

SESSION 2 | BASIC DATABASE OPERATIONS- I

SESSION OVERVIEW:

By the end of this session, the students will be able to:

- Understand how to import datasets in MySQL Workbench.
- Understand the datasets that will be utilized throughout the module using SELECT statements.
- Understand the importance of the DESCRIBE function.
- Understanding different data types that are used in SQL.
- Understanding WHERE clause in SQL.

KEY TOPICS AND EXAMPLES:

The following datasets will be used to showcase the importing procedure:

1. [Customers](#)
2. [Calendar](#)
3. [Product_Categories](#)
4. [Product_Subcategories](#)
5. [Products](#)
6. [Returns](#)
7. [Territories](#)
8. [Sales 2015](#)
9. [Sales 2016](#)
10. [Sales 2017](#)

Additionally, here is a [link](#) for the SQL file (.sql) for the above datasets.

NOTE: The above datasets need to be downloaded from the above links (preferably the SQL schema). The datasets need to be imported into your MySQL Workbench according to the guide mentioned below. The procedure of importing data will be explained in the class by the instructor.

Understanding the importing of datasets in MySQL Workbench:

1. **Different types of files can be imported into MySQL Workbench:**
 - a. **SQL Files(.sql):**

Importing SQL files is common for database backups or migrations. SQL scripts can create tables, insert data, and perform database schema changes.
 - b. **CSV Files (.csv):**

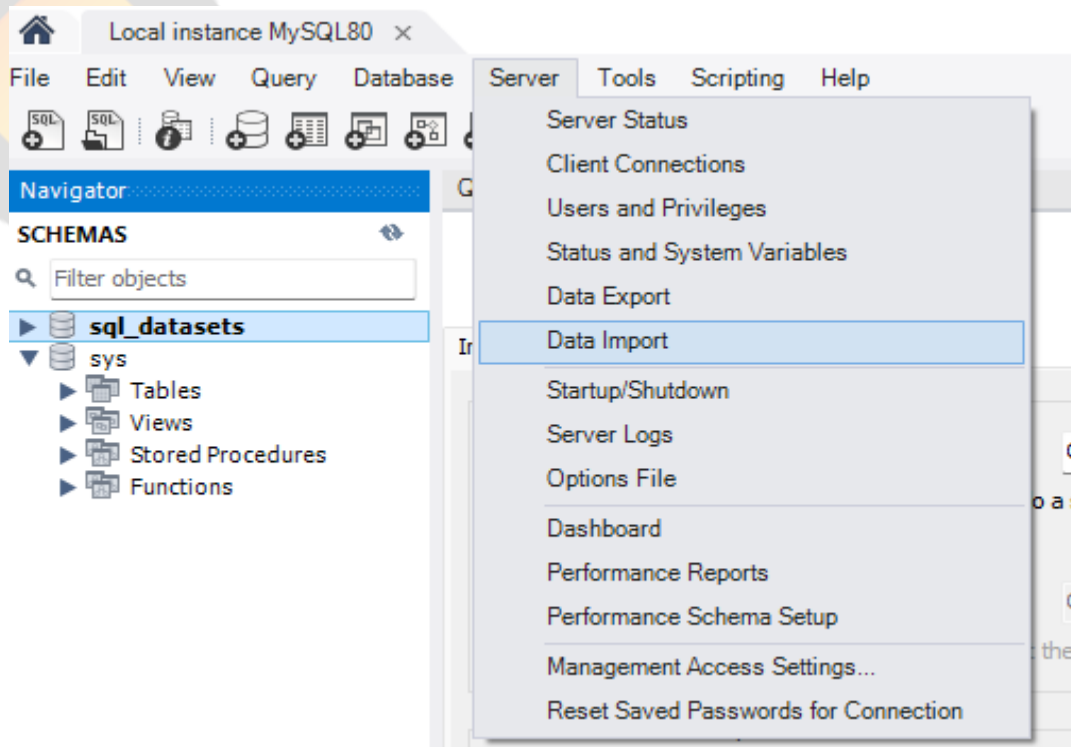
Comma-separated values files are used for importing data into tables. CSVs are straightforward for importing large datasets or transferring data between different applications.
 - c. **Excel Files (.xls, .xlsx):**

Directly importing data from Excel files, allowing users to bring data stored in spreadsheets into MySQL tables.

