

UNIT-4

Full Stack Development

Database Design using MySQL:

1. An Overview of SQL,
2. XAMPP and MySQL Setup,
3. Create Tables,
4. Columns and Insert Data,
5. Selecting Data,
6. Distinct,
7. Aliases & Concat,
8. Update,
9. Delete & Alter,
10. Foreign Keys,
11. Table Joins.

1. A Overview of SQL

SQL which stands for **Structured Query Language** is a language to manage and communicate with databases. For instance, it is used for database creation, deletion, update rows by fetching rows, modifying rows, etc. SQL statements are used for tasks like updating data on a database or retrieving data from a database.

SQL is an **ANSI (American National Standards Institute)** standard language. There are other versions of the SQL language like **T-SQL by Microsoft, PSQL by Interbase/ Firebird, etc.** Therefore, the most common relational database management systems that use SQL are Sybase, Microsoft SQL Server, Oracle, Ingres, Access, etc. Despite that most of these database systems use SQL, most of them have other additional proprietary extensions that are made particularly for their system. Hence, In this blog, we will be focusing on the ANSI/ ISO standard certified SQL language.

What is SQL?

SQL (Structured Query Language) is a relational query language that is used for interacting with the database. It is used for accessing, storing, and manipulating data in a relational database. SQL is an ANSI/ISO certified standard and mostly all the RDMS uses SQL as a standard database query language.

What is it used for?

SQL is a medium to communicate with the data stored in the relational database. It can perform all the CRUD (CREATE, RETRIEVE, UPDATE, DELETE) operations in the database.

Why SQL?

1. Common querying language

As mentioned above, SQL is a standard for database language. It is used by most of the database programs present. (i.e. SQL Server, MYSQL, SQLite, etc.)

2. Easy to learn

Yes, SQL can be considered as an easy language. SQL uses English like language statements, thus most people can easily understand SQL with a basic level of knowledge. Furthermore, SQL is an open-source program so learning resources can be effortlessly found.

3. Efficient and easy methods

SQL is an efficient way of handling data. Complex results can be produced with basic queries. With this simple querying method, data analysis, data manipulation, and data testing can be done easier and faster.

Types of SQL Commands

1. DQL (Data Query Language)

DQL is used for retrieving the saved records from the database.

Command Name	Description
SELECT	Retrieve data from tables in the database

2. DDL (Data Definition Language)

DDL is used for determining the table schemas.

Command Name	Description
CREATE	Make a new table or database
ALTER	Change/Alter the structure of the table
TRUNCATE	Delete records/rows of a table
DROP	Delete a table definition
RENAME	Change name of the table

3. DCL (Data Control Language)

DCL is used for changing the authority of database users.

Command Name	Description
GRANT	Give authorization to a user
REVOKE	Take back authorization of a user

4. TCL (Transaction Control Language)

TCL is used for managing changes made to the data in a table.

Command Name	Description
COMMIT	Make permanent changes
ROLLBACK	Reverse changes
SAVEPOINT	Make temporary changes

5. DML (Data Manipulation Language)

DML is used for manipulating the data in a table of a database.

Command Name	Description
INSERT	Addition of new rows
UPDATE	Update an existing row
DELETE	Delete a row

2. XAMPP and MySQL Setup

XAMPP

XAMPP is an open-sourced server setup to test the projects on a local machine before making it available to everyone over the web. This web app development server comes pre-installed with Apache web server, MySQL database, PHP, and Perl that help you build an offline application with desired features and functions.

XAMPP is a lightweight solution that works perfectly on multiple platforms like Linux, Windows, and Mac OS.

MYSQL

MySQL is an open-sourced relational database management system (RDBMS) that relies on SQL (structured query language). Most of the web-based applications use RDBMS for its development. Basically, MySQL helps you structure your data, present the information in an organized manner, edit, delete, update, or retrieve data whenever required.

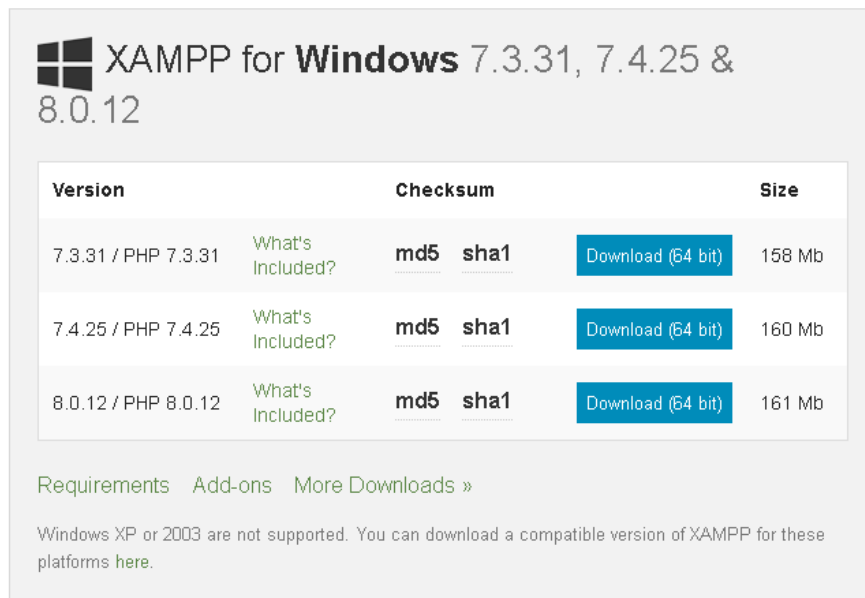
How to Install XAMPP on Windows 10

Let's get started with the step-by-step guide to how to install XAMPP on Windows 10.

Step 1: Download and Install XAMPP

To download and install XAMPP, go to [apachefriends downloads page](#).

You will see XAMPP ready to download for cross-platform like Windows, Linux, Mac OS X. Since we are discussing How to install XAMPP on Windows 10, we will choose the Windows option as shown below.



The screenshot shows the XAMPP for Windows download page. It features a table with three rows of download options for different versions of XAMPP. Each row includes the version number, a link to 'What's Included?', checksums for md5 and sha1, a 'Download (64 bit)' button, and the file size in Mb. Below the table are links for 'Requirements', 'Add-ons', and 'More Downloads'. A note at the bottom states that Windows XP or 2003 are not supported and provides a link to download a compatible version for these platforms.

Version	Checksum	Size
7.3.31 / PHP 7.3.31	md5 sha1	158 Mb
7.4.25 / PHP 7.4.25	md5 sha1	160 Mb
8.0.12 / PHP 8.0.12	md5 sha1	161 Mb

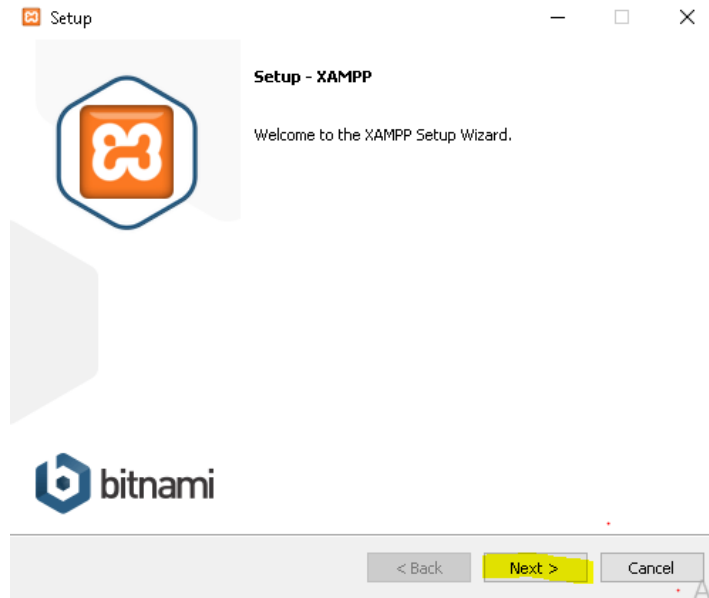
Requirements Add-ons More Downloads »

Windows XP or 2003 are not supported. You can download a compatible version of XAMPP for these platforms [here](#).

Step 2: Run the Installer to Install XAMPP

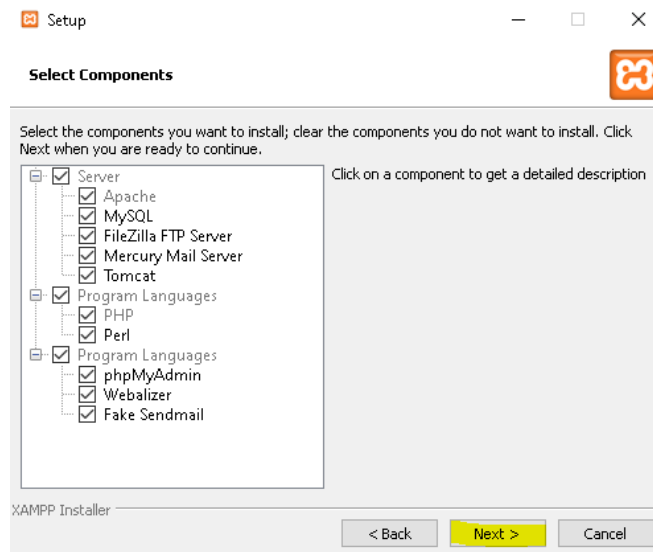
1. XAMPP Setup Wizard

During the installation process, you may come across warning pop-ups. But you would probably click 'Yes' to start the installation process. Soon after you click on the downloaded file, the XAMPP setup wizard will open. Now click on the 'Next' Button to proceed.



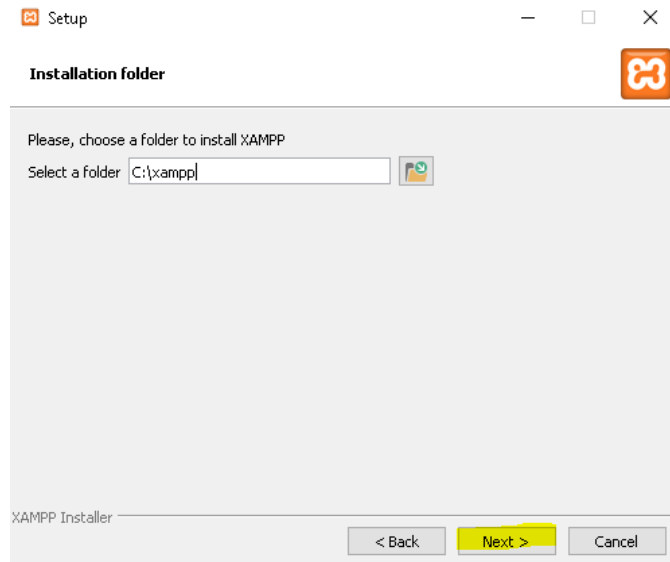
2. Select Components

Next, you need to check the components which you want to install and can uncheck or leave as it is which you don't want to install. You can see there are a few options which are light grey in color. These are the options which are necessary to run the software and will automatically be installed. Now click on the 'Next' button to continue.



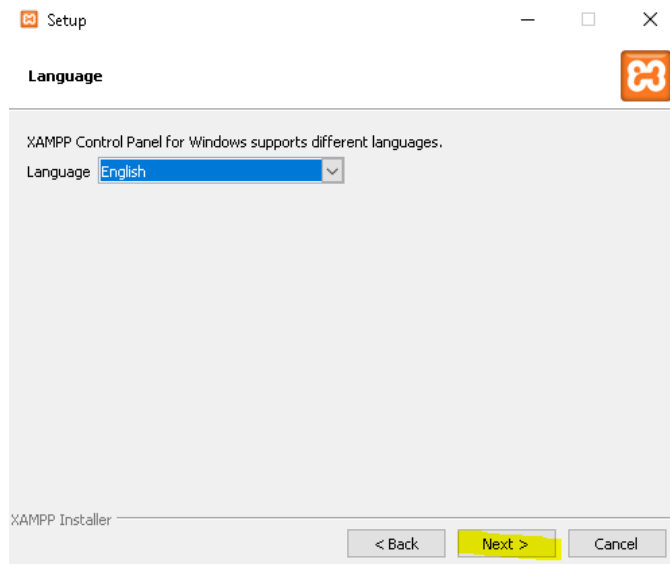
3. Select Installation Folder

Now you need to choose the folder where you want to install the XAMPP. You can choose the default location or you can choose any location of your choice and choose the 'Next' button to move ahead.



4. Select language

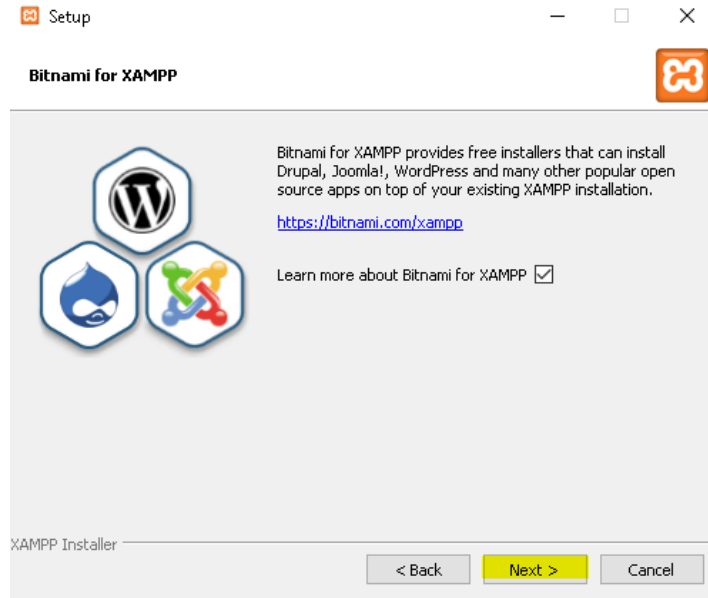
Now will see a window showing you information about language. You can choose which language you want to work with. Then, click on the 'Next' button to continue.



5. Bitnami for XAMPP

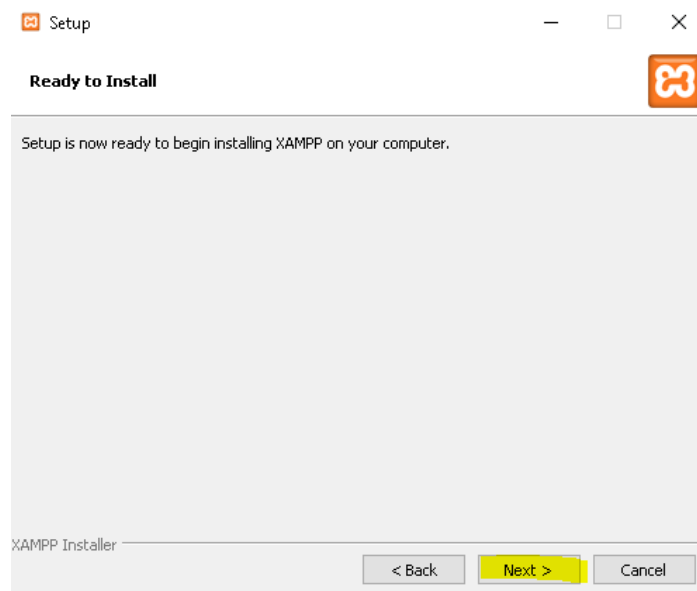
Now will see a window showing you information about Bitnami. Simply click on the 'Next' button to move further. However, if you wish to learn more about the Bitnami, then you may check the box saying 'Learn more about Bitnami for XAMPP.'

Basically Bitnami is for installing open source applications i.e. WordPress, Joomla etc on your newly installed XAMPP.

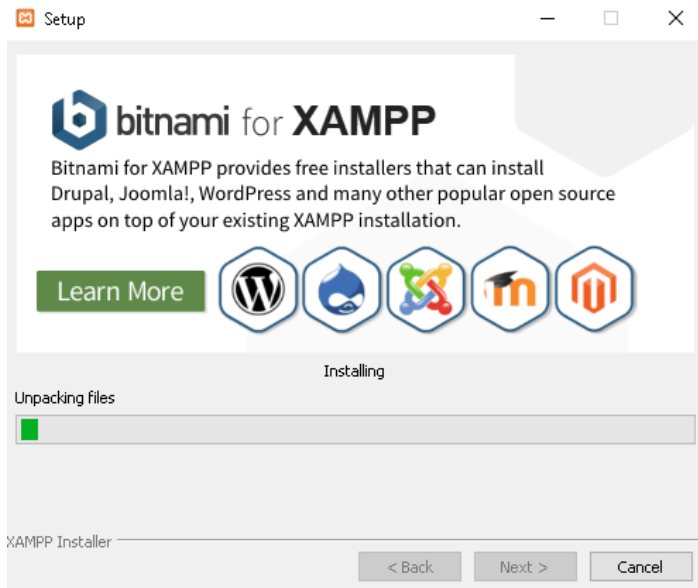


6. Ready to Install XAMPP

Now you'll see another window with a message "Setup is now ready to begin installing XAMPP on your computer" like shown below. You just have to hit the 'Next' button to proceed.

