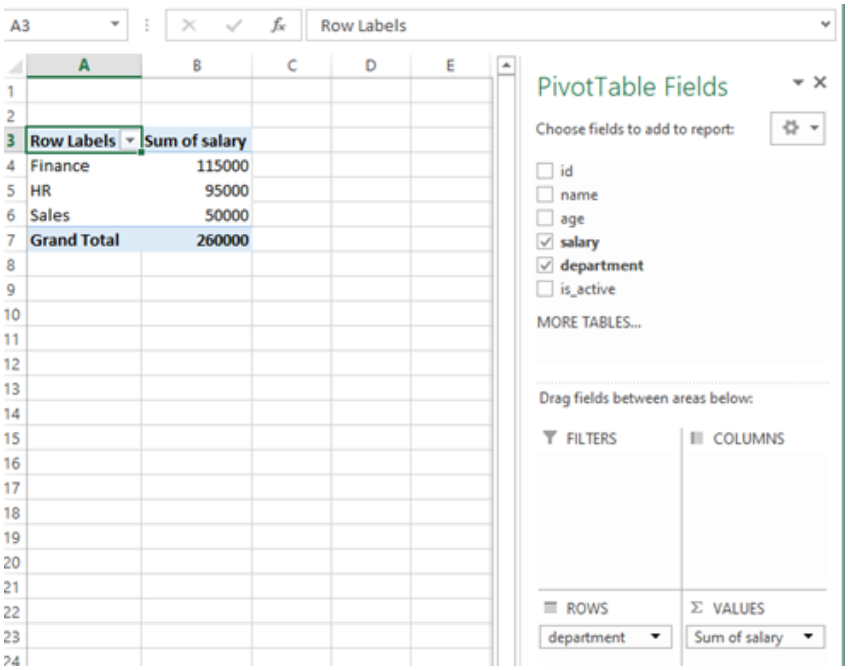
In data analytics, pivoting is a process used to summarize data in a large dataset stored in a table so that an analyst can see trends in the data.

Excel provides a powerful tool called **PivotTable** that can be used to calculate and summarize data for analysis. The following are the steps to create a pivot table in Excel.

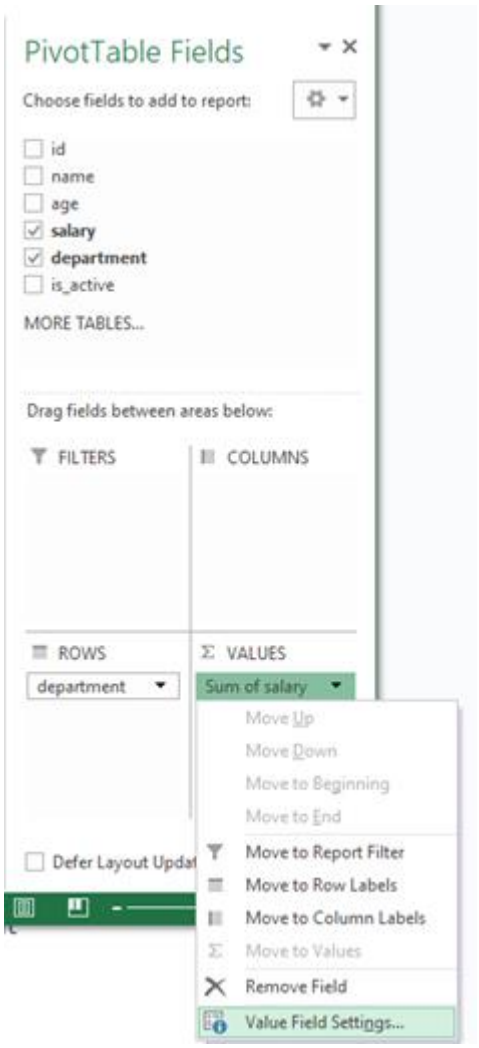
1. Click anywhere in the table and **select Insert -> PivotTable**
2. The Create PivotTable dialog will be opened. Click **OK**



3. A new worksheet will be opened that allows you to select the pivot fields. Here, we have selected the department and salary field. By default, it is showing the sum of the salary of each department.



4. By default, the calculation is the sum, but it can be changed to the count, min, or max.
    - Select **Value Field Settings** for the **VALUE** dropdown.



- Choose your calculation (Sum, Count, Average, Max, Min, Product) from the **Value Field Settings** dialog box and click **OK**

Summary

- Data manipulation is the process of modifying or changing data to make it more clean, organized, and readable for data analysis.
- ETL stands for extract, transform, and load. It is a three-step process in which we first extract data from a source system, perform transformations on that data, and then load it into a target system.
- Data cleaning is a process of identifying and fixing incorrect, incomplete, duplicate, and erroneous data in a dataset to make it correct, consistent, and usable for data analysis.
- Data validation is performed after the data cleaning process to ensure that the data is accurate, complete, consistent, and uniform.
- Data organization is the practice of categorizing and classifying data to make it easy to use.
- Data aggregation is useful in knowing the summary of the data.
- The aggregate functions are used to perform a calculation on multiple values and return a single value. Some of the most common and frequently used aggregation functions are sum, count, min, max, and avg.