2. Steps to import data in MySQL Workbench:

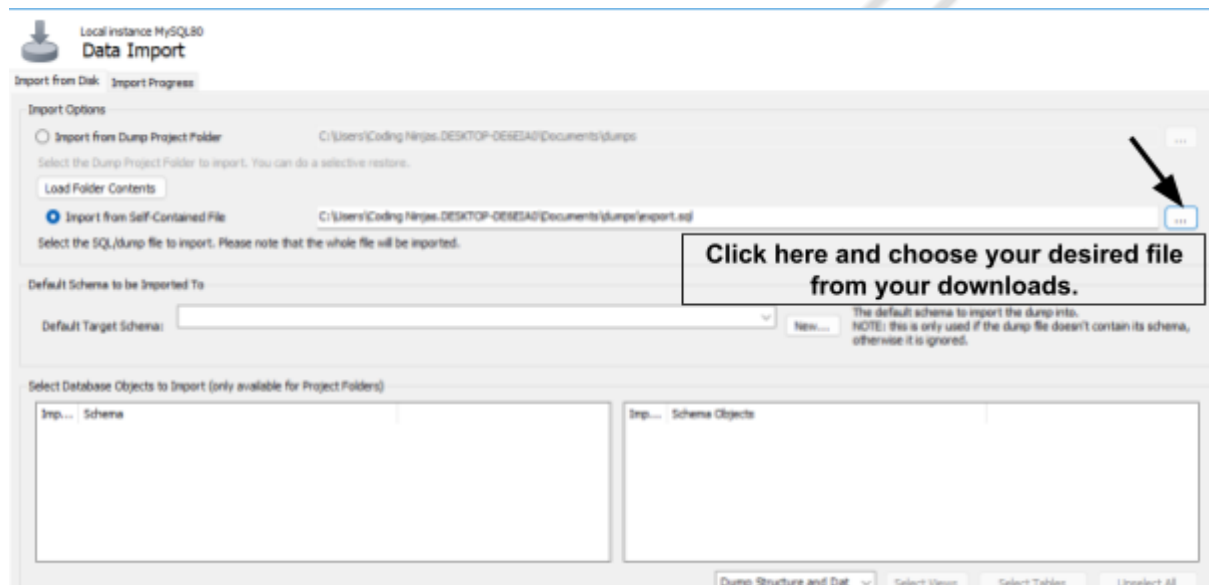
To import the data into MySQL Workbench when SQL schemas are provided:

- a. Launch MySQL Workbench and connect to your desired MySQL server instance.
- b. Navigate to the Data Import Wizard
 - i. From the top menu, click on Server.
 - ii. Select Data Import from the dropdown.

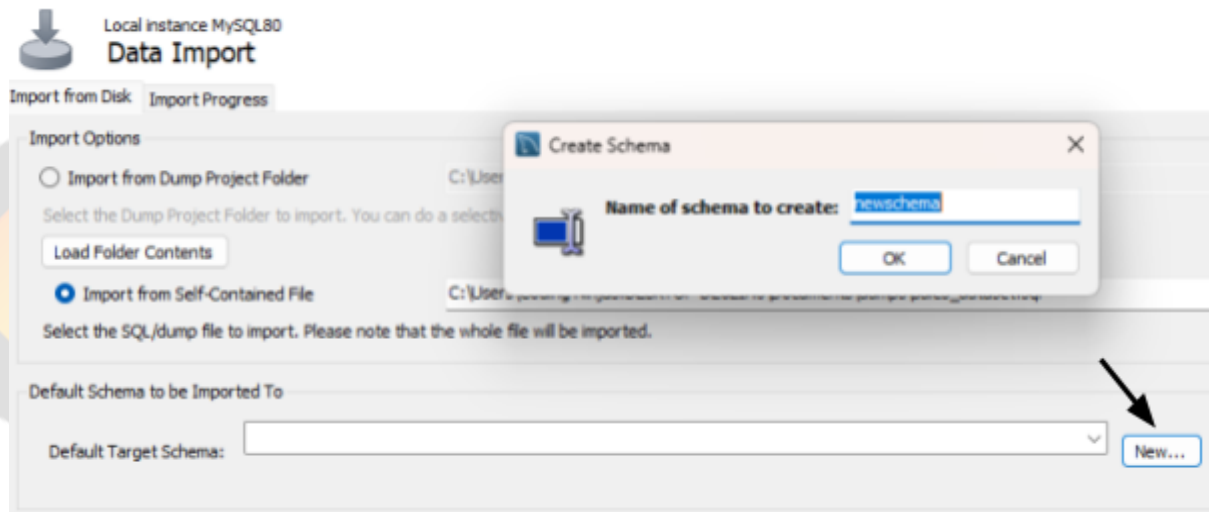


c. In the Data Import Wizard window:

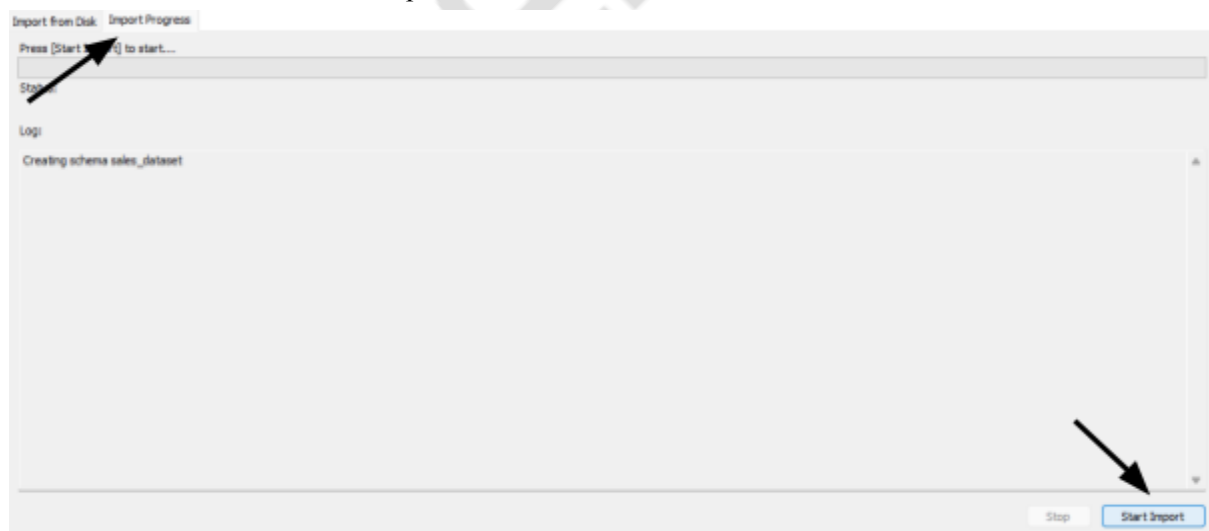
- i. Select Import from Self-Contained File.
- ii. Click Browse and locate your .sql file.



- d. If the file contains a dump of a specific database, ensure you select or create the database/schema where the file should be imported. Use the Default Target Schema dropdown or click New... to create one.

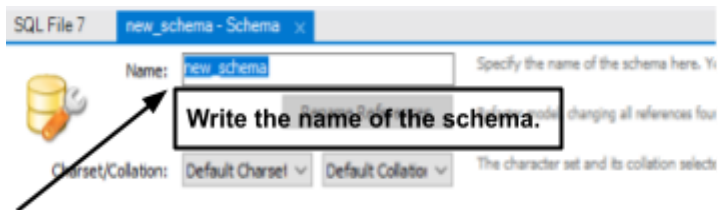


- e. Configure Import Options
 - i. Check the Dump Structure and Data option to import both the schema and its data.
 - ii. If the .sql file only contains structure or data, choose the appropriate option.
- f. Click the Start Import button at the bottom.



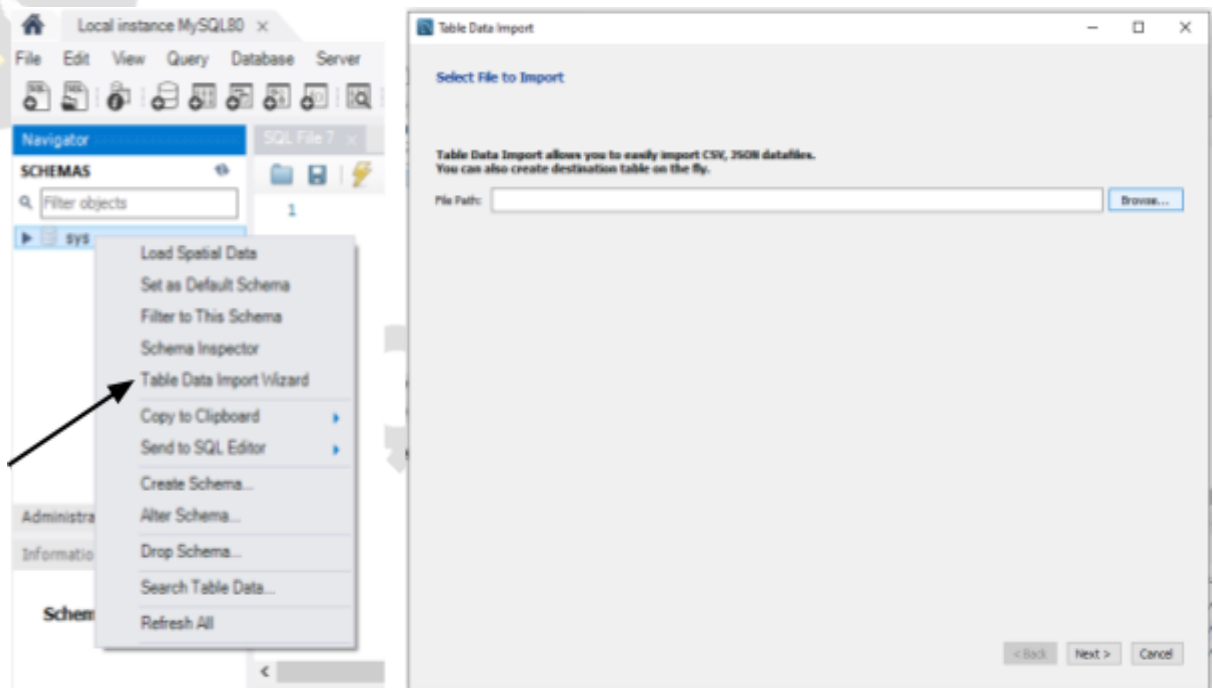
To import the data into MySQL Workbench, follow these steps for each CSV file:

- a. Open MySQL Workbench and connect to your MySQL server where you wish to import the data.
- b. **Create a New Schema:**
 - i. Navigate to the navigator panel on the left side, and right-click on 'Schemas'.
 - ii. Choose 'Create Schema...' and name it appropriately.
 - iii. Click 'Apply' at the bottom to create the schema.