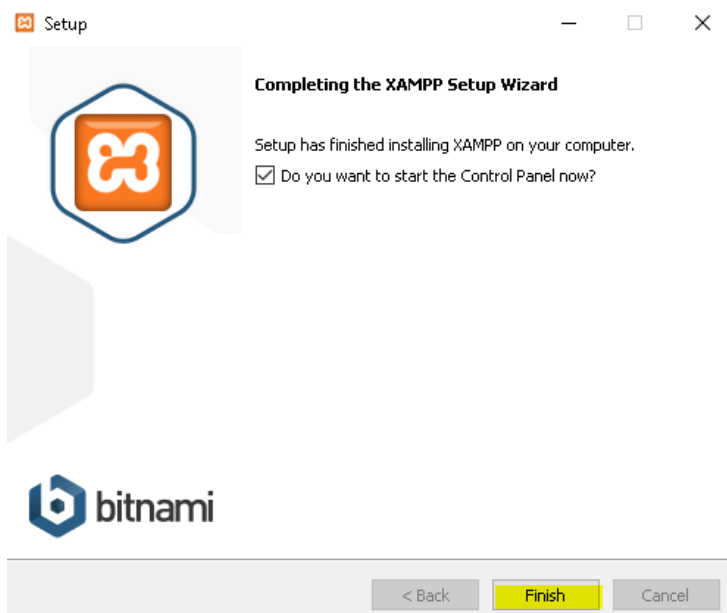


7. Waiting for installing process



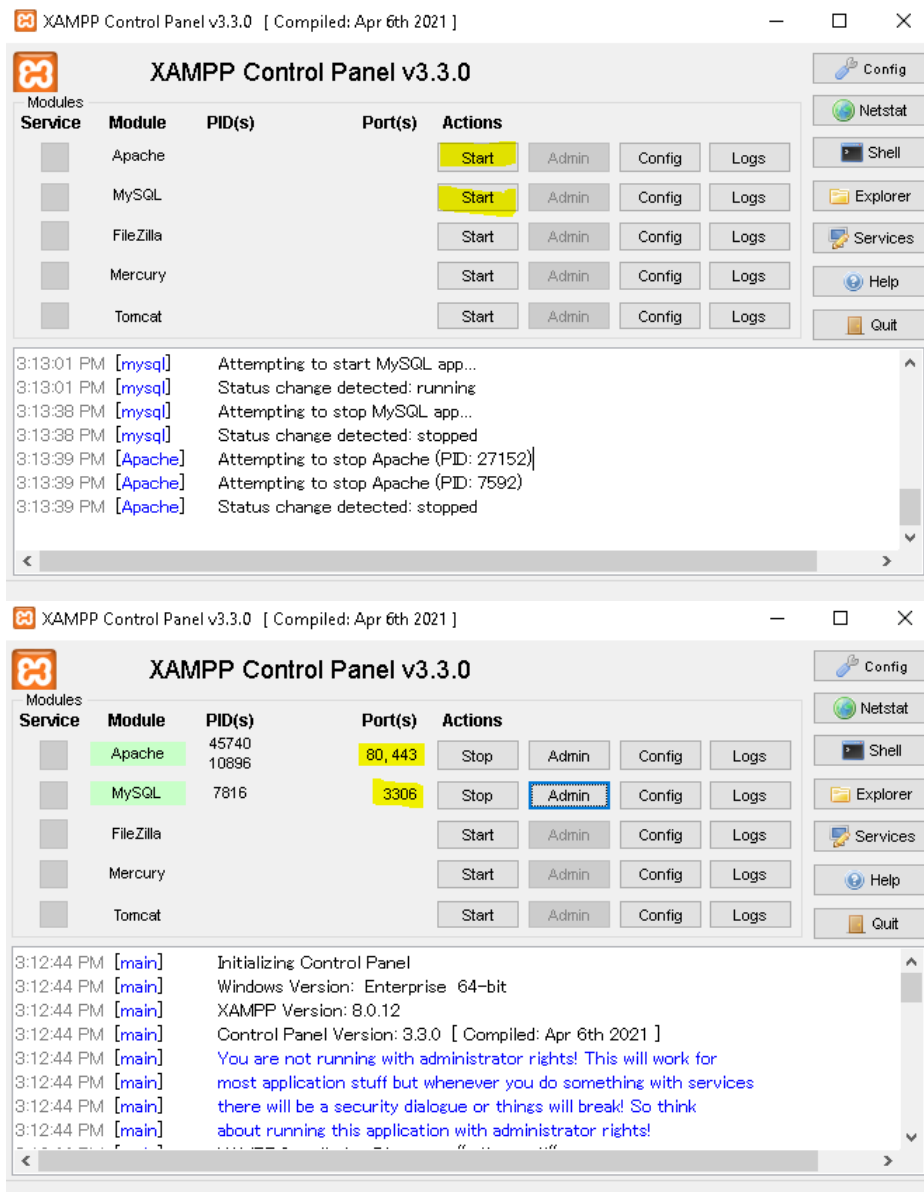
8. XAMPP Installation Complete

Once the installation is completed, you will be asked whether you would like to start the control panel now or not, displaying the message “Do you want to start the control panel now?” Check the box and click on the ‘Finish’ button and see if the XAMPP is working fine.



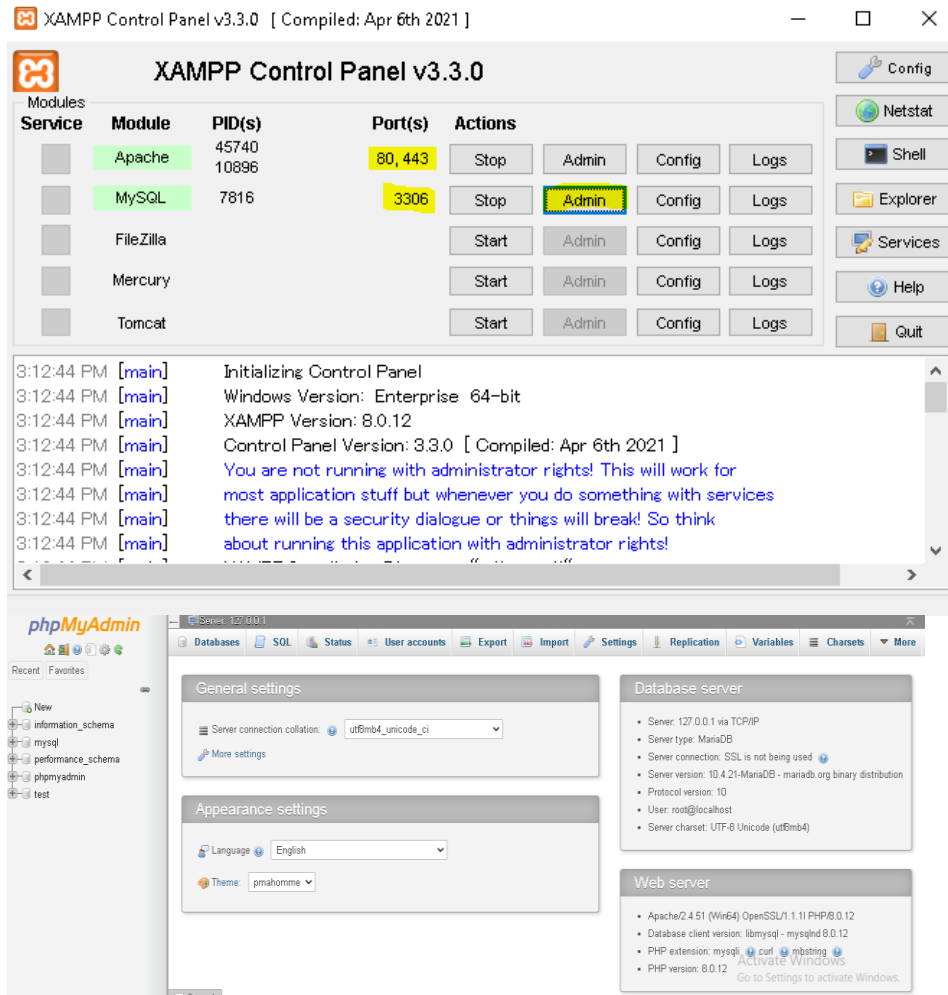
Step 4: XAMPP is now Installed on Windows, run it

If the entire process of XAMPP installation went correctly, then the control panel would open smoothly. Now click on the 'Start' button corresponding to Apache and MySQL.



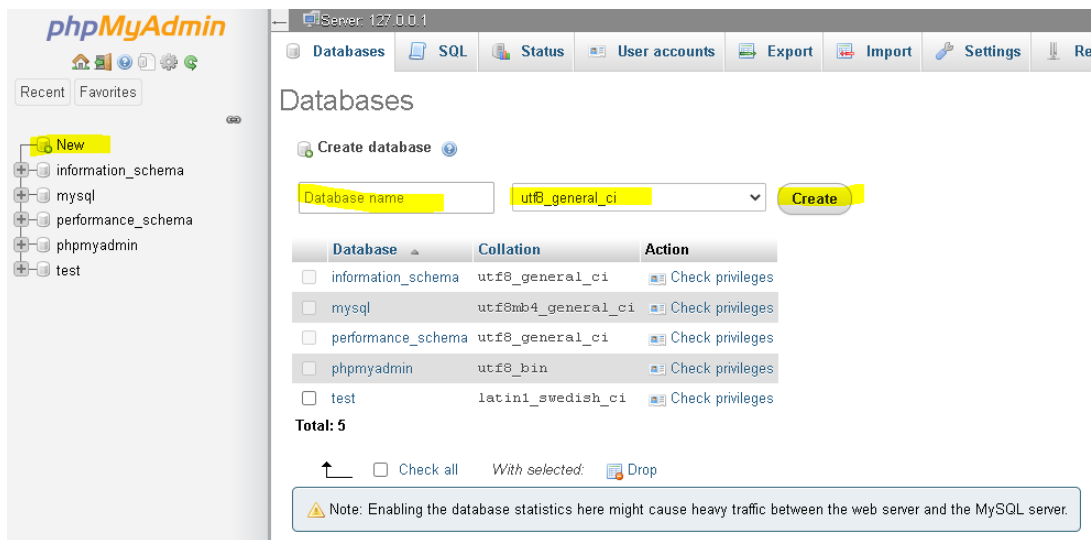
IV. Create a simple MYSQL Database on XAMPP with phpMyAdmin

To working with MYSQL we need a web interface - phpMyAdmin which makes working with database easy and simple. Click 'Admin' on Xampp Control Panel or enter the link <http://localhost/phpmyadmin/> to access it.



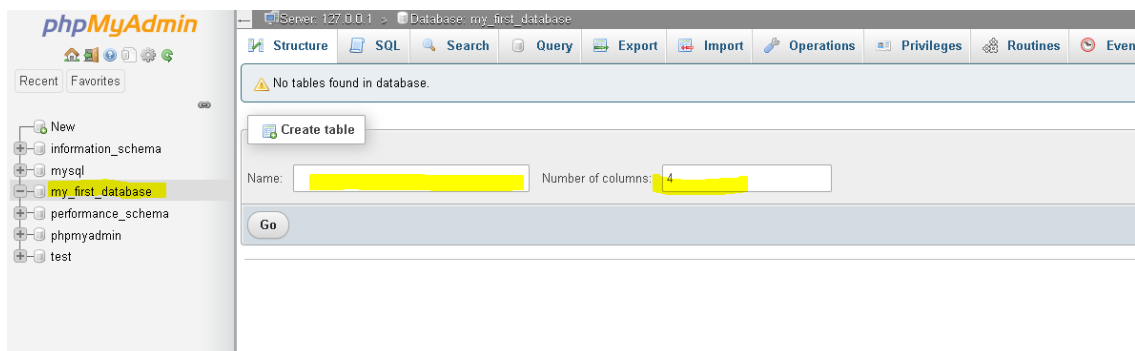
1. Create your first Database

You can create your Database by click 'New' then you will see the form which you need to complete to create your own. Enter name of Database, charset, then click 'Create' to start creating.

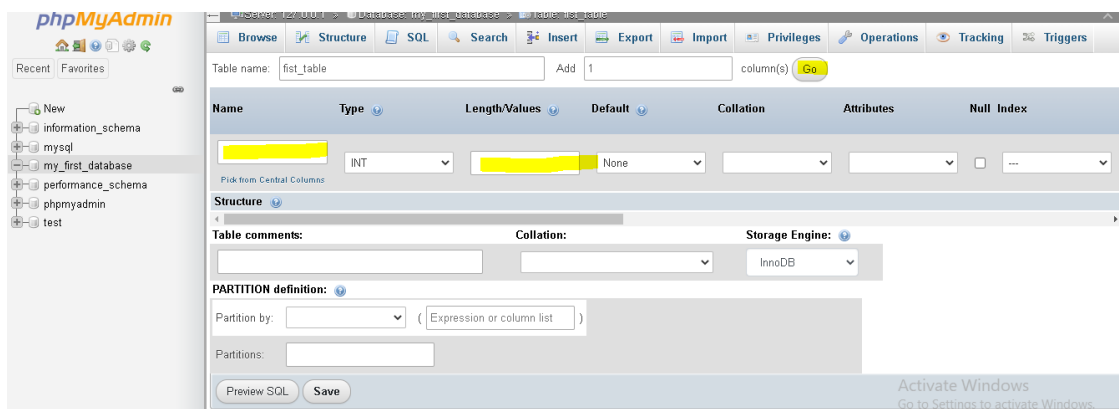


2. Create Table on your Database

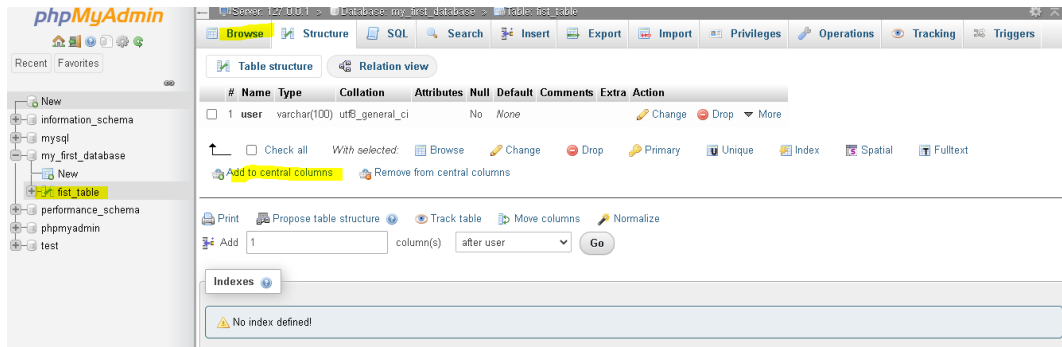
After creating Database successfully you can see your Database in the list on you left side. In the right side, you can create table by enter the name and number of column then click 'Go' to go ahead.



You can set you field name, type, length, .etc... in your table and then click 'Save' to complete creating Table.



Done, you can see all table in the left side list... and simply to update, create more tables.

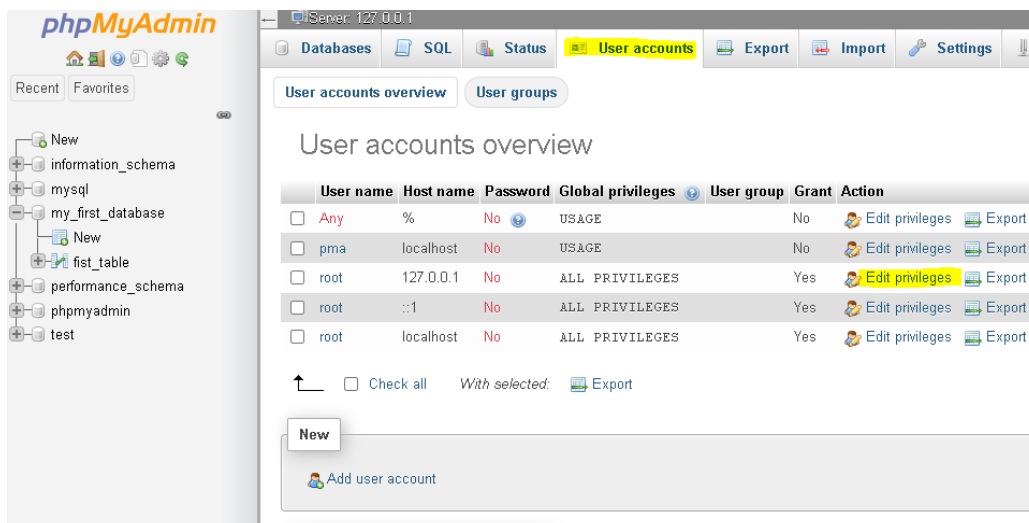


3. Set Password to phpMyAdmin on XAMPP

Step 1: On phpMyAdmin page, click on the 'User accounts' option at the top of the page.

Step 2: Now press the 'Edit Privileges' under 'Actions' option for the Username 'root' and Hostname 'localhost.'

Step 3: Now choose the third tab 'Change password' and type your password in the provided field, retype the password to confirm it and then finally click on the 'Go' key to conclude the process.

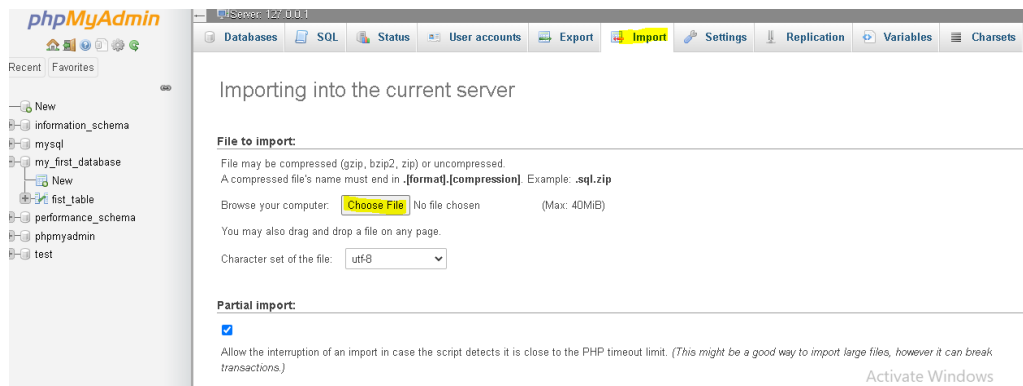


4. Import Database on XAMPP phpMyAdmin

In order to import MySQL database via phpMyAdmin XAMPP, follow the following step.

- Open the Database in phpMyAdmin.
- Click on the Databases from the top menu.
- Select the name of the database from the drop-down menu which you want to import.
- Click on the Import tab.

- Browse your .sql file by clicking on the ‘Choose File’ option that you wish to import. And then click on the ‘Go’ button at the bottom.
- You’re done!!



5. Export Database on XAMPP phpMyAdmin

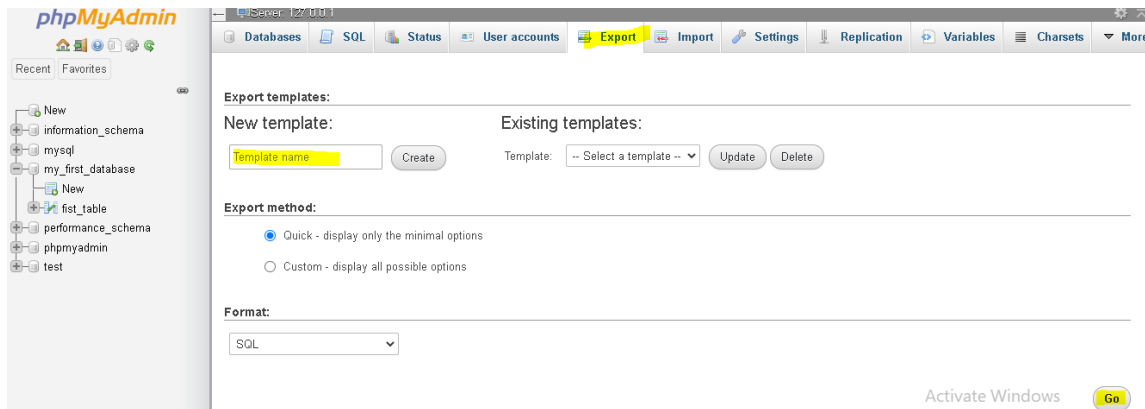
In order to export MySQL database via phpMyAdmin XAMPP, follow the following step.

- To begin the export process, log in to the cPanel and open the phpMyAdmin interface.
- In the left pane of the phpMyAdmin, select the database you want to export.
- Select the ‘Export’ tab at the top.
- Now, you shall see two options i.e. Quick and Custom. You can select Quick if you want to go with default option or select Custom if you want to select particular tables and compression type of the exported file in addition to many other options.

Note: The older version of phpMyAdmin does not support Quick. So you’ll have to follow another method for that.

- Now, single out the format you wish to export your database in. Click on the drop-down menu and opt out the one from the given list.
- Confirm your choice by clicking on the ‘Go’ button.
- Now, you will be asked to either open or save the chosen file. You may select save the file and save it to your desired location.

Congratulations!! You have successfully exported the file.