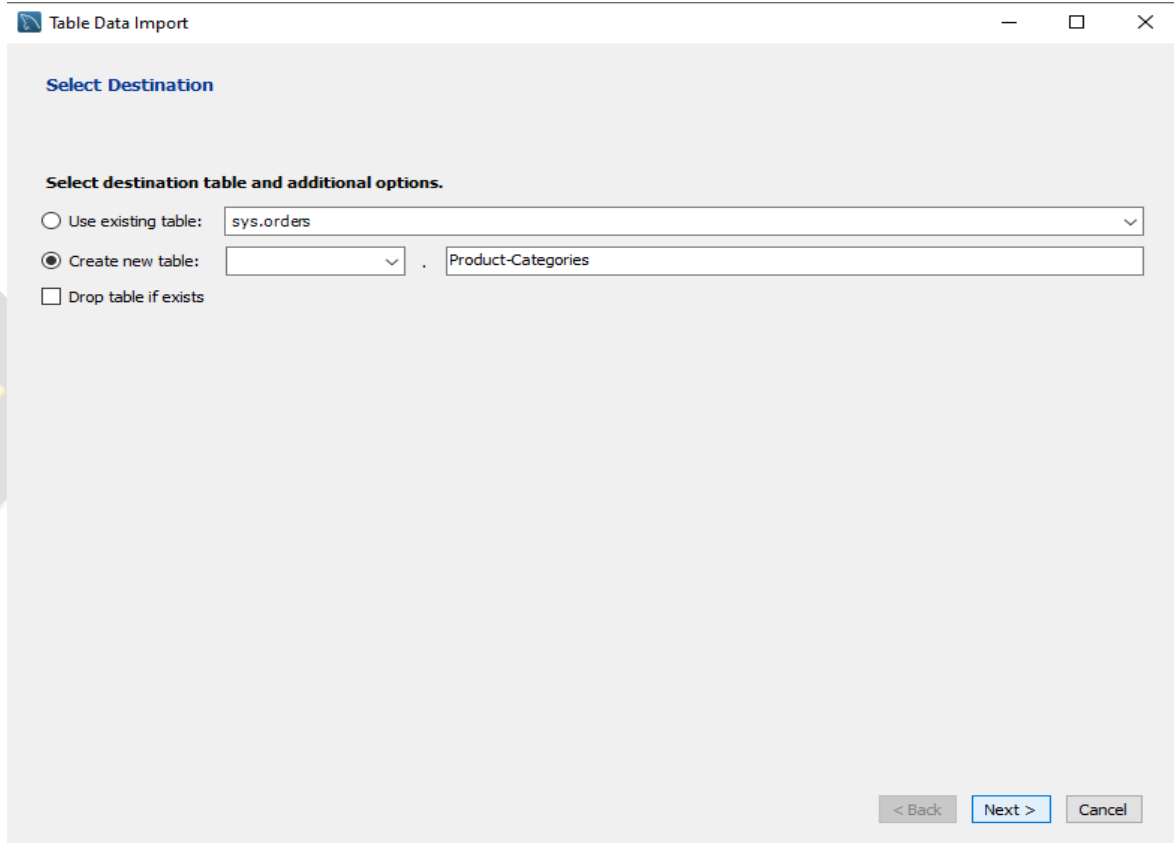


c. Import the CSV files:

- i. Navigate to the schema you created, right-click on 'Tables', and choose 'Table Data Import Wizard'.



- ii. Browse the CSV file, select the file you want to import, and click next.
- iii. In the import options, select 'Create New Table' and use the CSV filename as the table name.



Select Destination

Select destination table and additional options.

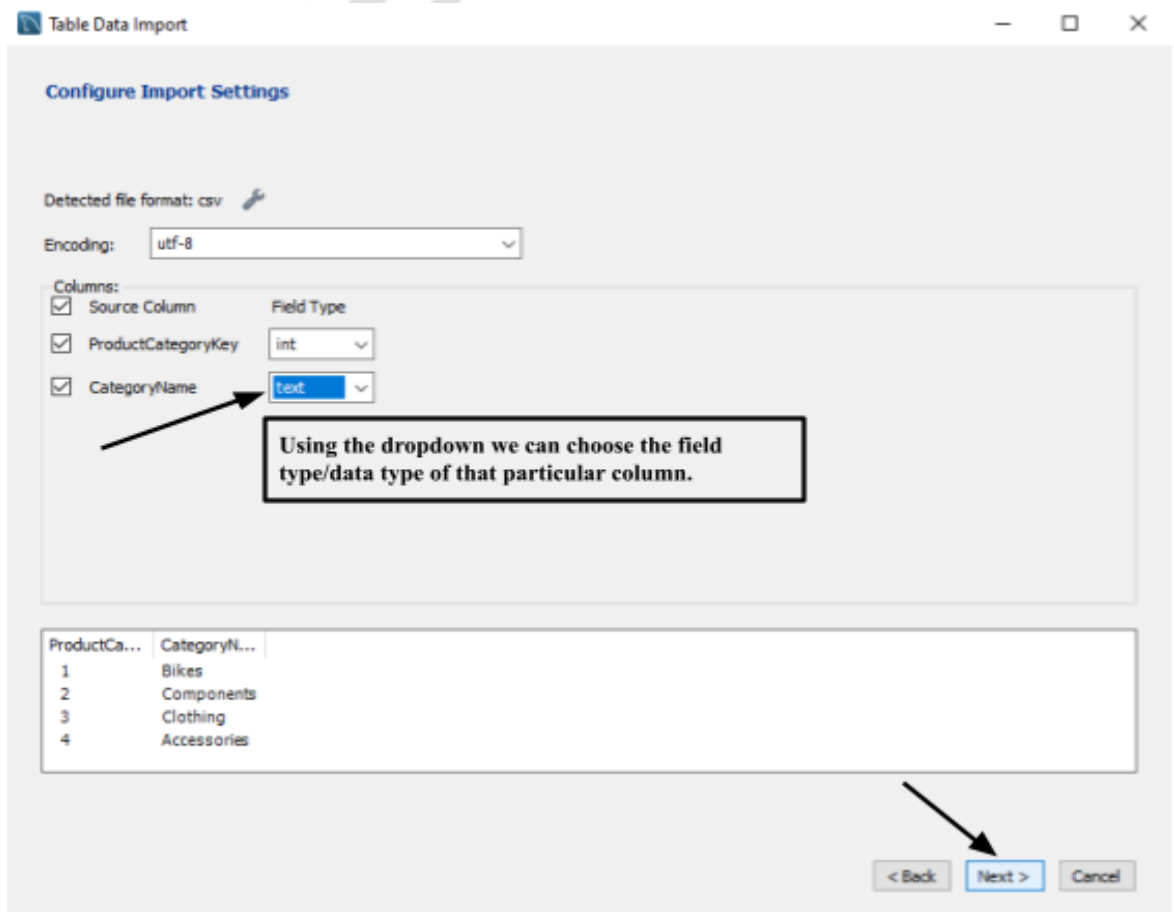
☐ Use existing table: sys.orders

☒ Create new table: , Product-Categories

☐ Drop table if exists

< Back Next > Cancel

iv. Configure the field definitions, adjust data types if necessary, and click next.



Configure Import Settings

Detected file format: csv

Encoding: utf-8

Columns:

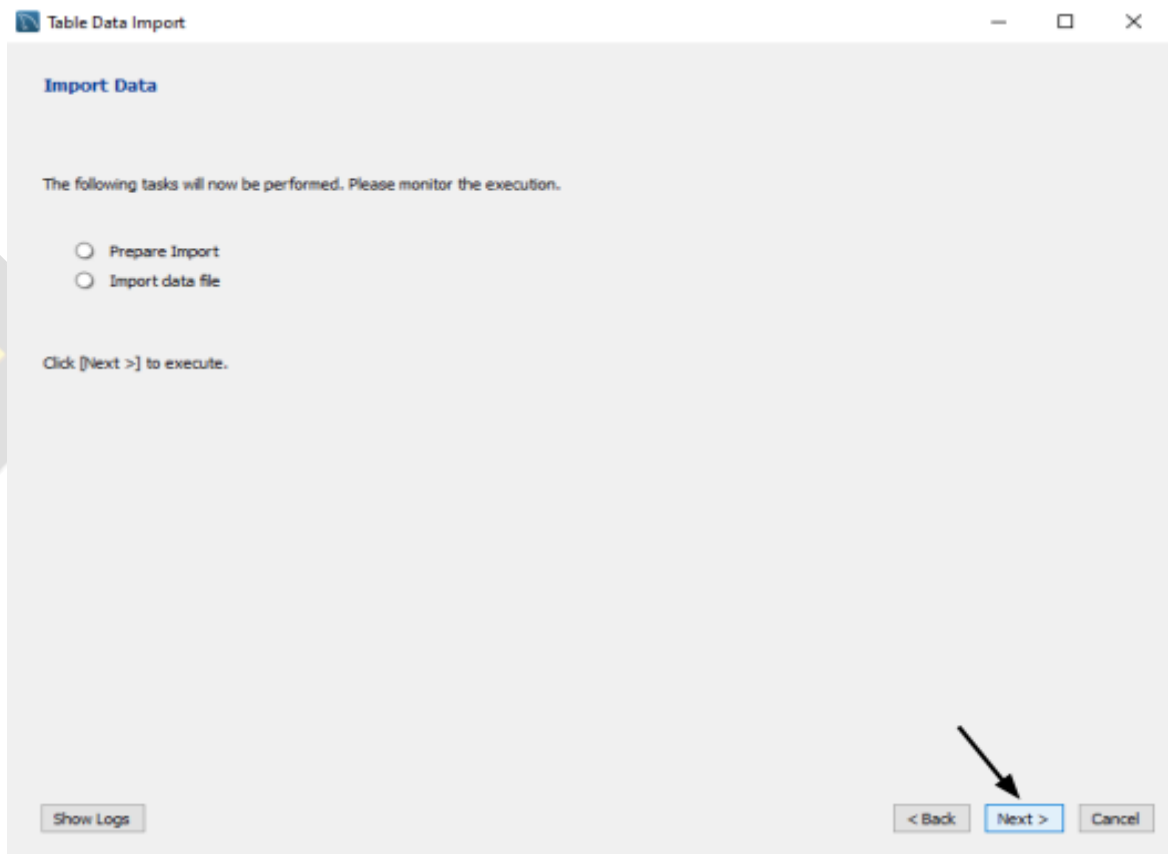
Source Column	Field Type
<input checked="" type="checkbox"/> ProductCategoryKey	int
<input checked="" type="checkbox"/> CategoryName	text