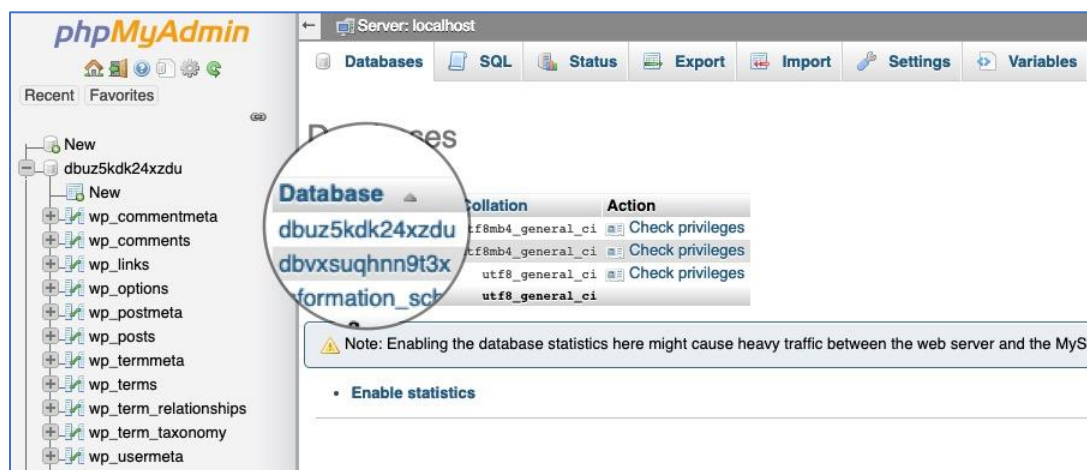


As you can see all the things in phpMyAdmin XAMPP are greatly powerful that gives you the freedom to manage the database efficiently.

Now, you can easily install your own Machine to work with MYSQL on XAMPP. Certainly, the biggest advantage of installing XAMPP is that you don't have to worry about the number of attempts you make or the kind of experiments you perform on your site. You can freely try anything you want without any tension of losing your users.

3.Creating tables in XAMPP:

To create new tables inside a database, open the phpMyAdmin tool, click on the **Databases** tab and click on the name of the desired database.



On the new page that opens you will see a list of all the current tables inside the database and a section named **Create table**. In that section, in the **Name** field, input the desired new name of the table and then select the number of columns that the table should have via the **Number of columns** drop-down. When ready, click on **Go** to create the table.



On the next page, you can configure the structure of the columns in the new table. The different fields there are:

- **Name** – The name of the column;
- **Type** – The type of data, which will be stored in the corresponding column. More details about the possible choices can be found in the official [MySQL Data Types](#) documentation;
- **Length/Values** – The length of the field;
- **Default** – With this option, you can specify if the fields in the column would have a default value. This is useful when you want to have timestamps for the entries in each row;
- **Collation** – The data collation for each of the fields;
- **Attributes** – assign any special attributes to the fields;
- **Null** – Define whether the field value can be *NULL*. More about the *NULL* value can be found in the [MySQL documentation](#);
- **Index** – Set the Index of the row. More information about the MySQL column indexes can be found in the [MySQL documentation](#);
- **A_I** – Short for Auto Increment. If this option is enabled then the values in the fields of the column will be auto incremented;
- **Comments** – Here add comments, which will be included in the database SQL code.

After you configured the different columns, specify the **Collation** and **Engine** of the new table via their respective drop-downs.

When ready, click on **Save** to create the new table.

The screenshot shows the 'Structure' tab in phpMyAdmin, which is used for defining the structure of a new database table. The interface includes a table with the following columns: 'Attributes', 'Null', 'Index', 'A.I.' (Auto-Increment), 'Comments', 'Virtuality', and 'Move column'. There are four rows in this table, each representing a column to be created. Each row contains a text input field for the column name, a checkbox for 'Null', a dropdown menu for 'Index' (currently showing '---'), a checkbox for 'A.I.', a text input field for 'Comments', a dropdown menu for 'Virtuality' (currently showing '---'), and a dropdown menu for 'Move column' (currently showing '---'). At the bottom right of the interface, there are two buttons: 'Preview SQL' and 'Save'. The 'Save' button is highlighted with a red circle.

Attributes	Null	Index	A.I.	Comments	Virtuality	Move column
<input type="text"/>	<input type="checkbox"/>	---	<input type="checkbox"/>	<input type="text"/>	---	---
<input type="text"/>	<input type="checkbox"/>	---	<input type="checkbox"/>	<input type="text"/>	---	---
<input type="text"/>	<input type="checkbox"/>	---	<input type="checkbox"/>	<input type="text"/>	---	---
<input type="text"/>	<input type="checkbox"/>	---	<input type="checkbox"/>	<input type="text"/>	---	---

Preview SQL Save

How to Add Content in a Database Table

To add records inside a database table, open the table with phpMyAdmin and click on the **Insert** tab.

Server: localhost » Database: dbuz5kdk24xzdu » Table: wp_commentmeta

[Browse](#)
[Structure](#)
[SQL](#)
[Search](#)
[Insert](#)
[Export](#)
[Import](#)
[Operations](#)
[Triggers](#)

Column	Type	Function	Null	Value
meta_id	bigint(20) unsigned	<input type="text"/>	<input type="checkbox"/>	<input type="text"/>
comment_id	bigint(20) unsigned	<input type="text"/>	<input type="checkbox"/>	0
meta_key	varchar(255)	<input type="text"/>	<input checked="" type="checkbox"/>	<input type="text"/>
meta_value	longtext	<input type="text"/>	<input checked="" type="checkbox"/>	<div><div></div></div>

[Go](#)

Enter the desired data in the corresponding fields and click on **Go** to store it. You can see the newly inserted record by clicking on the **Browse** tab.

SQL SELECT Statement

The SELECT statement is the most commonly used command in Structured Query Language. It is used to access the records from one or more database tables and views. It also retrieves the selected data that follow the conditions we want.

By using this command, we can also access the particular record from the particular column of the table. The table which stores the record returned by the SELECT statement is called a result-set table.

Syntax of SELECT Statement in SQL

1. **SELECT** Column_Name_1, Column_Name_2,, Column_Name_N **FROM** Table_Name;

In this SELECT syntax, **Column_Name_1, Column_Name_2,, Column_Name_N** are the name of those columns in the table whose data we want to read.

If you want to access all rows from all fields of the table, use the following SQL SELECT syntax with * asterisk sign: **SELECT * FROM** table_name;

Examples of SELECT Statement in SQL

Here, we took the following two different SQL examples which will help you to execute the SELECT statement for retrieving the records:

Example 1:

Firstly, we have to create the new table and then insert some dummy records into it.

Use the following query to create the **Student_Records** table in SQL:

1. **CREATE TABLE** Student_Records
2. (
3. Student_Id **Int PRIMARY KEY**,
4. First_Name **VARCHAR** (20),
5. Address **VARCHAR** (20),
6. Age **Int** NOT NULL,
7. Percentage **Int** NOT NULL,
8. Grade **VARCHAR** (10)
9.);

The following query inserts the record of intelligent students into the **Student_Records** table:

1. **INSERT INTO** Student **VALUES** (201, Akash, Delhi, 18, 89, A2),
2. (202, Bhavesh, Kanpur, 19, 93, A1),
3. (203, Yash, Delhi, 20, 89, A2),
4. (204, Bhavna, Delhi, 19, 78, B1),
5. (05, Yatin, Lucknow, 20, 75, B1),
6. (206, Ishika, Ghaziabad, 19, 51, C1),

7. (207, Vivek, Goa, 20, 62, B2);

The following SQL query displays all the values of each column from the above Student_records table:

1. **SELECT** * **FROM** Student_Records;

The output of the above query is:

Student_ID	First_Name	Address	Age	Percentage	Grade
201	Akash	Delhi	18	89	A2
202	Bhavesh	Kanpur	19	93	A1
203	Yash	Delhi	20	89	A2
204	Bhavna	Delhi	19	78	B1
205	Yatin	Lucknow	20	75	B1
206	Ishika	Ghaziabad	19	91	C1
207	Vivek	Goa	20	80	B2

Example 2:

The following query displays the values of particular column from the above **Student_Record** table:

1. **SELECT** Student_Id, Age, Percentage, Grade **FROM** Employee;

Student_ID	Age	Percentage	Grade
201	18	89	A2
202	19	93	A1
203	20	89	A2
204	19	78	B1
205	20	75	B1
206	19	91	C1
207	20	80	B2

SELECT Statement with WHERE clause

The WHERE clause is used with SELECT statement to return only those rows from the table, which satisfy the specified condition in the query.

In SQL, the WHERE clause is not only used with SELECT, but it is also used with other SQL statements such as UPDATE, ALTER, and DELETE statements.

Syntax of SELECT Statement with WHERE clause

1. **SELECT** * **FROM** Name_of_Table **WHERE** [condition];

In the syntax, we specify the condition in the WHERE clause using SQL logical or comparison operators.

Example of SELECT Statement with WHERE clause

Firstly, we have to create the new table and then insert some dummy records into it.

Use the following query to create the **Employee_Details** table in SQL:

1. **CREATE TABLE** Employee_Details
2. (

3. Employee_ID **INT** AUTO_INCREMENT **PRIMARY KEY**,
4. Emp_Name **VARCHAR** (50),
5. Emp_City **VARCHAR** (20),
6. Emp_Salary **INT** NOT NULL,
7. Emp_Panelty **INT** NOT NULL
8.);

The following INSERT query inserts the record of employees into the Employee_Details table:

1. **INSERT INTO** Employee_Details (Employee_ID, Emp_Name, Emp_City, Emp_Salary, Emp_Panelty) **VALUES** (101, Anuj, Ghaziabad, 25000, 500),
2. (102, Tushar, Lucknow, 29000, 1000),
3. (103, Vivek, Kolkata, 35000, 500),
4. (104, Shivam, Goa, 22000, 500);

The following SELECT query shows the data of the **Employee_Details** table:

1. **SELECT * FROM** Employee_Details;

Employee_Id	Emp_Name	Emp_City	Emp_Salary	Emp_Panelty
101	Anuj	Ghaziabad	25000	500
102	Tushar	Lucknow	29000	1000
103	Vivek	Kolkata	35000	500
104	Shivam	Goa	22000	500

The following query shows the record of those employees from the above table whose Emp_Panelty is 500:

1. **SELECT * FROM** Employee_Details **WHERE** Emp_Panelty = 500;

This SELECT query displays the following table in result:

Employee_Id	Emp_Name	Emp_City	Emp_Salary	Emp_Panelty
101	Anuj	Ghaziabad	25000	500
103	Vivek	Kolkata	35000	500
104	Shivam	Goa	22000	500

SQL SELECT Statement with GROUP BY clause

The GROUP BY clause is used with the SELECT statement to show the common data of the column from the table:

Syntax of SELECT Statement with GROUP BY clause

1. **SELECT** column_Name_1, column_Name_2,, column_Name_N aggregate_function _name(column_Name2) **FROM** table_name **GROUP BY** column_Name1;

Example of SELECT Statement with GROUP BY clause

Use the following query to create the **Cars_Details** table:

1. **CREATE TABLE** Cars_Details
2. (
3. Car_Number **INT PRIMARY KEY**,
4. Car_Name **VARCHAR** (50),
5. Car_Price **INT** NOT NULL,
6. Car_Amount **INT** NOT NULL
7.);

The following INSERT query inserts the record of cars into the **Cars_Details** table:

1. **INSERT INTO** Cars_Details (Car_Number, Car_Name, Car_Amount, Car_Price)
2. **VALUES** (2578, Creta, 3, 1500000),
3. (9258, Audi, 2, 3000000),
4. (8233, Venue, 6, 900000),
5. (6214, Nexon, 7, 1000000);

The following SELECT query displays the values in the output:

1. **SELECT** * **FROM** Cars_Details;

Car_Number	Car_Name	Car_Amount	Car_Price
2578	Creta	3	1000000
9258	Audi	2	900000
8233	Venue	6	900000
6214	Nexon	7	1000000

The following SELECT with GROUP BY query lists the number of cars of the same price:

1. **SELECT COUNT** (Car_Name), Car_Price **FROM** Cars_Details **GROUP BY** Car_Price
;

The output of above GROUP BY query is shown below:

Output:

Count (Car_Name)	Car_Price
2	1000000
2	900000

SQL SELECT Statement with HAVING clause

The HAVING clause in the SELECT statement creates a selection in those groups which are defined by the GROUP BY clause.

Syntax of SELECT Statement with HAVING clause

1. **SELECT** column_Name_1, column_Name_2,, column_Name_N aggregate_function_name(column_Name_2) **FROM** table_name **GROUP BY** column_Name1 **HAVING** ;

Example of SELECT Statement with HAVING clause

Let's create the **Employee_Having** table in SQL using the below CREATE command:

1. **CREATE TABLE** Employee_Having
2. (
3. Employee_Id **INT PRIMARY KEY**,
4. Employee_Name **VARCHAR** (50),
5. Employee_Salary **INT** NOT NULL,
6. Employee_City **VARCHAR** (50)
7.);

The following INSERT query inserts the record of employees into the Employee_Having table:

1. **INSERT INTO** Employee_Having (Employee_Id, Employee_Name, Employee_Salary, Employee_City)
2. **VALUES** (201, Jone, 20000, Goa),
3. (202, Basant, 40000, Delhi),
4. (203, Rashet, 80000, Jaipur),
5. (204, Aunj, 20000, Goa),
6. (205, Sumit, 50000, Delhi);

The following SELECT query shows the values of Employee_Having table in the output:

1. **SELECT * FROM** Employee_Having;

Employee_Id	Employee_Name	Employee_Salary	Employee_City
201	Jone	20000	Goa
202	Basant	40000	Delhi
203	Rashet	80000	Jaipur
204	Anuj	20000	Goa
205	Sumit	50000	Delhi

The following query shows the total salary of those employees having more than 5000 from the above Employee_Having table:

1. **SELECT SUM** (Employee_Salary), Employee_City **FROM** Employee_Having **GROUP BY** Employee_City **HAVING SUM**(Employee_Salary)>5000;

This HAVING query with SELECT statement shows the following table:

Output:

SUM (Employee_Salary)	Employee_City
90000	Delhi
80000	Jaipur

SELECT Statement with ORDER BY clause

The ORDER BY clause with the SQL SELECT statement shows the records or rows in a sorted manner.

The ORDER BY clause arranges the values in both ascending and descending order. Few database systems arrange the values of column in ascending order by default.

Syntax of SELECT Statement with ORDER BY clause

1. **SELECT** Column_Name_1, Column_Name_2,, column_Name_N **FROM** table_name **WHERE** [Condition] **ORDER BY**[column_Name_1, column_Name_2,, column_Name_N **asc** | **desc**];

Example of SELECT Statement with ORDER BY clause in SQL

1. **CREATE TABLE** Employee_Order
2. (
3. Id **INT** NOT NULL,
4. FirstName **VARCHAR** (50),
5. Salary **INT**,
6. City **VARCHAR** (50)
7.);

The following INSERT query inserts the record of employees into the Employee_Having table:

1. **INSERT INTO** Employee_Order (Id, FirstName, Salary, City)
2. **VALUES** (201, Jone, 20000, Goa),
3. (202, Basant, 15000, Delhi),
4. (203, Rashet, 80000,Jaipur),
5. (204, Aunj, 90000, Goa),
6. (205, Sumit, 50000, Delhi);

The following SELECT query shows the values of the table in the output:

1. **SELECT** * **FROM** Employee_Order;

Id	FirstName	Salary	City
201	Jone	20000	Goa
202	Basant	15000	Delhi

203	Rashet	80000	Jaipur
204	Anuj	90000	Goa
205	Sumit	50000	Delhi

The following query sorts the salary of employees in descending order from the above Employee_Order table:

1. **SELECT * FROM** Employee_Order **ORDER BY** Emp_Salary **DESC**;

This SQL query shows the following table in result:

Output:

Emp_Id	Emp_Name	Emp_Salary	Emp_City
204	Anuj	90000	Goa
203	Rashet	80000	Jaipur
205	Sumit	50000	Delhi
201	Jone	20000	Goa
202	Basant	15000	Delhi

Distinct in SQL:[5 Marks]

SQL **DISTINCT** clause is used to remove the duplicates columns from the result set.

The distinct keyword is used with select keyword in conjunction. It is helpful when we avoid duplicate values present in the specific **columns/tables**. The **unique values** are fetched when we use the distinct keyword.

- SELECT DISTINCT returns only distinct (**different**) values.

- DISTINCT eliminates duplicate records from the table.
- DISTINCT can be used with aggregates: **COUNT**, **AVG**, **MAX**, etc.
- DISTINCT operates on a single column.
- Multiple columns are not supported for DISTINCT.

Syntax:

1. **SELECT DISTINCT** expressions
2. **FROM** tables
3. [**WHERE** conditions];

Parameters:

Expressions: The columns or calculations that we want to retrieve are called expression.

Tables: The tables that we want to retrieve the records. There is only one table in the FROM clause.

WHERE conditions: The conditions may meet for the records which are selected and it is optional.

Note:

- When one expression is provided in the **DISTINCT** clause then the query will return the unique values of the expressions.
- The query will retrieve the unique combinations for the listed expressions if more than one expression is provided in the **DISTINCT** clause here.
- In SQL, the **DISTINCT** clause cannot ignore the NULL values. So when we use the DISTINCT clause in the SQL statement, our result set will include NULL as a distinct value.

Example:

Consider the following **EMPLOYEES** table.

ID	NAME	AGE	ADDRESS	SALARY
001	John	23	America	22000
002	David	21	Australia	24000
003	Newton	34	Paris	20000
004	William	31	Newyork	15000
005	Warner	30	Bangladesh	18000
006	Denial	20	Afghanistan	25000
007	Johnson	22	Brazil	20000

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

1. SQL> **SELECT SALARY FROM EMPLOYEES**
2. **ORDER BY SALARY;**

When we execute the above SQL query, it fetches all the records including the duplicate records. In the above table, salary of Newton and Johnson is same 20000.

SALARY
22000
24000
20000
15000
18000
25000
20000

Now, let us use the **DISTINCT** keyword with the above SELECT query.

1. SQL> **SELECT DISTINCT SALARY FROM EMPLOYEES**
2. **ORDER BY SALARY;**

The above SQL query removes the duplicate records and shows the following result.

SALARY
22000
24000
20000
15000
18000
25000

Example: Finding Unique Values in the Column

Look at the [DISTINCT clause](#) to find the unique values within one column in the table.

We have a table called *suppliers* with the following data:

Supplier ID	Supplier Name	City	State
100	Microsoft Corporation	Redmond	Washington
200	HCL	Mountain View	California
300	Wipro	Redwood City	California
400	Infosys	Irving	Texas
500	Zomato Foods	Springdale	Arkansas
600	Johnson& Johnson	Racine	Wisconsin
700	Adani	Westlake Village	California
800	Flowers Company	Thomasville	Georgia
900	Arts& craft	Redwood City	California

From the above table, we are going to find the unique states.