Using the dropdown we can choose the field type/data type of that particular column.

ProductCa...	CategoryN...
1	Bikes
2	Components
3	Clothing
4	Accessories

< Back Next > Cancel

v. Execute the import.



Understanding the datasets:

Here, we will be using the SELECT function to understand the datasets.

Uses:

1. The SELECT statement is fundamental in SQL for data retrieval, which is the process of fetching data from database tables or views. This functionality forms the basis of most interactions with a relational database as it allows users to specify and manipulate the exact data they need.
2. If we need to retrieve all columns from a table, you can use the asterisk (*) symbol instead of column names which will
3. You can specify one or more columns in the SELECT statement. This is useful when you know exactly which pieces of data you need, which can help improve the performance of your query by retrieving only the necessary data.

Syntax 1:

```
SELECT * FROM Table_Name
```

Example 1:

```
SELECT * FROM Customers
```

Output:

The output sample for the above query has been mentioned below.

CustomerKey	Prefix	FirstName	LastName	BirthDate	MyUnknownColumn	MaritalStatus	Gender	EmailAddress	AnnualIncome	TotalChildren	EducationLevel	Occupation	HomeOwner
11000	MR.	JON	YANG	04/08/1966		M	M	jon24@learnsector.com	\$90,000	2	Bachelors	Professional	Y
11001	MR.	EUGENE	HUANG	14/05/1965		S	M	eugene10@learnsector.com	\$60,000	3	Bachelors	Professional	N
11002	MR.	RUBEN	TORRES	08/12/1965		M	M	ruben35@learnsector.com	\$60,000	3	Bachelors	Professional	Y
11003	MS.	CHRISTY	ZHU	15/02/1968		S	F	christy12@learnsector.com	\$70,000	0	Bachelors	Professional	N
11004	MRS.	ELIZABETH	JOHNSON	08/08/1968		S	F	elizabeth5@learnsector.com	\$80,000	5	Bachelors	Professional	Y
11005	MR.	JULIO	RUIZ	08/05/1965		S	M	julio1@learnsector.com	\$70,000	0	Bachelors	Professional	Y
11007	MR.	MARCO	MEHTA	05/09/1964		M	M	marco14@learnsector.com	\$60,000	3	Bachelors	Professional	Y
11008	MRS.	ROBIN	VERHOFF	07/07/1964		S	F	rob4@learnsector.com	\$60,000	4	Bachelors	Professional	Y
11009	MR.	SHANNON	CARLSON	04/01/1964		S	M	shannon38@learnsector.com	\$70,000	0	Bachelors	Professional	N
11010	MS.	JACQUELYN	SUAREZ	02/06/1964		S	F	jacquelyn20@learnsector.com	\$70,000	0	Bachelors	Professional	N
11011	MR.	CURTIS	LU	11/04/1963		M	M	curtis9@learnsector.com	\$60,000	4	Bachelors	Professional	Y

*Note: The above query will return all the rows and columns available in the table as the query has * (asterisk) which represents all the records from the table.*

Syntax 2:

```
SELECT column 1, column 2,...
FROM Table_name
```

Example 2:

```
SELECT ReturnDate, TerritoryKey, ProductKey
FROM returns
```

Output:

ReturnDate	TerritoryKey	ProductKey
1/18/2015	9	312
1/18/2015	10	310
1/21/2015	8	346
1/22/2015	4	311
2/2/2015	6	312
2/15/2015	1	312
2/19/2015	9	311
2/24/2015	8	314
3/8/2015	8	350
3/13/2015	9	350
3/14/2015	4	346
3/15/2015	9	340

returns 7 ×

Note: The above query returns the specific column names mentioned after SELECT and returns all the rows of those columns.

Understand the importance of the DESCRIBE function:

1. Table Structure Insight

- DESCRIBE provides critical information about a table's columns, including the name, data type, whether the column can be NULL, and any default values associated with the columns.

2. Data Type Verification

- Knowing the data type of each column is crucial for data integrity and effective querying. It ensures that the correct type of data is stored in the column and helps prevent errors during data insertion or operations.

3. Constraint Checking

- The output from DESCRIBE includes constraints like primary keys, which are critical for database normalization and integrity.

Syntax:

```
DESCRIBE Table_Name;
```

OR

```
DESC Table_Name;
```

Note: We can use either DESCRIBE or DESC(both are Case Insensitive).

Example:

NOTE: The instructor will explain the layout of all the tables in the class. Apart from that, the student can use the DESCRIBE function syntax mentioned below to explore the layout and the data type of every table available in the schema.

```
DESCRIBE Customers;
```

OR

```
DESC Customers;
```

Output:

In MySQL Workbench, the output will be visible in this format. The output corresponds to the Customers table that we have already imported into our MySQL Workbench. Similarly, we can get the information for other tables.

Result Grid						
	Filter Rows:		Export:		Wrap Cell Content:	
	Field	Type	Null	Key	Default	Extra
▶	CustomerKey	int	YES		NULL	
	Prefix	text	YES		NULL	
	FirstName	text	YES		NULL	
	LastName	text	YES		NULL	
	BirthDate	text	YES		NULL	
	MyUnknownColumn	text	YES		NULL	
	MaritalStatus	text	YES		NULL	
	Gender	text	YES		NULL	
	EmailAddress	text	YES		NULL	
	AnnualIncome	text	YES		NULL	
	TotalChildren	int	YES		NULL	
	EducationLevel	text	YES		NULL	
	Occupation	text	YES		NULL	
	HomeOwner	text	YES		NULL	

Understanding different data types that are used in SQL:

String Data Types:

Data type	Description
1. CHAR(size)	<p>A FIXED length string (can contain letters, numbers, and special characters). The size parameter specifies the column length in characters - can be from 0 to 255.</p> <p>Uses: Best used for storing data that has a consistent length, such as abbreviations, country codes, postal codes, or other standardized data.</p>
2. VARCHAR(size)	<p>A VARIABLE length string (can contain letters, numbers, and special characters). The size parameter specifies the maximum column length in characters - can be from 0 to 65535.</p> <p>Uses: Ideal for storing strings where the length can vary significantly, such as names, addresses, and descriptions.</p>
3. TEXT(size)	<p>Holds a string with a maximum length of 65,535 bytes.</p> <p>Uses: Suitable for storing large texts such as articles, emails, or content of books where the length exceeds the limits of VARCHAR.</p>
4. BLOB(size)	<p>For BLOBs (Binary Large Objects). Holds up to 65,535 bytes of data.</p> <p>Uses: Used for storing binary data such as images, audio files, video files, and other multimedia formats.</p>
5. VARBINARY(size)	<p>Equal to VARCHAR(), but stores binary byte strings. The size parameter specifies the maximum column length in bytes.</p> <p>Uses: Commonly used to store files directly in the database, such as PDF documents, Word documents, or executable files.</p>

Numeric Data Types:

Data Type	Description
1. INT(size)	<p>A medium integer. The signed range is from -2147483648 to 2147483647. The unsigned range is from 0 to 4294967295. The size parameter specifies the maximum display width (which is 255)</p> <p>Uses: Ideal for storing data that represent counts, such as the number of users, posts, or transactions, where fractional numbers are not applicable.</p>
2. BIGINT(Size)	<p>A large integer. The size parameter specifies the maximum display width (which is 255)</p> <p>Uses: Useful in scenarios involving very high counts, such as counting the number of interactions on a high-traffic website or application.</p>
2. Numeric (size, d)	<p>The NUMERIC data type in SQL is a highly precise data type used primarily for storing numbers with fixed precision and scale. The total number of digits is specified in size. The number of decimal digits specified in d.</p> <p>Uses: NUMERIC is favored in any scenario where the exactness of decimal numbers is crucial and where rounding errors cannot be tolerated.</p>
2. FLOAT(p)	<p>A floating point number. MySQL uses the p value to determine whether to use FLOAT or DOUBLE for the resulting data type. If p is from 0 to 24, the data type becomes FLOAT().</p> <p>Uses: Suitable for scientific calculations where the exact precision is less critical than the ability to represent numbers in a large range.</p>
3. DECIMAL(size, d)	<p>An exact fixed-point number. The total number of digits is specified in size. The number of digits after the decimal point is specified in the d parameter. The maximum number for size is 65. The maximum number for d is 30. The default value for size is 10.</p> <p>Uses: Predominantly used in financial and accounting applications where precision in calculations like currency operations is mandatory.</p>
4. DOUBLE (size, d)	<p>A normal-size floating point number. The total number of digits is specified in size. The number of digits after the decimal point is specified in the d parameter.</p>