1. **SELECT DISTINCT** state
2. **FROM** suppliers
3. **ORDER BY** state;

These are **six the** records.

state
Kajipuram
California
Georgia
Texas
Washington
Barcelona

The example returns the unique *state* from *suppliers table* and *removes* the duplicate records from the result set.

Example: Finding Unique Values in Multiple Column

The **SQL DISTINCT** clause is used to remove the duplicate records from many fields in the **SELECT** statement.

Enter the **SQL** statement:

1. **SELECT DISTINCT** city, state
2. **FROM** suppliers
3. **ORDER BY** city, state;

Output:

These are 8 records:

city	state
Irving	Texas
Mountain View	California
Racine	Barcelona
Redmond	Washington
Redwood City	California
Springdale	Kajipuram
Thomasville	Georgia
Westlake Village	California

The example returns each unique *city and state* combination. We see the **Redwood City** and **California**, appears in the result set.

Example: DISTINCT Clause handles NULL Values

The DISTINCT clause considers **NULL** to the unique value in **SQL**. We have a table called *products* which contains the below data.

fruit_id	fruit_name	category_id
1	Grapes	55
2	Papaya	55
3	Apple	55
4	Orange	55
5	Banana	70
6	Mango	35
7	Pear	NULL

Select the unique values from the field fruit_id which contains the null value. Enter the below [SQL](#) syntax:

1. **SELECT DISTINCT** fruit_id
2. **FROM** fruits
3. **ORDER BY** category_id;

There are four records selected. These are the results which we see below:

category_id
NULL
35
55
70

In the above example, the query returns the unique values that are in the *category_id* column. We see by the first row in the result set, **NULL** is an exceptional value which is returned by the **DISTINCT** clause.

SQL JOIN[10 marks]

SQL Join is used to fetch data from two or more tables, which is joined to appear as single set of data. It is used for combining column from two or more tables by using values common to both tables.

JOIN Keyword is used in SQL queries for joining two or more tables. Minimum required condition for joining table, is **(n-1)** where **n**, is number of tables. A table can also join to itself, which is known as, **Self Join**.

Types of JOIN

Following are the types of JOIN that we can use in SQL:

- Inner
- Outer
- Left
- Right

Cross JOIN or Cartesian Product

This type of JOIN returns the cartesian product of rows from the tables in Join. It will return a table which consists of records which combines each row from the first table with each row of the second table.

Cross JOIN Syntax is,

```
SELECT column-name-list
```

FROM

table-name1 CROSS JOIN table-name2;

Copy

Example of Cross JOIN

Following is the **class** table,

ID	NAME
1	abhi
2	adam
4	alex

and the **class_info** table,

ID	Address
1	DELHI
2	MUMBAI
3	CHENNAI

Cross JOIN query will be,

SELECT * FROM

class CROSS JOIN class_info;

Copy

The resultset table will look like,

ID	NAME	ID	Address
1	abhi	1	DELHI
2	adam	1	DELHI
4	alex	1	DELHI
1	abhi	2	MUMBAI
2	adam	2	MUMBAI
4	alex	2	MUMBAI
1	abhi	3	CHENNAI
2	adam	3	CHENNAI
4	alex	3	CHENNAI

As you can see, this join returns the cross product of all the records present in both the tables.

INNER Join or EQUI Join

This is a simple JOIN in which the result is based on matched data as per the equality condition specified in the SQL query.

Inner Join Syntax is,

SELECT column-name-list FROM table-name1 INNER JOIN table-name2

WHERE table-name1.column-name = table-name2.column-name;

Example of INNER JOIN

Consider a **class** table,

ID	NAME
1	abhi
2	adam
3	alex
4	anu

and the **class_info** table,

ID	Address
1	DELHI
2	MUMBAI
3	CHENNAI

Inner JOIN query will be,

SELECT * from class INNER JOIN class_info where class.id = class_info.id;

Copy

The resultset table will look like,

ID	NAME	ID	Address
1	abhi	1	DELHI
2	adam	2	MUMBAI
3	alex	3	CHENNAI

Natural JOIN

Natural Join is a type of Inner join which is based on column having same name and same datatype present in both the tables to be joined.

The syntax for Natural Join is,

SELECT * FROM

table-name1 NATURAL JOIN table-name2;

Copy

Example of Natural JOIN

Here is the **class** table,

ID	NAME
1	abhi
2	adam

3	alex
4	anu

and the **class_info** table,

ID	Address
1	DELHI
2	MUMBAI
3	CHENNAI

Natural join query will be,

```
SELECT * from class NATURAL JOIN class_info;
```

Copy

The resultset table will look like,

ID	NAME	Address
1	abhi	DELHI
2	adam	MUMBAI
3	alex	CHENNAI

In the above example, both the tables being joined have **ID** column(same name and same datatype), hence the records for which value of **ID** matches in both the tables will be the result of Natural Join of these two tables.

OUTER JOIN

Outer Join is based on both matched and unmatched data. Outer Joins subdivide further into,

1. Left Outer Join
2. Right Outer Join
3. Full Outer Join

LEFT Outer Join

The left outer join returns a result set table with the **matched data** from the two tables and then the remaining rows of the **left** table and null from the **right** table's columns.

Syntax for Left Outer Join is,

```
SELECT column-name-list FROM
```

```
table-name1 LEFT OUTER JOIN table-name2
```

```
ON table-name1.column-name = table-name2.column-name;
```

Copy

To specify a condition, we use the **ON** keyword with Outer Join.

Left outer Join Syntax for **Oracle** is,

```
SELECT column-name-list FROM
```

```
table-name1, table-name2 on table-name1.column-name = table-name2.column-name(+);
```

Copy

Example of Left Outer Join

Here is the **class** table,

ID	NAME
1	abhi
2	adam
3	alex
4	anu
5	ashish

and the **class_info** table,

ID	Address
1	DELHI
2	MUMBAI
3	CHENNAI
7	NOIDA
8	PANIPAT

Left Outer Join query will be,

```
SELECT * FROM class LEFT OUTER JOIN class_info ON (class.id = class_info.id);
```

Copy

The resultset table will look like,

ID	NAME	ID	Address
1	abhi	1	DELHI
2	adam	2	MUMBAI
3	alex	3	CHENNAI
4	anu	null	null
5	ashish	null	null

RIGHT Outer Join

The right outer join returns a resultset table with the **matched data** from the two tables being joined, then the remaining rows of the **right** table and null for the remaining **left** table's columns. Syntax for Right Outer Join is,

```
SELECT column-name-list FROM  
  
table-name1 RIGHT OUTER JOIN table-name2  
  
ON table-name1.column-name = table-name2.column-name;
```

Copy

Right outer Join Syntax for **Oracle** is,

```
SELECT column-name-list FROM  
  
table-name1, table-name2  
  
ON table-name1.column-name(+) = table-name2.column-name;
```

Copy

Example of Right Outer Join

Once again the **class** table,

ID	NAME
1	abhi
2	adam
3	alex
4	anu
5	ashish

and the **class_info** table,

ID	Address
1	DELHI
2	MUMBAI
3	CHENNAI
7	NOIDA
8	PANIPAT

Right Outer Join query will be,

```
SELECT * FROM class RIGHT OUTER JOIN class_info ON (class.id = class_info.id);
```

Copy

The resultant table will look like,

ID	NAME	ID	Address
1	abhi	1	DELHI
2	adam	2	MUMBAI
3	alex	3	CHENNAI
null	null	7	NOIDA
null	null	8	PANIPAT

Full Outer Join

The full outer join returns a resultset table with the **matched data** of two table then remaining rows of both **left** table and then the **right** table.

Syntax of Full Outer Join is,

```
SELECT column-name-list FROM
table-name1 FULL OUTER JOIN table-name2
ON table-name1.column-name = table-name2.column-name;
```

Copy

Example of Full outer join is,

The **class** table,

ID	NAME
1	abhi
2	adam
3	alex
4	anu
5	ashish

and the **class_info** table,

ID	Address
1	DELHI
2	MUMBAI
3	CHENNAI
7	NOIDA
8	PANIPAT

Full Outer Join query will be like,

```
SELECT * FROM class FULL OUTER JOIN class_info ON (class.id = class_info.id);
```

Copy

The resultset table will look like,

ID	NAME	ID	Address
----	------	----	---------

1	abhi	1	DELHI
2	adam	2	MUMBAI
3	alex	3	CHENNAI
4	anu	null	null
5	ashish	null	null
null	null	7	NOIDA
null	null	8	PANIPAT

SQL UPDATE[5 Marks]

The SQL commands (*UPDATE* and *DELETE*) are used to modify the data that is already in the database. The SQL *DELETE* command uses a *WHERE* clause.

SQL UPDATE statement is used to change the data of the records held by tables. Which rows is to be update, it is decided by a condition. To specify condition, we use *WHERE* clause.

The *UPDATE* statement can be written in following form:

UPDATE table_name **SET** [column_name1= value1,... column_nameN = valueN] [**WHERE** condition]

Let's see the Syntax:

1. **UPDATE** table_name
2. **SET** column_name = expression
3. **WHERE** conditions

Let's take an example: here we are going to update an entry in the source table.

SQL statement:

1. **UPDATE** students
2. **SET** User_Name = 'beinghuman'
3. **WHERE** Student_Id = '3'

Source Table:

Student_Id	FirstName	LastName	User_Name
------------	-----------	----------	-----------

1	Ada	Sharma	sharmili
2	Rahul	Maurya	sofamous
3	James	Walker	jonny

See the result after updating value:

Student_Id	FirstName	LastName	User_Name
1	Ada	Sharma	sharmili
2	Rahul	Maurya	sofamous
3	James	Walker	beinghuman

Updating Multiple Fields:

If you are going to update multiple fields, you should separate each field assignment with a comma.

SQL UPDATE statement for multiple fields:

1. **UPDATE** students
2. **SET** User_Name = 'beserious', First_Name = 'Johnny'
3. **WHERE** Student_Id = '3'

Result of the table is given below:

Student_Id	FirstName	LastName	User_Name
1	Ada	Sharma	sharmili
2	Rahul	Maurya	sofamous
3	Johnny	Walker	beserious

MYSQL SYNTAX FOR UPDATING TABLE:

1. **UPDATE** table_name
2. **SET** field1 = new-value1, field2 = new-value2,

3. [WHERE CLAUSE]

SQL UPDATE SELECT:

SQL UPDATE WITH SELECT QUERY:

We can use SELECT statement to update records through UPDATE statement.

SYNTAX:

1. **UPDATE** tableDestination
2. **SET** tableDestination.col = value
3. **WHERE** EXISTS (
4. **SELECT** col2.value
5. **FROM** tblSource
6. **WHERE** tblSource.join_col = tblDestination. Join_col
7. **AND** tblSource.**Constraint** = value)

You can also try this one -

1. **UPDATE**
2. **Table**
3. **SET**
4. **Table**.column1 = othertable.**column** 1,
5. **Table**.column2 = othertable.**column** 2
6. **FROM**
7. **Table**
8. **INNER** JOIN
9. Other_table
10. **ON**
11. **Table**.id = other_table.id

SQL DELETE [5 Marks]

The **SQL DELETE statement** is used to delete rows from a table. Generally DELETE statement removes one or more records from a table.

SQL DELETE Syntax

Let's see the Syntax for the SQL DELETE statement:

1. **DELETE FROM** table_name [**WHERE** condition]; Here table_name is the table which has to be deleted. The *WHERE clause* in SQL DELETE statement is optional here.

SQL DELETE Example

Let us take a table, named "EMPLOYEE" table.

ID	EMP_NAME	CITY	SALARY
101	Adarsh Singh	Obra	20000
102	Sanjay Singh	Meerut	21000
103	Priyanka Sharma	Raipur	25000
104	Esha Singhal	Delhi	26000

Example of delete with WHERE clause is given below:

1. **DELETE FROM** EMPLOYEE **WHERE** ID=101;

Resulting table after the query:

ID	EMP_NAME	CITY	SALARY
102	Sanjay Singh	Meerut	21000

103	Priyanka Sharma	Raipur	25000
104	Esha Singhal	Delhi	26000

Another example of delete statement is given below

1. **DELETE FROM** EMPLOYEE;

Resulting table after the query:

ID	EMP_NAME	CITY	SALARY
----	----------	------	--------

It will delete all the records of EMPLOYEE table.

It will delete the all the records of EMPLOYEE table where ID is 101.

The WHERE clause in the SQL DELETE statement is optional and it identifies the rows in the column that gets deleted.

WHERE clause is used to prevent the deletion of all the rows in the table, If you don't use the WHERE clause you might loss all the rows.

Note:

Topics of SQL DELETE Statement

SQL DELETE TABLE

How to delete the table and what is the difference between DELETE and TRUNCATE statement?

SQL DELETE ROW

How to delete a row from the database?

SQL DELETE All Rows

How to delete all the rows of a table?

SQL DELETE Duplicate Rows

How to use distinct keyword to delete all the duplicate rows from the table?

SQL DELETE DATABASE

There is not used DELETE statement to delete the database. But, there is used DROP statement to delete the database.

SQL DELETE VIEW

How to delete the view from the database?

SQL DELETE JOIN

How to use delete statement with INNER JOIN?

SQL FOREIGN KEY[5 marks]

In the relational databases, a foreign key is a field or a column that is used to establish a link between two tables.

In simple words you can say that, a foreign key in one table used to point primary key in another table.

Let us take an example to explain it:

Here are two tables first one is students table and second is orders table.

Here orders are given by students.

First table:

S_Id	LastName	FirstName	CITY
-------------	-----------------	------------------	-------------

1	MAURYA	AJEET	ALLAHABAD
2	JAISWAL	RATAN	GHAZIABAD
3	ARORA	SAUMYA	MODINAGAR

Second table:

O_Id	OrderNo	S_Id
1	99586465	2
2	78466588	2
3	22354846	3
4	57698656	1

Here you see that "S_Id" column in the "Orders" table points to the "S_Id" column in "Students" table.

- The "S_Id" column in the "Students" table is the PRIMARY KEY in the "Students" table.
- The "S_Id" column in the "Orders" table is a FOREIGN KEY in the "Orders" table.

The foreign key constraint is generally prevents action that destroy links between tables.

It also prevents invalid data to enter in foreign key column.

SQL FOREIGN KEY constraint ON CREATE TABLE:

(Defining a foreign key constraint on single column)

To create a foreign key on the "S_Id" column when the "Orders" table is created:

MySQL:

1. **CREATE TABLE** orders
2. (
3. O_Id **int** NOT NULL,

4. Order_No **int** NOT NULL,
5. S_Id **int**,
6. PRIMAY **KEY** (O_Id),
7. **FOREIGN KEY** (S_Id) **REFERENCES** Persons (S_Id)
8.)

SQL Server /Oracle / MS Access:

1. **CREATE TABLE** Orders
2. (
3. O_Id **int** NOT NULL PRIMAY **KEY**,
4. Order_No **int** NOT NULL,
5. S_Id **int FOREIGN KEY REFERENCES** persons (S_Id)
6.)

SQL FOREIGN KEY constraint for ALTER TABLE:

If the Order table is already created and you want to create a FOREIGN KEY constraint on the "S_Id" column, you should write the following syntax:

Defining a foreign key constraint on single column:

MySQL / SQL Server / Oracle / MS Access:

1. **ALTER TABLE** Orders
2. **ADD CONSTRAINT** fk_PerOrders
3. **FOREIGN KEY**(S_Id)
4. **REFERENCES** Students (S_Id)

DROP SYNTAX for FOREIGN KEY COSTRAINT:

If you want to drop a FOREIGN KEY constraint, use the following syntax:

MySQL:

1. **ALTER TABLE** Orders
2. ROP **FOREIGN KEY** fk_PerOrders

SQL Server / Oracle / MS Access:

1. **ALTER TABLE** Orders
2. **DROP CONSTRAINT** fk_PerOrders

Difference between primary key and foreign key in SQL:

These are some important difference between primary key and foreign key in SQL-

Primary key cannot be null on the other hand foreign key can be null.

Primary key is always unique while foreign key can be duplicated.

Primary key uniquely identify a record in a table while foreign key is a field in a table that is primary key in another table.

There is only one primary key in the table on the other hand we can have more than one foreign key in the table.

By default primary key adds a clustered index on the other hand foreign key does not automatically create an index, clustered or non-clustered. You must manually create an index for foreign key.

SQL Aliases[5 Marks]

Aliases are the temporary names given to tables or columns for the purpose of a particular [SQL](#) query. It is used when the name of a column or table is used other than its original name, 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 the 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 is more than one table involved in a query.

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

Parameter Explanation

The following table explains the arguments in detail:

- Column_Name: The column name can be defined as the column on which we are going to create an alias name.
- Alias_Name: It can be defined as a temporary name that we are going to assign for the column or table.
- AS: It is optional. If you have not specified it, there is no effect on the query execution.

Syntax 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

Lets see examples for SQL Aliases.

```
CREATE TABLE Customer(
```

```
    CustomerID INT PRIMARY KEY,
```

```
    CustomerName VARCHAR(50),
```

```
    LastName VARCHAR(50),
```

```
    Country VARCHAR(50),
```

```
    Age int(2),
```

```
    Phone int(10)
```

```
);
```

```
-- Insert some sample data into the Customers table
```

```
INSERT INTO Customer (CustomerID, CustomerName, LastName, Country, Age, Phone)
```

```
VALUES (1, 'Shubham', 'Thakur', 'India', '23', 'xxxxxxxxxx'),
```

```
    (2, 'Aman ', 'Chopra', 'Australia', '21', 'xxxxxxxxxx'),
```

```
    (3, 'Naveen', 'Tulasi', 'Sri lanka', '24', 'xxxxxxxxxx'),
```

```
    (4, 'Aditya', 'Arpan', 'Austria', '21', 'xxxxxxxxxx'),
```

```
(5, 'Nishant. Salchichas S.A.', 'Jain', 'Spain', '22', 'xxxxxxxxxx');
```

```
Select * from Customer;
```

Output:

CustomerID	CustomerName	LastName	Country	Age	Phone
1	Shubham	Thakur	India	23	xxxxxxxxxx
2	Aman	Chopra	Australia	21	xxxxxxxxxx
3	Naveen	Tulasi	Sri lanka	24	xxxxxxxxxx
4	Aditya	Arpan	Austria	21	xxxxxxxxxx
5	Nishant. Salchichas S.A.	Jain	Spain	22	xxxxxxxxxx

Example 1: Column Alias

To fetch SSN from the customer table using CustomerID as an alias name.

Query:

```
SELECT CustomerID AS SSN FROM Customer;
```

Output:

SSN
1
2
3
4
5

Example 2: Table Alias

Generally, table aliases are used to fetch the data from more than just a single table and connect them through field relations.

To fetch the CustomerName and Country of the customer with Age = 21.

Query:

```
SELECT s.CustomerName, d.Country
```

```
FROM Customer AS s, Customer
```

```
AS d WHERE s.Age=21 AND
```

```
s.CustomerID=d.CustomerID;
```

Output:

CustomerName	Country
Aman	Australia
Aditya	Austria

Advantages of SQL Alias

1. It is useful when you use the function in the query.
2. It can also allow us to combine two or more columns.
3. It is also useful when the column names are big or not readable.
4. It is used to combine two or more columns.

SQL CONCAT Function [5 marks]

The CONCAT function in SQL is a String function, which is used to merge two or more strings. The Concat service converts the Null values to an Empty string when we display the result. This function is used to concatenate two strings to make a single string. The **operator** is used to link **character strings** and **column string**.

We can use a **literal** in CONCAT Function. A literal is a **number**, **character**, or **date** that includes the SELECT statement.

Syntax of CONCAT function:

1. **SELECT** CONCAT (String 1, String 2, String3..., String N)
FROM [Source]

Example-

1. SQL> **SELECT** CONCAT ('FIRST', 'SECOND');

CONCAT(' FIRST','SECOND')	FIRST SECOND
---------------------------	--------------

To understand the CONCAT function in detail, consider an employee_tbl table, which has the following records -

1. SQL> **SELECT** * **FROM** employee_ tbl ;

ID	NAME	WORK_DATE	DAILY_TYPING_PAGES
1	Michal	2009-02-15	270
2	Zeena	2003-03-24	250
2	kachner	2007-08-19	277
2	warner	2007-04-25	264
3	Joy	2007-05-17	250
4	atire	2006-06-23	270
5	delph	2004-05-28	230

So if we want to concatenate all the names, employee IDs, and work_date of above table, then we can do it using the following command -

1. SQL > **SELECT** CONCAT (id , **name** , work_date)
2. ->**FROM** employee_ tbl;
3. CONCAT(id, **name**, work_date)

1Michal2009-02-15
2Zeena2003-03-24
2kachner2007-08-19
2warner2007-04-25
3joy2007-05-17
4atire2006-06-23
5delph2004-05-28

Example 2:

1. **SELECT** id, first_name, last_name, first_name || last_name,
2. salary, first_name || salary **FROM** myTable
3. **Output** (Third and Fifth Columns show **values** concatenated **by** operator ||)

Output:

id	last_name	first_name	first_name last_name	salary	first_name salary
1	bean	Mr.	Mr.bean	10000	Mr.10000
2	William	Sunita	Sunita William	50000	Sunita50000
3	tpoint	Java	Javatpoint	20000	Java20000
4	&example	tutorial	tutorial&example	90000	Tutorial90000

Note: In above example, we have used "||", which is known as the Concatenation operator, and it is used to link two or more columns in select query. This operator is independent of the data type of column.

Here, we have linkined 2 columns i.e, first_name+last_name as well as first_name+salary.

We can use **string literals** in CONCAT operator.

Example 1: Using the character literal

Syntax

1. **SELECT** id, first_name, last_name, salary,
2. first_name||' has salary '||salary **as** "new" **FROM** myTable

Output: (Concatenating three values and giving a new 'name')

id	first_name	last_name	salary	new
1	Javatpoint	tpoint	20000	Java has salary 20000
2	tutorial	&example	30000	the tutorial has salary 30000
3	Shane	Watson	40000	Shane has salary 40000
4	Jennifer	louse	60000	Jennifer has salary 60000

Note: We have used salary as a character literal in the select statement. We can use the date literal and number literal according to our requirement in the table.

Example 2: Using character as well as the number literal

Syntax:

1. **SELECT** id, first_name, last_name, salary, first_name||100||'

2. has id '||id AS "new" FROM myTable
3. **Output** (Making the **output** readable **by** concatenating a string
4. **with values**)

Output:

id	first_name	last_name	salary	new
1	Javatpoint	tpoint	20000	Java100 has id 1
2	tutorial	&example	30000	Tutorial100 has id 2
3	Shane	Watson	40000	Shane100 has id 3
4	Jennifer	louse	60000	Jennifer100 has id 4

In the above example, we have used **the salary** as a character literal as well as **100** as number authentic in our select statement.

Chapter II

App Development with Angular:

- 1) Angular 2 App from Scratch
- 2) Components & Properties
- 3) Events & Binding With ngModel
- 4) Fetch Data from A Service
- 5) Submit Data to Service
- 6) Http Module & Observables
- 7) Routing

1 Angular 2 App from Scratch:

Angular JS is an open source framework built over JavaScript. It was built by the developers at Google. This framework was used to overcome obstacles encountered while working with Single Page applications. Also, testing was considered as a key aspect while building the framework. It was ensured that the framework could be easily tested. The initial release of the framework was in October 2010.

Features of Angular 2

Following are the key features of Angular 2 –

- **Components** – The earlier version of Angular had a focus of Controllers but now has changed the focus to having components over controllers. Components help to build the applications into many modules. This helps in better maintaining the application over a period of time.
- **TypeScript** – The newer version of Angular is based on TypeScript. This is a superset of JavaScript and is maintained by Microsoft.
- **Services** – Services are a set of code that can be shared by different components of an application. So for example if you had a data component that picked data from a database, you could have it as a shared service that could be used across multiple applications.

In addition, Angular 2 has better event-handling capabilities, powerful templates, and better support for mobile devices.

Components of Angular 2

Angular 2 has the following components –

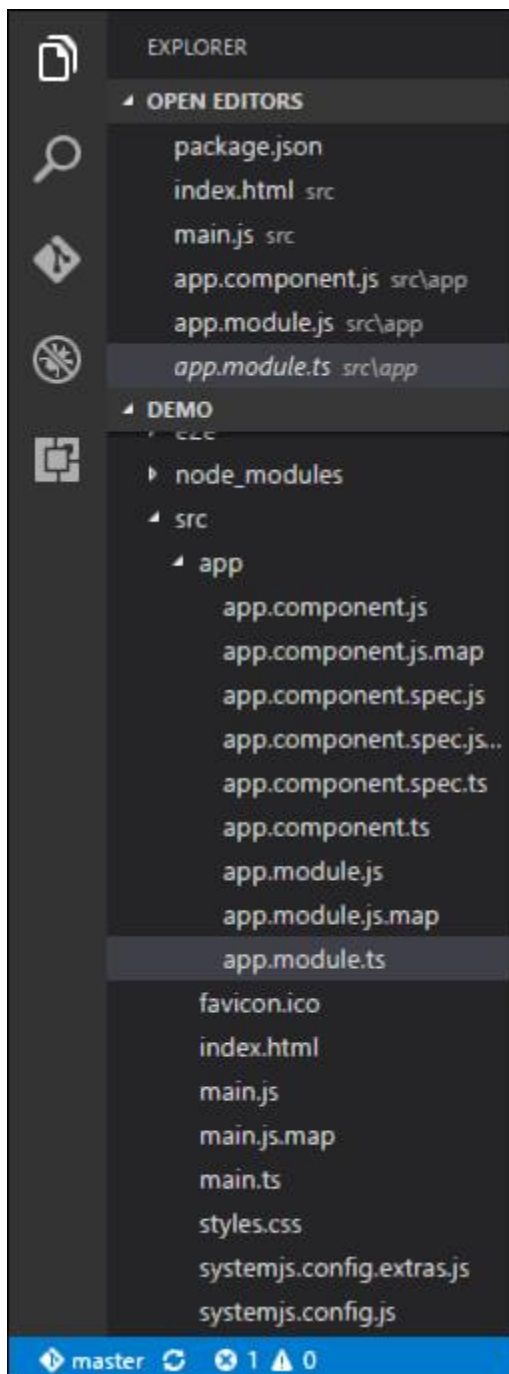
- **Modules** – This is used to break up the application into logical pieces of code. Each piece of code or module is designed to perform a single task.
- **Component** – This can be used to bring the modules together.
- **Templates** – This is used to define the views of an Angular JS application.
- **Metadata** – This can be used to add more data to an Angular JS class.
- **Service** – This is used to create components which can be shared across the entire application.

To start working with Angular 2, you need to get the following key components installed.

- **Npm** – This is known as the node package manager that is used to work with the open source repositories. Angular JS as a framework has dependencies on other components. And **npm** can be used to download these dependencies and attach them to your project.
- **Git** – This is the source code software that can be used to get the sample application from the **github** angular site.
- **Editor** – There are many editors that can be used for Angular JS development such as Visual Studio code and WebStorm. In our tutorial, we will use Visual Studio code which comes free of cost from Microsoft.

Modules are used in Angular JS to put logical boundaries in your application. Hence, instead of coding everything into one application, you can instead build everything into separate modules to separate the functionality of your application. Let's inspect the code which gets added to the demo application.

In Visual Studio code, go to the app.module.ts folder in your app folder. This is known as the root module class.



The following code will be present in the **app.module.ts** file.

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';

@NgModule ({
  imports: [ BrowserModule ],
  declarations: [ AppComponent ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```

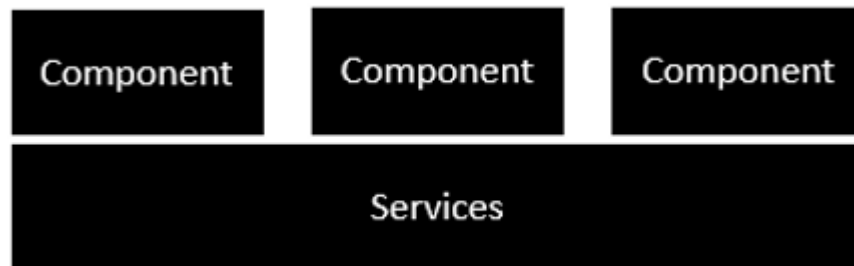
Let's go through each line of the code in detail.

- The import statement is used to import functionality from the existing modules. Thus, the first 3 statements are used to import the NgModule, BrowserModule and AppComponent modules into this module.
- The NgModule decorator is used to later on define the imports, declarations, and bootstrapping options.
- The BrowserModule is required by default for any web based angular application.
- The bootstrap option tells Angular which Component to bootstrap in the application.

A module is made up of the following parts –

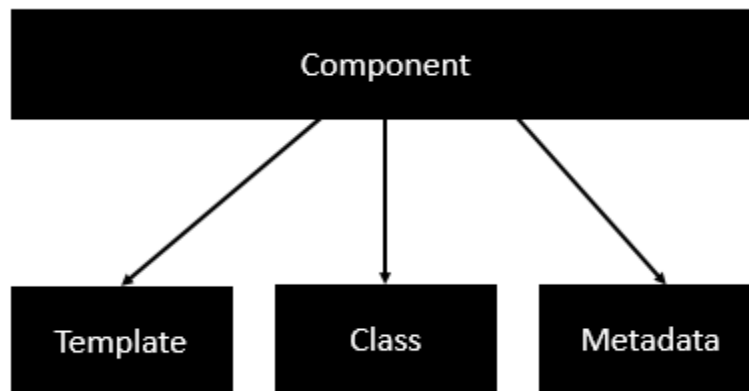
- **Bootstrap array** – This is used to tell Angular JS which components need to be loaded so that its functionality can be accessed in the application. Once you include the component in the bootstrap array, you need to declare them so that they can be used across other components in the Angular JS application.
- **Export array** – This is used to export components, directives, and pipes which can then be used in other modules.
- **Import array** – Just like the export array, the import array can be used to import the functionality from other Angular JS modules.

The following screenshot shows the anatomy of an Angular 2 application. Each application consists of Components. Each component is a logical boundary of functionality for the application. You need to have layered services, which are used to share the functionality across components.



Following is the anatomy of a Component. A component consists of –

- **Class** – This is like a C++ or Java class which consists of properties and methods.
- **Metadata** – This is used to decorate the class and extend the functionality of the class.
- **Template** – This is used to define the HTML view which is displayed in the application.



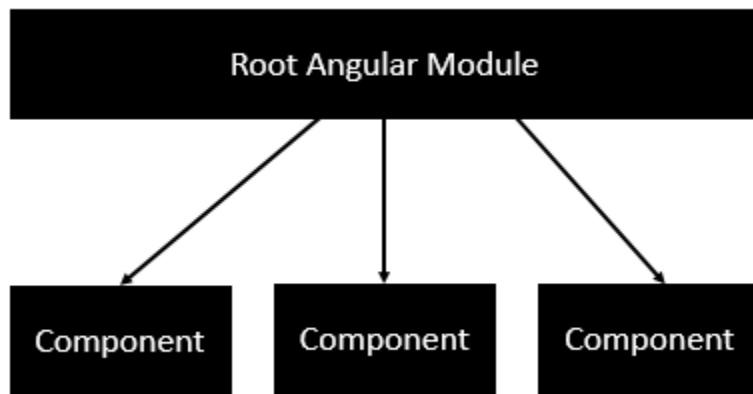
Following is an example of a component.

```
import { Component } from '@angular/core';

@Component ({
  selector: 'my-app',
  templateUrl: 'app/app.component.html'
})

export class AppComponent {
  appTitle: string = 'Welcome';
}
```

Each application is made up of modules. Each Angular 2 application needs to have one Angular Root Module. Each Angular Root module can then have multiple components to separate the functionality.

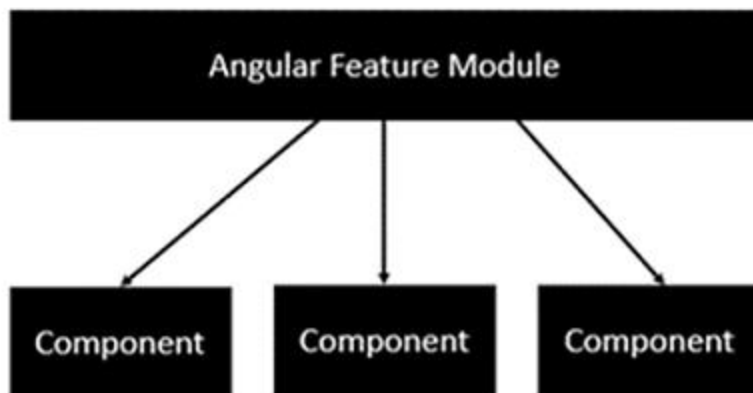


Following is an example of a root module.

```
import { NgModule }   from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';

@NgModule ({
  imports:    [ BrowserModule ],
  declarations: [ AppComponent ],
  bootstrap:  [ AppComponent ]
})
export class AppModule { }
```

Each application is made up of feature modules where each module has a separate feature of the application. Each Angular feature module can then have multiple components to separate the functionality.

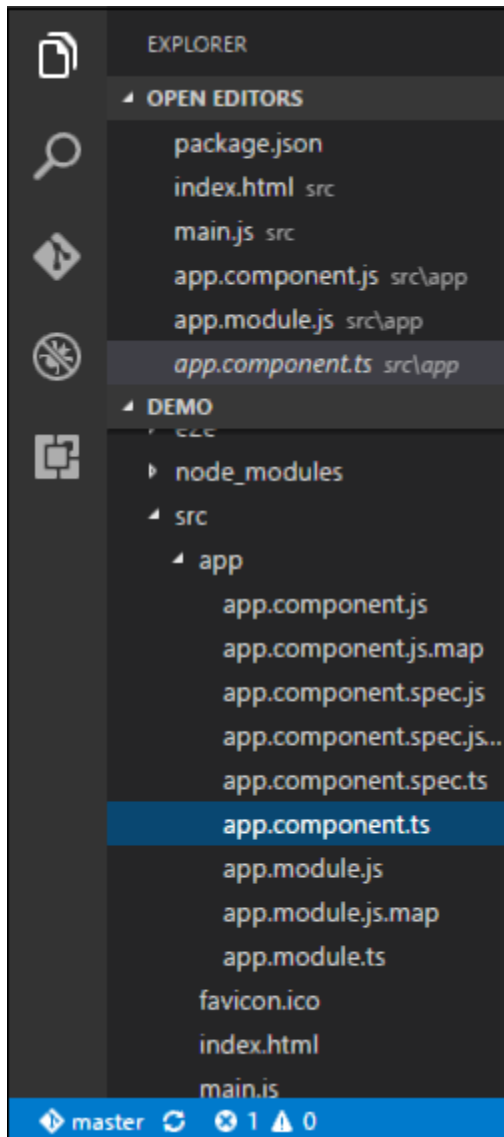


2 Angular 2 Components & Properties[5 marks]

Components are a logical piece of code for Angular JS application. A Component consists of the following –

- **Template** – This is used to render the view for the application. This contains the HTML that needs to be rendered in the application. This part also includes the binding and directives.
- **Class** – This is like a class defined in any language such as C. This contains properties and methods. This has the code which is used to support the view. It is defined in TypeScript.
- **Metadata** – This has the extra data defined for the Angular class. It is defined with a decorator.

Let's now go to the app.component.ts file and create our first Angular component.



Let's add the following code to the file and look at each aspect in detail.

Class

The class decorator. The class is defined in TypeScript. The class normally has the following syntax in TypeScript.

Syntax

```
class classname {  
  Propertyname: PropertyType = Value  
}
```

Parameters

- **Classname** – This is the name to be given to the class.
- **Propertyname** – This is the name to be given to the property.

- **PropertyType** – Since TypeScript is strongly typed, you need to give a type to the property.
- **Value** – This is the value to be given to the property.

Example

```
export class AppComponent {
  appTitle: string = 'Welcome';
}
```

In the example, the following things need to be noted –

- We are defining a class called AppComponent.
- The export keyword is used so that the component can be used in other modules in the Angular JS application.
- appTitle is the name of the property.
- The property is given the type of string.
- The property is given a value of 'Welcome'.

Template

This is the view which needs to be rendered in the application.

Syntax

```
Template: '
  <HTML code>
  class properties
'
```

Parameters

- **HTML Code** – This is the HTML code which needs to be rendered in the application.
- **Class properties** – These are the properties of the class which can be referenced in the template.

Example

```
template: '
  <div>
    <h1>{{ appTitle }}</h1>
    <div>To Tutorials Point</div>
  </div>
'
```

In the example, the following things need to be noted –

- We are defining the HTML code which will be rendered in our application
- We are also referencing the appTitle property from our class.

Metadata

This is used to decorate Angular JS class with additional information.

Let's take a look at the completed code with our class, template, and metadata.