Date Data types:

Data Types	Description
1. DATE	A date. Format: YYYY-MM-DD. The supported range is from '1000-01-01' to '9999-12-31'
2. DATETIME	A date and time combination. Format: YYYY-MM-DD hh: mm: ss. The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'.
3. TIMESTAMP	TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD hh:mm:ss.

3. YEAR	A year in four-digit format. Values allowed in four-digit format: 1901 to 2155, and 0000.
---------	---

NOTE: The above-mentioned datatypes are used frequently in the industry. Here is a link that will help you understand other data types that exist in SQL. [Link](#)

Conversion of the data type using CAST:

The CAST function in SQL is a powerful and standard tool used for converting a value from one data type to another. This type of conversion is often necessary in database operations to ensure data compatibility, to meet the requirements of SQL functions, or to ensure accurate calculation results.

The primary purpose of the CAST function is to change the data type of an expression or a database column to another type, enabling more flexible control over how data is processed and presented. This is particularly useful in scenarios where data types might affect the outcome of queries or when data needs to be explicitly formatted before being passed to another system or used in reports.

Syntax:

```
CAST(expression AS data_type)
```

NOTE:

- *Here, expression is any valid SQL expression such as a column name, a literal, or a function call.*
- *The data_type specifies the target data type that the expression should be converted to, and it can include additional qualifiers like length or precision, depending on the data type.*

Example:

```
SELECT CAST(CustomerKey AS CHAR(50))  
FROM customers;
```

Note:

- *Here we have converted the CustomerKey column to CHAR(50) from int data type.*
- *The above example is just for demonstration purpose and there wasn't any need of converting the data type of the customerKey Column.*

Understanding WHERE clause in SQL:

The WHERE clause in SQL is a fundamental element used to specify conditions on columns for the rows to be returned in a query.

The primary use of the WHERE clause is to filter rows based on one or more conditions. It allows you to specify conditions on columns that must be met for a row to be included in the result set.

Syntax:

```
SELECT column1, column2, ...  
FROM table_name  
WHERE condition;
```

Example 1:



```
# Query to find the customers with MaritalStatus as 'M'.
SELECT CustomerKey, MaritalStatus, Gender, AnnualIncome
FROM Customers
WHERE MaritalStatus="M";
```


Note:

- The above query returns the specific columns mentioned along with the filter applied using the WHERE clause. The returned table will consist of columns like CustomerKey, MaritalStatus, Gender, and AnnualIncome where the Marital status is "M".
- Whenever we use the WHERE clause, we have to put the filtering criteria inside ". You are likely to encounter an SQL error stating that a column doesn't exist if you mistakenly do not add quotes in your filtering element.
- To avoid confusion and ensure compatibility across different SQL databases, it's a best practice to use double quotes for string literals in MySQL Workbench. For example: *SELECT * FROM table WHERE column ="value";*

Output:

Result Grid



Filter Rows:

Export: 




	CustomerKey	MaritalStatus	Gender	AnnualIncome
▶	11000	M	M	\$90,000
	11002	M	M	\$60,000
	11007	M	M	\$60,000
	11011	M	M	\$60,000
	11012	M	F	\$100,000
	11013	M	M	\$100,000
	11016	M	M	\$30,000
	11022	M	M	\$40,000
	11023	M	M	\$40,000
	11024	M	M	\$60,000
	11025	M	NA	\$10,000
	11027	M	M	\$30,000

Customers 9 ×

Example 2:

```
# Query to find the married female customers and their Annual Income.
SELECT CustomerKey, FirstName, LastName, AnnualIncome
From Customers
WHERE Prefix="MRS.";
```

Output:

Result Grid |   Filter Rows: | Export: 

	CustomerKey	FirstName	LastName	AnnualIncome
▶	11004	ELIZABETH	JOHNSON	\$80,000
	11008	ROBIN	VERHOFF	\$60,000
	11012	LAUREN	WALKER	\$100,000
	11014	SYDNEY	BENNETT	\$100,000
	11017	SHANNON	WANG	\$20,000
	11028	JILL	JIMENEZ	\$30,000
	11030	BETHANY	YUAN	\$10,000
	11031	THERESA	RAMOS	\$20,000
	11032	DENISE	STONE	\$20,000
	11034	EBONY	GONZALEZ	\$20,000
	11038	DIANA	HERNANDEZ	\$10,000
	11046	CHRISTINE	YUAN	\$30,000

Customers 13 × 