Example

```
import { Component } from '@angular/core';

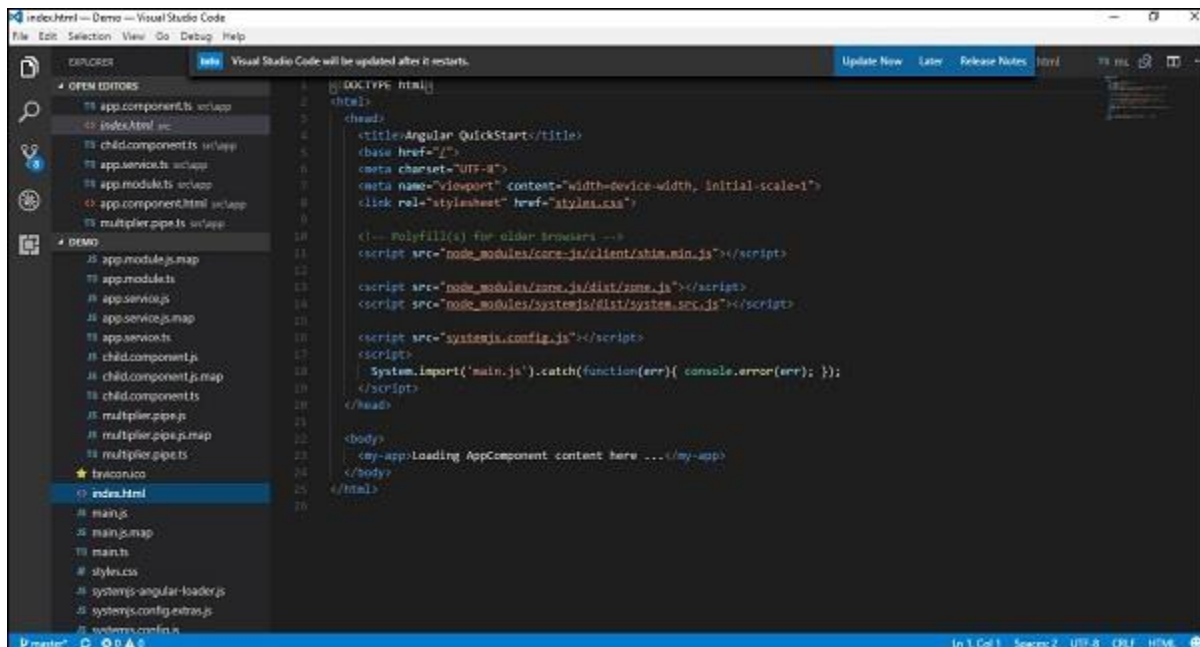
@Component ({
  selector: 'my-app',
  template: `<div>
    <h1>{{ appTitle }}</h1>
    <div>To Tutorial Point</div>
  </div>`,
})

export class AppComponent {
  appTitle: string = 'Welcome';
}
```

In the above example, the following things need to be noted –

- We are using the import keyword to import the 'Component' decorator from the angular/core module.
- We are then using the decorator to define a component.
- The component has a selector called 'my-app'. This is nothing but our custom html tag which can be used in our main html page.

Now, let's go to our index.html file in our code.

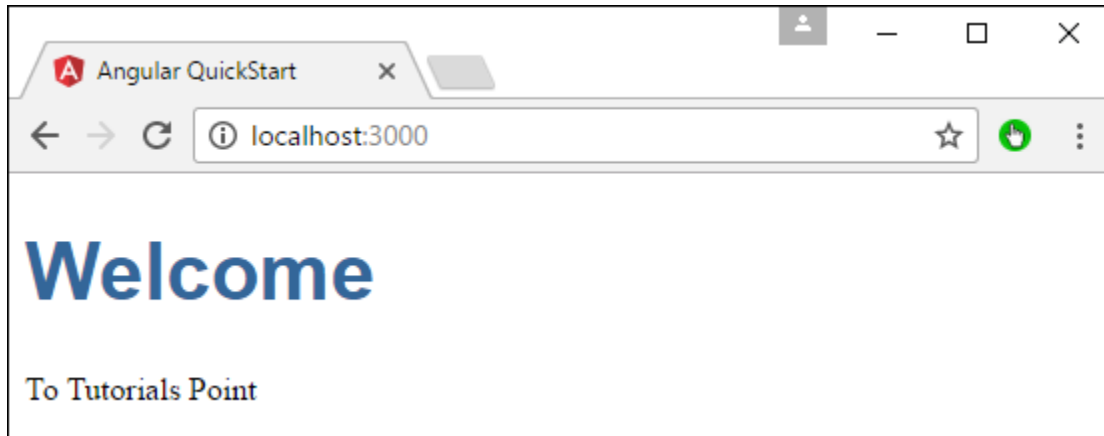


Let's make sure that the body tag now contains a reference to our custom tag in the component. Thus in the above case, we need to make sure that the body tag contains the following code –

```
<body>
  <my-app></my-app>
</body>
```

Output

Now if we go to the browser and see the output, we will see that the output is rendered as it is in the component.

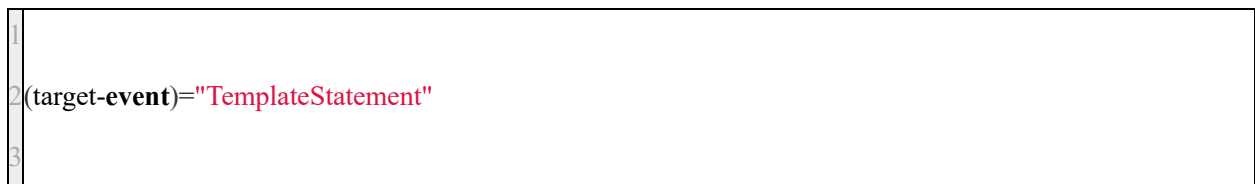


Angular Event Binding [10 marks]

Event binding allows us to bind events such as keystroke, clicks, hover, touche, etc to a method in component. It is one way from view to component. By tracking the user events in the view and responding to it, we can keep our component in sync with the view. For Example, when the user changes to an input in a text box, we can update the model in the component, run some validations, etc. When the user submits the button, we can then save the model to the backend server.

Syntax

The Angular event binding consists of two parts

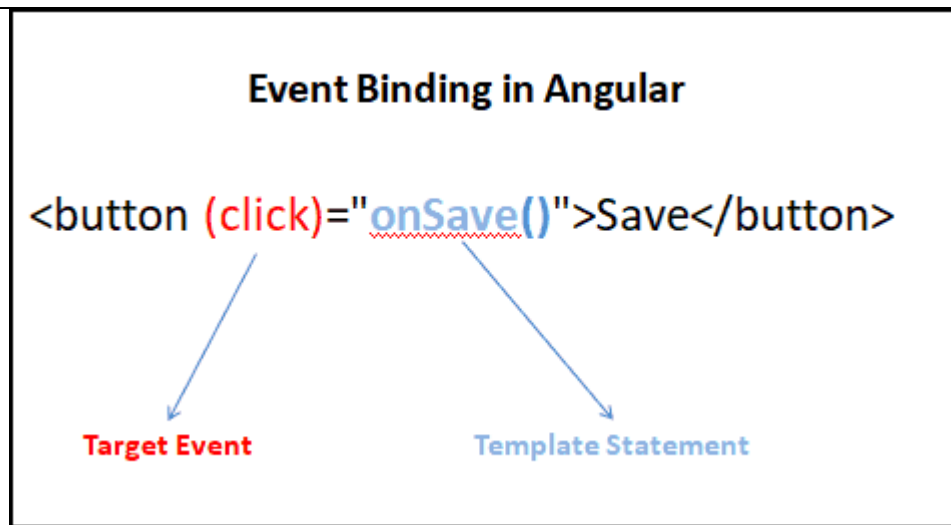


- We enclose the target event name in parentheses on the left side
- Assign it to a template statement within a quote on the right side

Angular event binding syntax consists of a target event name within parentheses on the left of an equal sign, and a quoted template statement on the right.

The following event binding listens for the button's click events, calling the component's `onSave()` method whenever a click occurs

```
1
2 <button (click)="onSave()">Save</button>
3
```



Event Binding Example

Create a new angular application

```
1
2 ng new event
3
```

Copy the following code to `app.component.html`

app.component.html

```
1
2 <h1 [innerText]="title"></h1>
```

```
3
4<h2>Example 1</h2>
5<button (click)="clickMe()">Click Me</button>
6<p>You have clicked {{clickCount}}</p>
7
```

[Source Code](#)

Add the following code to the `app.component.ts`

app.component.ts

```
1
2clickCount=0
3 clickMe() {
4   this.clickCount++;
5 }
6
```

[Source Code](#)

In the above example, the component listens to the click event on the button. It then executes the `clickMe()` method and increases the `clickCount` by one.

Template statements have side effects

Unlike the [Property Binding](#) & [Interpolation](#), where we use the template expression is used, in the case of event binding we use template statement.

The Template statement can change the state of the component. Angular runs the change detection and updates the view so as to keep it in sync with the component.

on-

Instead of parentheses, you can also use the `on-` syntax as shown below.

```
1
2<button on-click="clickMe()">Click Me</button>
3
```

Multiple event handlers

You can also bind an unlimited number of event handlers on the same event by separating them with a semicolon ;

Add a new component property

```
1
2 clickCount1=0;
3
```

[Source Code](#)

And in the template, call `clickMe()` method and then an assignment `clickCount1=clickCount`

```
1
2 //Template
3
4 <h2>Example 2</h2>
5 <button (click)="clickMe() ; clickCount1=clickCount">Click Me</button>
6 <p>You have clicked {{clickCount}}</p>
7 <p>You have clicked {{clickCount1}}</p>
8
```

[Source Code](#)

\$event Payload

DOM Events carries the event payload. I.e the information about the event. We can access the event payload by using `$event` as an argument to the handler function.

```
1
2 <input (input)="handleInput($event)">
3 <p>You have entered {{value}}</p>
4
```

[Source Code](#)

And in the component

```

1
2 value=""
3 handleInput(event) {
4   this.value = (event.target as HTMLInputElement).value;
5 }
6

```

[Source Code](#)

The properties of a `$event` object vary depending on the type of DOM event. For example, a mouse event includes different information than an input box editing event.

Remember you need to use the variable as `$event` in the Template statement.

Example `handleInput($event)`. Otherwise, it will result in an error

Template reference variable

We can also make use of the template reference variable to pass the value instead of `$event`.

In the template

```

1
2 <h2>Template Reference Variable</h2>
3 <input #el (input)="handleInput1(el)">
4 <p>You have entered {{val}}</p>
5

```

[Source Code](#)

```

1
2 val="";
3 handleInput1(element) {
4   this.val=element.value;
5 }
6
7

```

[Source Code](#)

Key event filtering (with key.enter)

We use keyup/keydown events to listen for keystrokes. The following example does that

```
1
2<input (keyup)="value1= $any($event.target).value" />
3<p>You entered {{value1}}</p>
4
```

[Source Code](#)

But Angular also offers a feature, where it helps to filter out certain keys. For Example, if you want to listen only to the enter keys you can do it easily

```
1
2<input (keyup.enter)="value2=$any($event.target).value">
3<p>You entered {{value2}}</p>
4
```

[Source Code](#)

Here is an interesting example. On pressing enter key it updates the value3 variable and on escape clears the variable.

```
1
2<input (keyup.enter)="value3=$any($event.target).value"
3(keyup.escape)="$any($event.target).value='';value3=''">
4<p>You entered {{value3}}</p>
```

[Source Code](#)

Note that we are using \$any to cast \$event.target to [any type](#). Otherwise, the typescript will raise the error [Property 'value' does not exist on type 'EventTarget' Error in Angular](#)

Angular calls these pseudo-events.

You can also listen for the key combination

```
1
```

```
2<input (keyup.control.shift.enter)="value4=$any($event.target).value">
3<p>You entered {{value4}}</p>
4
```

[Source Code](#)

Custom events with EventEmitter

Directives & components can also raise events with [EventEmitter](#).

Using [EventEmiiter](#) you can create a property and raise it using the `EventEmitter.emit(payload)`. The Parent component can listen to these events using the event binding and also read the payload using the `$event` argument.

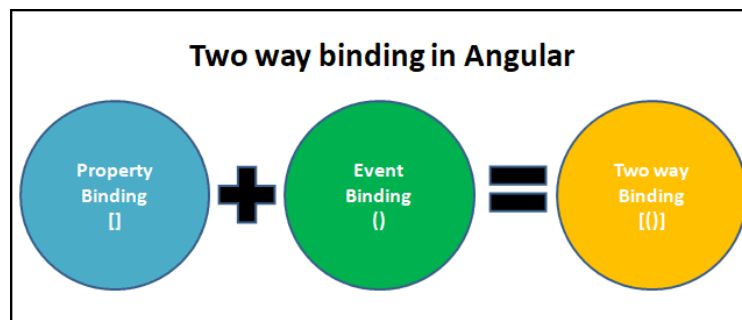
ngModel & Two way Data binding in Angular [10 marks]:

Two way data binding:

Two way data binding means that changes made to our model in the component are propagated to the view and that any changes made in the view are immediately updated in the underlying component data.

Two way data binding is useful in data entry forms. Whenever a user makes changes to a form field, we would like to update our model. Similarly, when we update the model with new data, we would like to update the view as well

The two way data binding is nothing but both property binding & event binding applied together. Property Binding is one way from component to view. The event binding is one way from view to component. If we combine both we will get the Two-way binding.



Two way using property & Event Binding:

The following example shows how we can achieve two-way binding using the combination of property binding & event binding

Create a new Angular application

copy the following code to `app.component.html`

```
1
2<h2>Example 1</h2>
3<input type="text" [value]="name" (input)="name=$event.target.value">
4<p> You entered {{name}} </p>
5<button (click)="clearName()">Clear</button>
6
```

Update the `app.component.ts` with the following code.

```
1
2 name=""
3 clearName() {
4   this.name="";
5 }
6
7
```

We bind the name property to the input element (`[value]="name"`). We also use the event binding `(input)="name=$event.target.value"`. It updates the name property whenever the input changes. The Angular interpolation updates the `{{name}}`, so we know the value of name property.

`$event.target.value` raises the error Property 'value' does not exist on type 'EventTarget' if `fullTemplateTypeCheck` is set to true under `angularCompilerOptions` in the `tsconfig.json`.

The error is due to the fact that the value property is not guaranteed to exist in the `$event.target`.

To solve this problem either you can use the `$any` typecast function

(\$any(\$event.target).value) to stop the type checking in the template or set fullTemplateTypeCheck to false in tsconfig.json.

Two-way binding syntax:

The above example uses the event & property binding combination to achieve the two-way binding. But Angular does provide a way to achieve the two-way binding using the syntax `[()]`. Note that both square & parentheses are used here. This is now known as **Banana in a box** syntax. The square indicates the Property binding & parentheses indicates the event binding.

For Example

```
1  
2<someElement [(someProperty)]="value"></someElement>  
3
```

The above syntax sets up both property & event binding. But to make use of it, the property must follow the following naming convention.

But most HTML elements have a `value` property. But do not have a `valueChange` event, instead, they usually have an `input` event. Hence they cannot be used in the above syntax

For Example, the following will not work as there is no `valueChange` event supported by the `input` element.

Hence we have a `ngModel` directive.

ngModel

The Angular uses the `ngModel` directive to achieve the two-way binding on HTML Form elements. It binds to a form element like `input`, `select`, `selectarea`. etc.

Internally It uses the `ngModel` in property, binding to bind to the `value` property and `ngModelChange` which binds to the `input` event.

How to use ngModel

The `ngModel` directive is not part of the Angular Core library. It is part of the `FormsModule` library. You need to import the `FormsModule` package into your Angular module.

In the template use the following syntax

```
1  
2<input type="text" name="value" [(ngModel)]="value">
```

```
3
```

The `ngModel` directive placed inside the square & parentheses as shown above. This is assigned to the Template Expression. Template Expression is the property in the component class

ngModel Example

Import FormsModule

Open the `app.module.ts` and make the following changes

```
1
2import { FormsModule } from '@angular/forms';
3
```

Template

```
1
2<h2>Example 2</h2>
3<input type="text" name="value" [(ngModel)]="value">
4<p> You entered {{value}} </p>
5<button (click)="clearValue()">Clear</button>
6
```

Component

```
1
2value="";
3clearValue() {
4  this.value="";
5}
6
7
```

The `ngModel` data property sets the element's value property and the `ngModelChange` event property listens for changes to the element's value. Run the project and see that as you modify the name, the component class model is automatically updated.

Custom Two-way binding:

As we mentioned earlier the `[(ngModel)]` to work, we need to have a property with the change event as `<nameofProperty>Change`.

We do not have any HTML Elements which follows the above naming conventions, but we can create a custom component

create new component and name it as `counter.component.ts`. copy the following code.

```
1
2import { Component, Input, Output, EventEmitter } from '@angular/core';
3@Component({
4  selector: 'counter',
5  template: `
6    <div>
7      <p>
8        Count: {{ count }}
9        <button (click)="increment()">Increment</button>
10     </p>
11   </div>
12 `
13})
14export class CounterComponent {
15
16  @Input() count: number = 0;
17  @Output() countChange: EventEmitter<number> = new EventEmitter<number>();
18
19  increment() {
20    this.count++;
21    this.countChange.emit(this.count);
22  }
```

23}
24
25

The component has two properties one is input property count decorated with @Input(). The other in is an event (or output property), which we decorate with @Output(). We name the input property as count. Hence the output property becomes countChange. Now we can use this component and create two-way binding to the count property using the syntax [(count)].

```
1  
2<h2>Example 3</h2>  
3<counter [(count)]="count"></counter>  
4<p> Current Count {{count}} </p>  
5<button (click)="clearCount()">Clear</button>  
6
```

The two-way binding is a simple, but yet powerful mechanism. We use Property binding & Event binding to achieve the two-way binding. Angular does have a [(value)] syntax to which sets up the two-way binding. It automatically sets up property binding to the value property of the element. It also sets up the event binding to valueChange Property. But since we hardly have any HTML element, which follows those naming conventions unless we create our own component. This is where ngModel directive from FormsModule steps in and provides two way binding to all the known HTML form elements.

NgModelChange Example [10 marks]

The following is the simple example of ngModelChange.

We assign the method in the component class (handler function) to the ngModelChange using the [event binding](#) syntax

```
1  
2//Template  
3  
4Name:  
5<input type="text" name="name" ngModel (ngModelChange)="nameChanged($event)">
```

```
6
```

nameChanged is the handler function, which we need to define in the component class. We can access the *new value* by using the \$event as an argument to the handler function.

```
1
2//Component
3nameChanged(arg) {
4  console.log("modelchanged " + arg);
5}
6
```

ngModel

We usually use the ngModel as follows to achieve the [two-way](#) binding. [(ngModel)]= "email" keeps the email property in the component class in sync with the template.

```
1
2<input type="text" name="email" [(ngModel)]= "email">
3
```

Internally, Angular converts the above syntax to the following syntax.

```
1
2<input [(ngModel)]= "email" (ngModelChange)= "email = $event">
3
```

The component property email is bound to the input element using the [property binding](#) ([(ngModel)]= "email"). Any changes made to the input is updated in the component using the (ngModelChange)= "email = \$event" [event binding](#).

ngModelChange with ngModel

Consider the following example.

```
1
2<input [(ngModel)]= "firstName" (ngModelChange)= "firstNameChanged($event)"/>
3
```

The Angular converts the above to the following syntax. We end up with the two ngModelChange event bindings.

```
1
2<input [ngModel]="firstName (ngModelChange)="firstName =
3$event" (ngModelChange)="firstNameChanged($event)"/>
```

Here the ngModelChange fires in the order it is specified. Hence the (ngModelChange)="firstName = \$event" fires first. (ngModelChange)="firstNameChanged(\$event)" fires later. Hence in the component class, the arg & this.firstName is always the same.

```
1
2firstName
3;
4
5firstNameChanged(arg) {
6  console.log(
7    "firstNameChanged argument " + arg + " component " + this.firstName
8  );
9}
10
```

But if you put ngModelChange ahead of the ngModel as in the example below

```
1
2<input (ngModelChange)="lastNameChanged($event)" [(ngModel)]="lastName" />
3
4
```

Angular internally converts it as follows

```
1
2<input (ngModelChange)="lastNameChanged($event)" [ngModel]="lastName"
3  (ngModelChange)="lastName= $event" />
```

3

Here `(ngModelChange)="lastNameChanged($event)"` fires first. Hence in the component class `arg` contains the latest value of the, while `this.lastName` **still holds the previous value**

1
2
3
4
5
6
7
8
9
10

```
lastName  
;  
  
lastNameChanged(arg) {  
  console.log(  
    "lastNameChanged argument " + arg + " component " + this.lastName  
  );  
}
```

Change Event

The `(change)` is a DOM event fires when changes to the form fields like `<input>`, `<select>`, and `<textarea>` is committed by the user. This event fires when

- user changes the input & moves the focus away from the text box (blur event)
- On `<select>` it fires when the user selects a new option either by a mouse click or using a keyboard.
- Fires when the state of a check box or radio button change due to users action

Change Event Example

The following example shows how to use the `change` event in Angular.

1
2
3
4

```
Name  
  
<input type="text" name="name1" (change)="name1Changed($event)">
```



```

5
6<br>
7country
8<select name="country1" (change)="country1Changed($event)" >
9  <option [ngValue]="null" disabled>Select Country</option>
10  <option *ngFor="let country of countries" [ngValue]="country.id">{{country.name}}</option>
11</select>
12

```

Component class

```

1
2name1Changed(arg) {
3  console.log("name1Changed " + arg.target.value);
4  console.log(arg);
5}
6
7country1Changed(arg) {
8  console.log("country1Changed " + arg.target.value);
9  console.log(arg);
10}
11

```

The change event for text element fires when we move the focus away from the element (blurred the input). This is different from the `ngModelChange`, which fires the event for each input change.

The other import point is the `$event` parameter. In the `ngModelChange`, it is always the new value. But in the case of a change event, it is event data. The event data is an object containing data about the event. We need to use the `target.value` to access the value.

NgModelChange Vs Change[2 marks]

NgModelChange	Change
NgModelChange is Angular specific event	Change is a DOM Event and has nothing to do with the Angular.
We must use the ngModelChange along with the ngModel directive	You can use change event on <input>, <select>, and <textarea> form elements.
ngModelChange event passes new value	Change event passes event parameter, Use the target.value to access the new value
ngModelChange will trigger with each input change	Change event fires when you remove the focus from input text after changing the content.

4 Fetch Data from A Service & Submit Data to Service [10 marks]

Http Service will help us fetch external data, post to it, etc. We need to import the http module to make use of the http service. Let us consider an example to understand how to make use of the http service.

To start using the http service, we need to import the module in **app.module.ts** as shown below –

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
import { HttpClientModule } from '@angular/http';
import { AppComponent } from './app.component';
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    BrowserAnimationsModule,
    HttpClientModule
  ],
  providers: [],
  bootstrap: [AppComponent]
```

```
})  
export class AppModule {}
```

If you see the highlighted code, we have imported the `HttpModule` from `@angular/http` and the same is also added in the imports array.

Let us now use the `http` service in the **app.component.ts**.

```
import { Component } from '@angular/core';  
import { Http } from '@angular/http';  
import 'rxjs/add/operator/map';  
@Component({  
  selector: 'app-root',  
  templateUrl: './app.component.html',  
  styleUrls: ['./app.component.css']  
})  
export class AppComponent {  
  constructor(private http: Http) {}  
  ngOnInit() {  
    this.http.get("http://jsonplaceholder.typicode.com/users").  
      map((response) => response.json()).  
      subscribe((data) => console.log(data))  
  }  
}
```

Let us understand the code highlighted above. We need to import `http` to make use of the service, which is done as follows –

```
import { Http } from '@angular/http';
```

In the class **AppComponent**, a constructor is created and the private variable `http` of type `Http`. To fetch the data, we need to use the **get API** available with `http` as follows

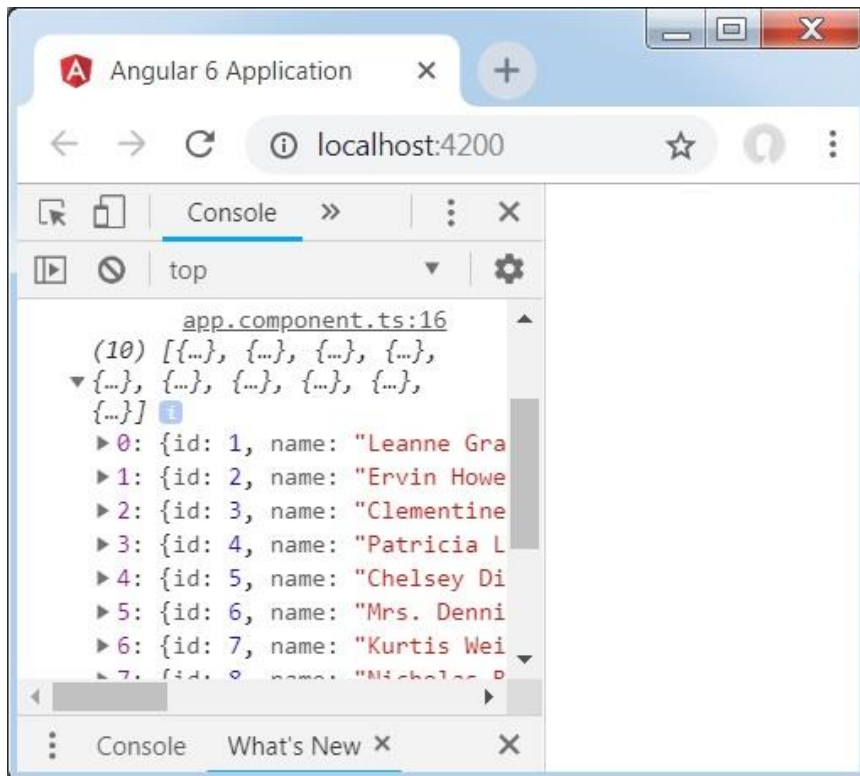
```
this.http.get();
```

It takes the url to be fetched as the parameter as shown in the code.

We will use the test url – <https://jsonplaceholder.typicode.com/users> to fetch the json data. Two operations are performed on the fetched url data `map` and `subscribe`. The `Map` method helps to convert the data to json format. To use the `map`, we need to import the same as shown below –

```
import { map } from 'rxjs/operators';
```

Once the `map` is done, the `subscribe` will log the output in the console as shown in the browser –



If you see, the json objects are displayed in the console. The objects can be displayed in the browser too.

For the objects to be displayed in the browser, update the codes in **app.component.html** and **app.component.ts** as follows –

```
import { Component } from '@angular/core';
import { Http } from '@angular/http';
import { map } from 'rxjs/operators';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  constructor(private http: Http) {}
  httpdata;
  ngOnInit() {
    this.http.get("http://jsonplaceholder.typicode.com/users")
      .pipe(map((response) => response.json()))
      .subscribe((data) => this.displaydata(data));
  }
  displaydata(data) {this.httpdata = data;}
}
```

In **app.component.ts**, using the subscribe method we will call the display data method and pass the data fetched as the parameter to it.

In the display data method, we will store the data in a variable httpdata. The data is displayed in the browser using **for** over this httpdata variable, which is done in the **app.component.html** file.

```
<ul *ngFor = "let data of httpdata">
  <li>Name : {{data.name}} Address: {{data.address.city}}</li>
</ul>
```

The json object is as follows –

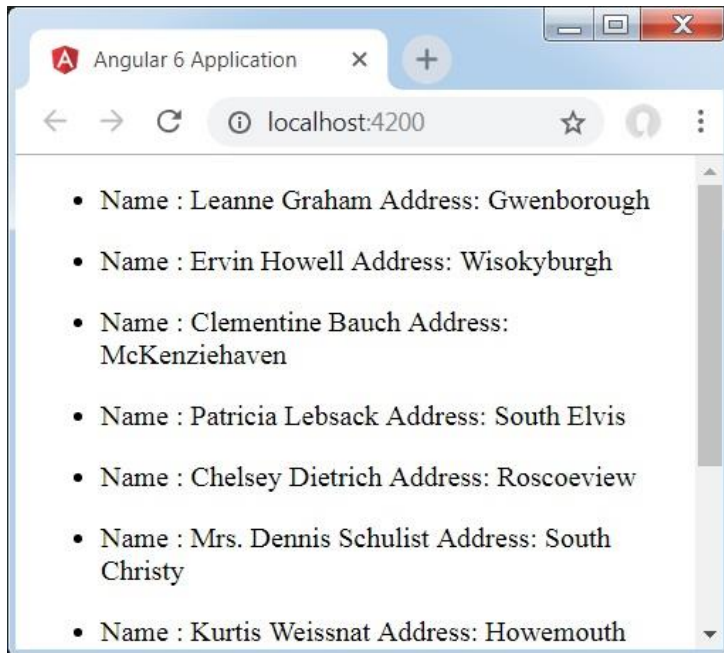
```
{
  "id": 1,
  "name": "Leanne Graham",
  "username": "Bret",
  "email": "Sincere@april.biz",

  "address": {
    "street": "Kulas Light",
    "suite": "Apt. 556",
    "city": "Gwenborough",
    "zipcode": "92998-3874",
    "geo": {
      "lat": "-37.3159",
      "lng": "81.1496"
    }
  },

  "phone": "1-770-736-8031 x56442",
  "website": "hildegard.org",
  "company": {
    "name": "Romaguera-Crona",
    "catchPhrase": "Multi-layered client-server neural-net",
    "bs": "harness real-time e-markets"
  }
}
```

The object has properties such as id, name, username, email, and address that internally has street, city, etc. and other details related to phone, website, and company. Using the **for** loop, we will display the name and the city details in the browser as shown in the **app.component.html** file.

This is how the display is shown in the browser –



Let us now add the search parameter, which will filter based on specific data. We need to fetch the data based on the search param passed.

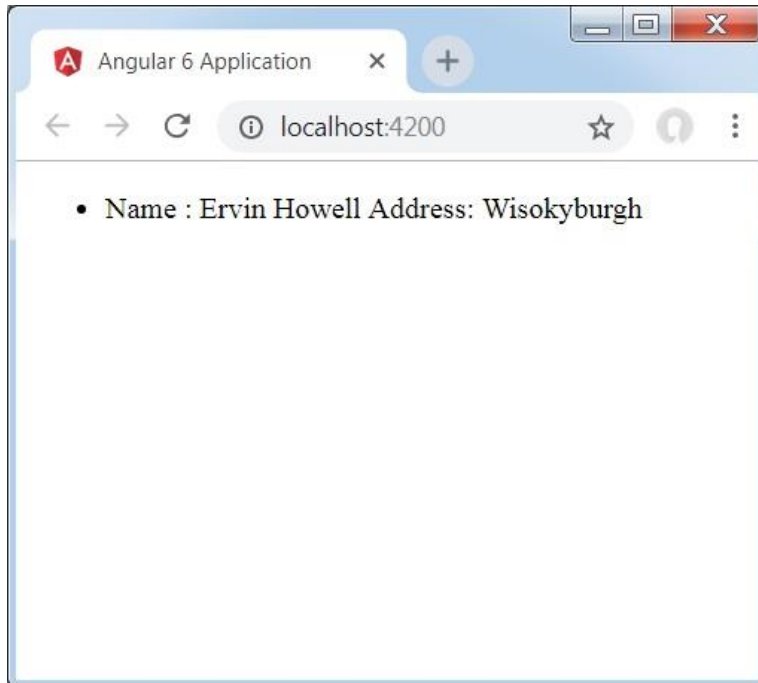
Following are the changes done in **app.component.html** and **app.component.ts** files –

app.component.ts

```
import { Component } from '@angular/core';
import { Http } from '@angular/http';
import { map } from 'rxjs/operators';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  constructor(private http: Http) {}
  httpdata;
  name;
  searchparam = 2;
  ngOnInit() {
    this.http.get("http://jsonplaceholder.typicode.com/users?id="+this.searchparam)
      .pipe(map((response) => response.json()))
      .subscribe((data) => this.displaydata(data));
  }
  displaydata(data) {this.httpdata = data;}
}
```

For the **get api**, we will add the search param `id = this.searchparam`. The searchparam is equal to 2. We need the details of **id = 2** from the json file.

This is how the browser is displayed –



We have console the data in the browser, which is received from the http. The same is displayed in the browser console. The name from the json with **id = 2** is displayed in the browser.

5.module in Angular:

In AngularJS, a module defines an application. It is a container for the different parts of your application like controller, services, filters, directives etc.

A module is used as a `Main()` method. Controller always belongs to a module.

How to create a module

The angular object's `module()` method is used to create a module. It is also called AngularJS function `angular.module`

1. `<div ng-app="myApp">...</div>`

2. `<script>`
3. `var app = angular.module("myApp", []);`
4. `</script>`

Here, "myApp" specifies an HTML element in which the application will run.

Now we can add controllers, directives, filters, and more, to AngularJS application.

How to add controller to a module

If you want to add a controller to your application refer to the controller with the ng-controller directive.

See this example:

1. `<!DOCTYPE html>`
2. `<html>`
3. `<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>`
4. `<body>`
5. `<div ng-app="myApp" ng-controller="myCtrl">`
6. `{{ firstName + " " + lastName }}`
7. `</div>`
8. `<script>`
9. `var app = angular.module("myApp", []);`
10. `app.controller("myCtrl", function($scope) {`
11. `$scope.firstName = "Ajeet";`
12. `$scope.lastName = "Maurya";`
13. `});`
14. `</script>`
15. `</body>`
16. `</html>`

How to add directive to a module

AngularJS directives are used to add functionality to your application. You can also add your own directives for your applications.

Following is a list of AngularJS directives:

Directive	Description
ng-app	It defines the root element of an application.
ng-bind	It binds the content of an html element to application data.
ng-bind-html	It binds the innerhtml of an html element to application data, and also removes dangerous code from the html string.
ng-bind-template	It specifies that the text content should be replaced with a template.
ng-blur	It specifies a behavior on blur events.
ng-change	It specifies an expression to evaluate when content is being changed by the user.
ng-checked	It specifies if an element is checked or not.
ng-class	It specifies css classes on html elements.
ng-class-even	It is same as ng-class, but will only take effect on even rows.
ng-class-odd	It is same as ng-class, but will only take effect on odd rows.
ng-click	It specifies an expression to evaluate when an element is being clicked.
ng-cloak	It prevents flickering when your application is being loaded.
ng-controller	It defines the controller object for an application.
ng-copy	It specifies a behavior on copy events.
ng-csp	It changes the content security policy.
ng-cut	It specifies a behavior on cut events.

ng-dblclick	It specifies a behavior on double-click events.
ng-disabled	It specifies if an element is disabled or not.
ng-focus	It specifies a behavior on focus events.
ng-form	It specifies an html form to inherit controls from.
ng-hide	It hides or shows html elements.
ng-href	It specifies a URL for the <a> element.
ng-if	It removes the html element if a condition is false.
ng-include	It includes html in an application.
ng-init	It defines initial values for an application.
ng-jq	It specifies that the application must use a library, like jQuery.
ng-keydown	It specifies a behavior on keydown events.
ng-keypress	It specifies a behavior on keypress events.
ng-keyup	It specifies a behavior on keyup events.
ng-list	It converts text into a list (array).
ng-model	It binds the value of html controls to application data.
ng-model-options	It specifies how updates in the model are done.
ng-mousedown	It specifies a behavior on mousedown events.
ng-mouseenter	It specifies a behavior on mouseenter events.
ng-mouseleave	It specifies a behavior on mouseleave events.
ng-mousemove	It specifies a behavior on mousemove events.
ng-mouseover	It specifies a behavior on mouseover events.
ng-mouseup	It specifies a behavior on mouseup events.

ng-non-bindable	It specifies that no data binding can happen in this element, or it's children.
ng-open	It specifies the open attribute of an element.
ng-options	It specifies <options> in a <select> list.
ng-paste	It specifies a behavior on paste events.
ng-pluralize	It specifies a message to display according to en-us localization rules.
ng-readonly	It specifies the readonly attribute of an element.
ng-repeat	It defines a template for each data in a collection.
ng-required	It specifies the required attribute of an element.
ng-selected	It specifies the selected attribute of an element.
ng-show	It shows or hides html elements.
ng-src	It specifies the src attribute for the element.
ng-srcset	It specifies the srcset attribute for the element.
ng-style	It specifies the style attribute for an element.
ng-submit	It specifies expressions to run on onsubmit events.
ng-switch	It specifies a condition that will be used to show/hide child elements.
ng-transclude	It specifies a point to insert transcluded elements.
ng-value	It specifies the value of an input element.

How to add directives

See this example:

1. <!DOCTYPE html>
2. <html>

```
3. <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"
   > </script>
4. <body>
5.
6. <div ng-app="myApp" w3-test-directive> </div>
7. <script>
8. var app = angular.module("myApp", []);
9. app.directive("w3TestDirective", function() {
10.   return {
11.     template : "This is a directive constructor. "
12.   };
13. });
14. </script>
15. </body>
16. </html>
```

Modules and controllers in file

In AngularJS applications, you can put the module and the controllers in JavaScript files.

In this example, "myApp.js" contains an application module definition, while "myCtrl.js" contains the controller:

See this example:

```
1. <!DOCTYPE html>
2. <html>
3. <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"
   > </script>
4. <body>
5. <script src="myApp.js"> </script>
6. <script src="myCtrl.js"> </script>
7. </body>
8. </html>
```

Here "myApp.js" contains:

1. app.controller("myCtrl", function(\$scope) {
2. \$scope.firstName = "Ajeet";
3. \$scope.lastName = "Maurya";
4. });

Here "myCtrl.js" contains:

1. <div ng-app="myApp" ng-controller="myCtrl">
2. {{ firstName + " " + lastName }}
3. </div>

This example can also be written as:

1. <!DOCTYPE html>
2. <html>
3. <body>
4. <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
5. <div ng-app="myApp" ng-controller="myCtrl">
6. {{ firstName + " " + lastName }}
7. </div>
8. <script>
9. var app = angular.module("myApp", []);
10. app.controller("myCtrl", function(\$scope) {
11. \$scope.firstName = "Ajeet";
12. \$scope.lastName = "Maurya";
13. });
14. </script>
15. </body>
16. </html>

Angular Routing [10 marks]

Routing helps in directing users to different pages based on the option they choose on the main page. Hence, based on the option they choose, the required Angular Component will be rendered to the user.

Let's see the necessary steps to see how we can implement routing in an Angular 2 application.

Step 1 – Add the base reference tag in the index.html file.

```
<!DOCTYPE html>
<html>
  <head>
    <base href = "/">
    <title>Angular QuickStart</title>
    <meta charset = "UTF-8">
    <meta name = "viewport" content = "width = device-width, initial-scale = 1">

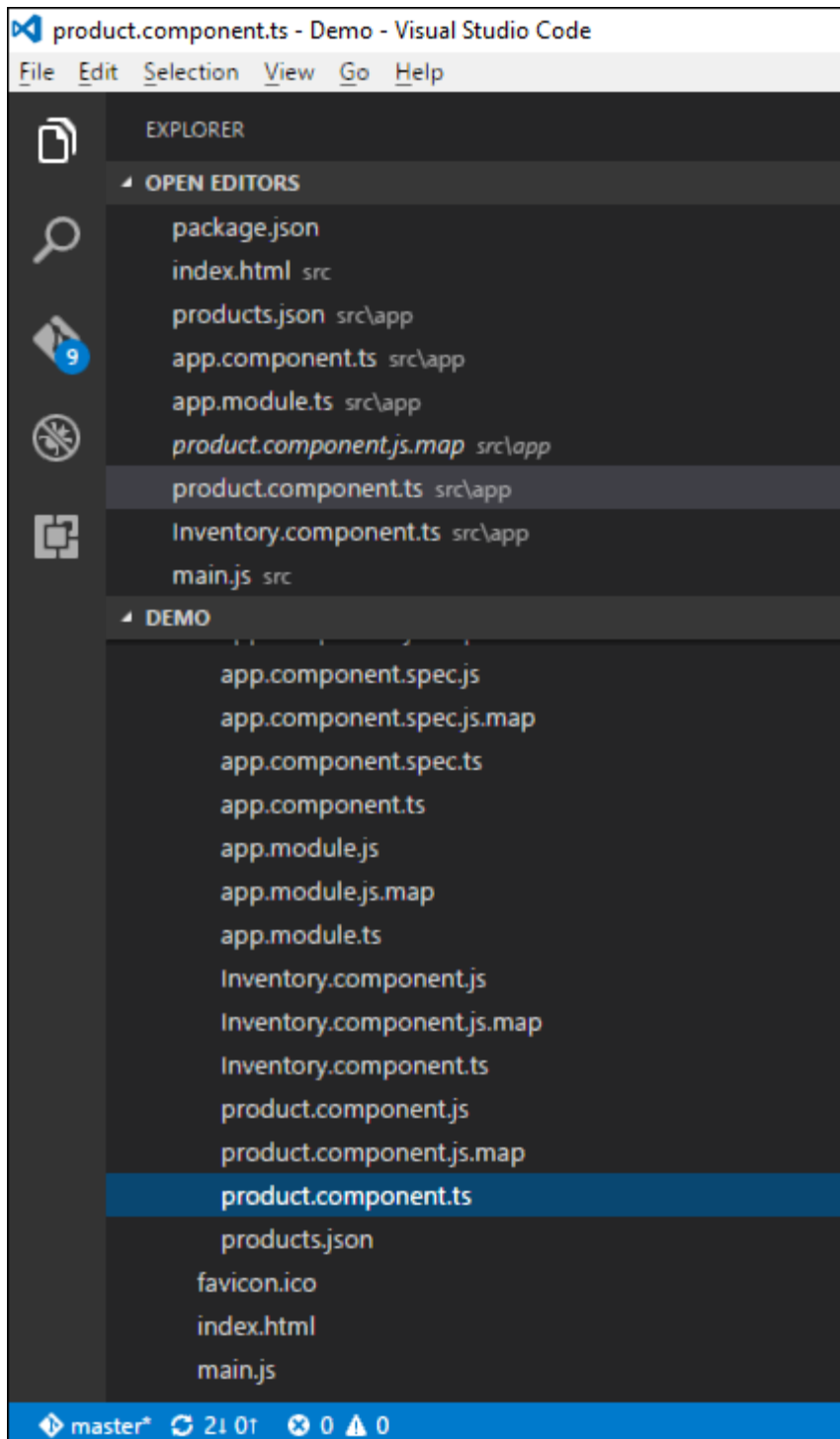
    <base href = "/">
    <link rel = "stylesheet" href = "styles.css">

    <!-- Polyfill(s) for older browsers -->
    <script src = "node_modules/core-js/client/shim.min.js"></script>
    <script src = "node_modules/zone.js/dist/zone.js"></script>
    <script src = "node_modules/systemjs/dist/system.src.js"></script>
    <script src = "systemjs.config.js"></script>

    <script>
      System.import('main.js').catch(function(err){ console.error(err); });
    </script>
  </head>

  <body>
    <my-app></my-app>
  </body>
</html>
```

Step 2 – Create two routes for the application. For this, create 2 files called **Inventory.component.ts** and **product.component.ts**



Step 3 – Place the following code in the product.component.ts file.

```
import { Component } from '@angular/core';

@Component ({
  selector: 'my-app',
```

```
    template: 'Products',
  })
  export class Appproduct {
  }
```

Step 4 – Place the following code in the Inventory.component.ts file.

```
import { Component } from '@angular/core';

@Component({
  selector: 'my-app',
  template: 'Inventory',
})
export class AppInventory {
}
```

Both of the components don't do anything fancy, they just render the keywords based on the component. So for the Inventory component, it will display the Inventory keyword to the user. And for the products component, it will display the product keyword to the user.

Step 5 – In the app.module.ts file, add the following code –

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import { Appproduct } from './product.component';
import { AppInventory } from './Inventory.component';
import { RouterModule, Routes } from '@angular/router';

const appRoutes: Routes = [
  { path: 'Product', component: Appproduct },
  { path: 'Inventory', component: AppInventory },
];

@NgModule({
  imports: [ BrowserModule,
    RouterModule.forRoot(appRoutes)],
  declarations: [ AppComponent,Appproduct,AppInventory],
  bootstrap: [ AppComponent ]
})
export class AppModule {}
```

The following points need to be noted about the above program –

- The appRoutes contain 2 routes, one is the Appproduct component and the other is the AppInventory component.
- Ensure to declare both of the components.

- The RouterModule.forRoot ensures to add the routes to the application.

Step 6 – In the app.component.ts file, add the following code.

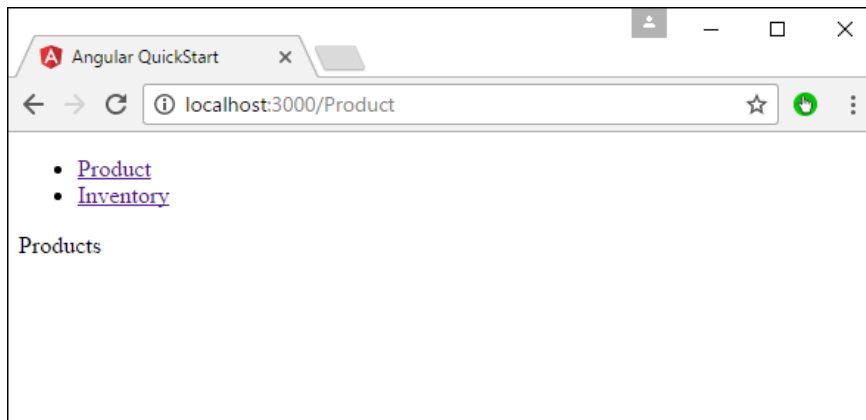
```
import { Component } from '@angular/core';

@Component({
  selector: 'my-app',
  template: `
    <ul>
      <li><a [routerLink] = "['/Product']">Product</a></li>
      <li><a [routerLink] = "['/Inventory']">Inventory</a></li>
    </ul>
    <router-outlet></router-outlet>`
})
export class AppComponent {}
```

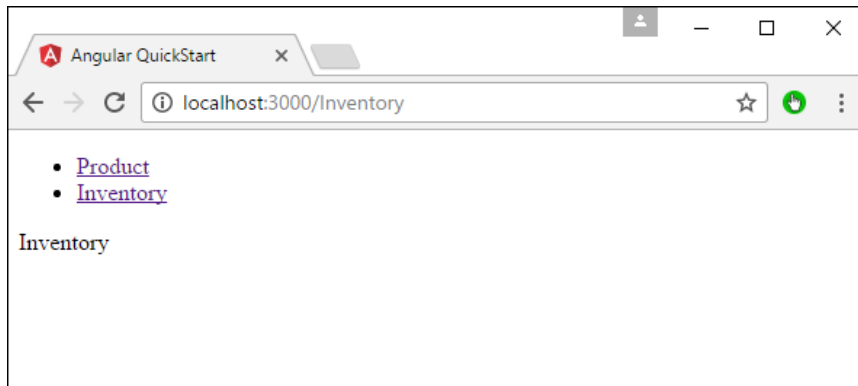
The following point needs to be noted about the above program –

- <router-outlet></router-outlet> is the placeholder to render the component based on which option the user chooses.

Now, save all the code and run the application using npm. Go to the browser, you will see the following output.



Now if you click the Inventory link, you will get the following output.



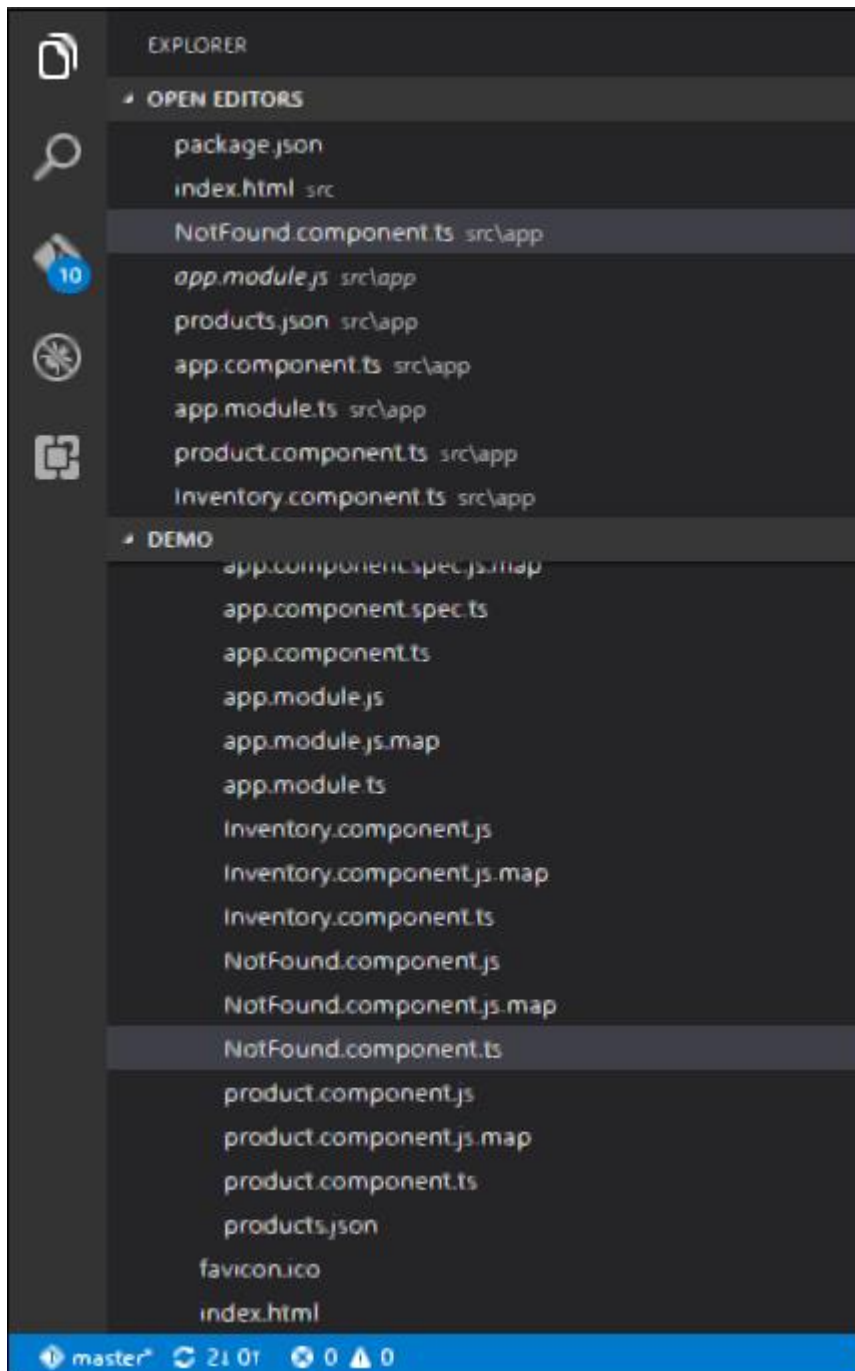
Adding an Error Route

In Routing, one can also add an error route. This can happen if the user goes to a page which does not exist in the application.

Let's see how we can go about implementing this.

Step 1 – Add a PageNotFound component as `NotFound.component.ts` as shown below

–



Step 2 – Add the following code to the new file.

```
import { Component } from '@angular/core';

@Component({
  selector: 'my-app',
  template: 'Not Found',
})
export class PageNotFoundComponent {
```

```
}
```

Step 3 – Add the following code to the app.module.ts file.

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import { Appproduct } from './product.component'
import { AppInventory } from './Inventory.component'
import { PageNotFoundComponent } from './NotFound.component'
import { RouterModule, Routes } from '@angular/router';

const appRoutes: Routes = [
  { path: 'Product', component: Appproduct },
  { path: 'Inventory', component: AppInventory },
  { path: '**', component: PageNotFoundComponent }
];

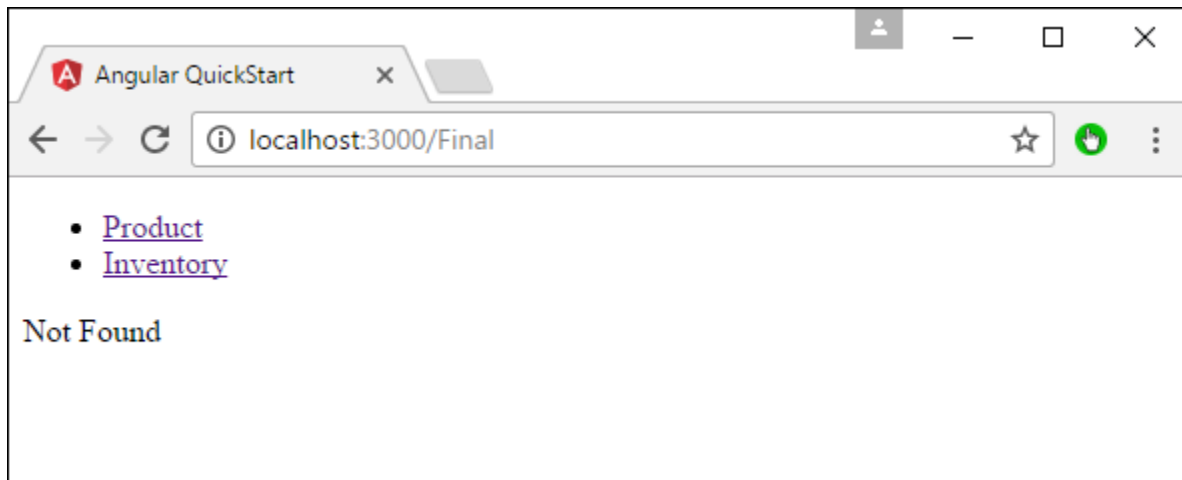
@NgModule ({
  imports: [ BrowserModule,
    RouterModule.forRoot(appRoutes)],
  declarations: [ AppComponent,Appproduct,AppInventory,PageNotFoundComponent],
  bootstrap: [ AppComponent ]
})

export class AppModule {
}
```

The following point needs to be noted about the above program –

- Now we have an extra route called path: '**', component: PageNotFoundComponent. Hence, ** is for any route which does not fit the default route. They will be directed to the PageNotFoundComponent component.

Now, save all the code and run the application using npm. Go to your browser, and you will see the following output. Now, when you go to any wrong link you will get the following output.



Observable in Angular[5 Marks]

The Observables in Angular, a popular framework and a platform in **Javascript** using which you can build tremendous single-page client-side applications using the bootlegs of [Typescript](#) and [HTML](#). It is primarily coded in Typescript, a superset of Javascript that lets you import core and optional features in your apps. Thus, before going any further on the implementation of Observable in Angular, let's just dive into understanding what observables are.

Observable in Angular is a feature that provides support for delivering messages between different parts of your single-page application. This feature is frequently used in Angular because it is responsible for handling **multiple values, asynchronous programming** in [Javascript](#), and also event handling processes.

However, an observer or observable is a software paradigm constituting the design of an object which is called a subject that maintains a list of various dependencies that are called observers. These dependencies are automatically notified whenever you try to change states.

Nature of Observable

Observables in Angular are generally declarative i.e., if you ought to define a function for value publication. It won't be executed until and unless it is subscribed. The subscriber is termed as a **consumer** who receives the notifications until the function completes itself or until they manually unsubscribe.

An observable can be used to deliver any type of multiple values be it literal events, messages depending upon the context provided. The receiving and delivering values are synchronously or asynchronously the same. Since the setup and breakdown are handled by the observable, you don't need to worry about your application code being subscribed to consumer values or unsubscribed. Be it an HTTP response or a timer of intervals, the interface for listening to events and stopping them will be the same. This adds an advantage of making use of observables frequently for development purposes.

Usage

The basic usage of Observable in Angular is to create an instance to define a **subscriber function**. Whenever a consumer wants to execute the function the **subscribe()** method is called. This function defines how to obtain messages and values to be published.

To make use of the observable, all you need to do is to begin by creating notifications using subscribe() method, and this is done by passing observer as discussed previously. The notifications are generally Javascript objects that handle all the received notifications. Also, the unsubscribe() method comes along with subscribing () method so that you can stop receiving notifications at any point in time.

Consider the below example constituting geolocation updates whenever a subscriber subscribes.

Creating an observable

1. `const locations = new Observable((observer) =>`
2. `{`
3. `let watchId: number;`

Making API integration to check values

1. `if ('geolocation' in navigator) {`
2. `watchId = navigator.geolocation.watchPosition((position: Position) => {`
3. `observer.next(position);`
4. `}, (error: PositionError) => {`
5. `observer.error(error);`
6. `});`
7. `} else {`
8. `observer.error('Geolocation not available'); }`

Data is ready for cleanup for next subscription using the below command.

```

1. return {
2.   unsubscribe() {
3.     navigator.geolocation.clearWatch(watchId);
4.   }
5. };
6. });

```

Now, use `subscribe()` function for getting continuous updates using the below code:

```

1. const locationsSubscription = locations.subscribe({
2.   next(position) {
3.     console.log('Current Position: ', position);
4.   },
5.   error(msg) {
6.     console.log('Error Getting Location: ', msg);
7.   }
8. });

```

Set the time intervals for 5 seconds which the notifications will be stopped automatically through the help of API integration. Consider the below code for reference.

```

1. setTimeout(() =>
2. {
3.   locationsSubscription.unsubscribe();
4. },
5. 5000);

```

Types of Notifications and Description

1. **Next:** It is called after the execution starts for zero times or more than that. It is a mandatory notification for catching each value delivered.
2. **Error:** This is an optional handler for chasing error notifications. This notification halts execution while running instances of observable.
3. **Complete:** It is an optional handler that notifies completion of executions. The currently delayed values continue to be delivered even when the execution is marked as complete

In this , we learned the Observables from scratch and saw through an example how a `subscribe()` and `unsubscribe()` method works in a real-life geo-location application. We also came across various methods constituting the API integration techniques through which the time interval can be defined. Moreover, we came across various types of notifications along with their description which is specifically used whenever the subscriber subscribes or unsubscribes with the help of Observable.