

## Q1. Create Database as SQL practice and use it for further questions.

- Creating a database name `Sql_Class_2_Assignment` and will use it for further questions.

### Query:

```
CREATE DATABASE Sql_Class_2_Assignment;
```

### Execution:

The screenshot shows a code editor interface with a dark theme. On the left is the Explorer sidebar, which contains a tree view of database structures. A red box highlights the folder 'sql\_class\_2\_assignment' under the 'DATA: DATABASE' section. The main workspace shows a query editor tab titled 'Sql\_Class\_2\_Assignment.sql' containing the SQL command 'CREATE DATABASE Sql\_Class\_2\_Assignment;'. This query is highlighted with a red box. Below the query, the 'Result' tab displays the output: 'CREATE DATABASE Sql\_Class\_2\_Assignment' and 'AffectedRows : 1'. The status bar at the bottom indicates a cost of 20ms for the execution.

```
USE Sql_Class_2_Assignment;
```

The screenshot shows the Visual Studio Code interface with the following details:

- Explorer View:** Shows a file tree with a red box around the "sakila" database under "growdataskills".
- SQL Editor:** A code editor window titled "Sql\_Class\_2\_Assignment.sql" containing the following SQL script:
 

```

1 -- Q1. Create Database as SQL practice and use it for further ques
2 CREATE DATABASE Sql_Class_2_Assignment;
3
4 -- Use this database
5 USE Sql_Class_2_Assignment; 2ms
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
      
```
- Output Panel:** A "Result" tab showing the execution of the "USE" statement:
 

```

USE Sql_Class_2_Assignment
AffectedRows : 0
      
```
- Right-hand Side:** Includes a "Preview README.md" panel, a "Query:" section with the same SQL, and an "Execution:" section showing the result in a database viewer.

**Q2.** Create a table named "Students" with the following columns: StudentID (int), FirstName (varchar), LastName (varchar), and Age (int). Insert at least three records into the table.

- Creating a table named **Students** with the following columns: StudentID (int), FirstName (varchar), LastName (varchar), and Age (int)

#### Query:

```

CREATE TABLE Students(
    StudentID INT,
    FirstName VARCHAR(60),
    LastName VARCHAR(60),
    Age INT
);
      
```

#### Execution:

The screenshot shows the Visual Studio Code interface with several tabs open:

- Sql\_Class\_2\_Assignment.sql**: Contains the following SQL code:
 

```

1 -- Q1. Create Database as SQL practice and use it for further questions.
2 CREATE DATABASE Sql_Class_2_Assignment;
3
4 -- Use this database
5 USE Sql_Class_2_Assignment;
6
7 /*
8 Q2. Create a table named "Students" with the following columns:
9 StudentID (int), FirstName (varchar), LastName (varchar),
10 and Age (int).
11 Insert at least three records into the table.
12 */
13
14 --> Execute | JSON | Copy
15 CREATE TABLE Students(
16   StudentID INT,
17   FirstName VARCHAR(60),
18   LastName VARCHAR(60),
19   Age INT
20 );
      
```
- README.md**: Contains the following text:
 

Q2. Create a table named "Students" with the following columns:  
StudentID (int), FirstName (varchar), LastName (varchar),  
and Age (int).  
Insert at least three records into the table.
- Result**: Shows the output of the query:
 

```

CREATE TABLE Students( StudentID INT, FirstName VARCHAR(60), LastName VARCHAR(60), Age INT )
AffectedRows : 0
      
```
- students**: A table viewer showing the structure of the Students table:
 

	StudentID	FirstName	LastName	Age
1	1	Arpit	Dubey	25
2	2	Shaija	Mishra	23
3	3	Spider	Man	29

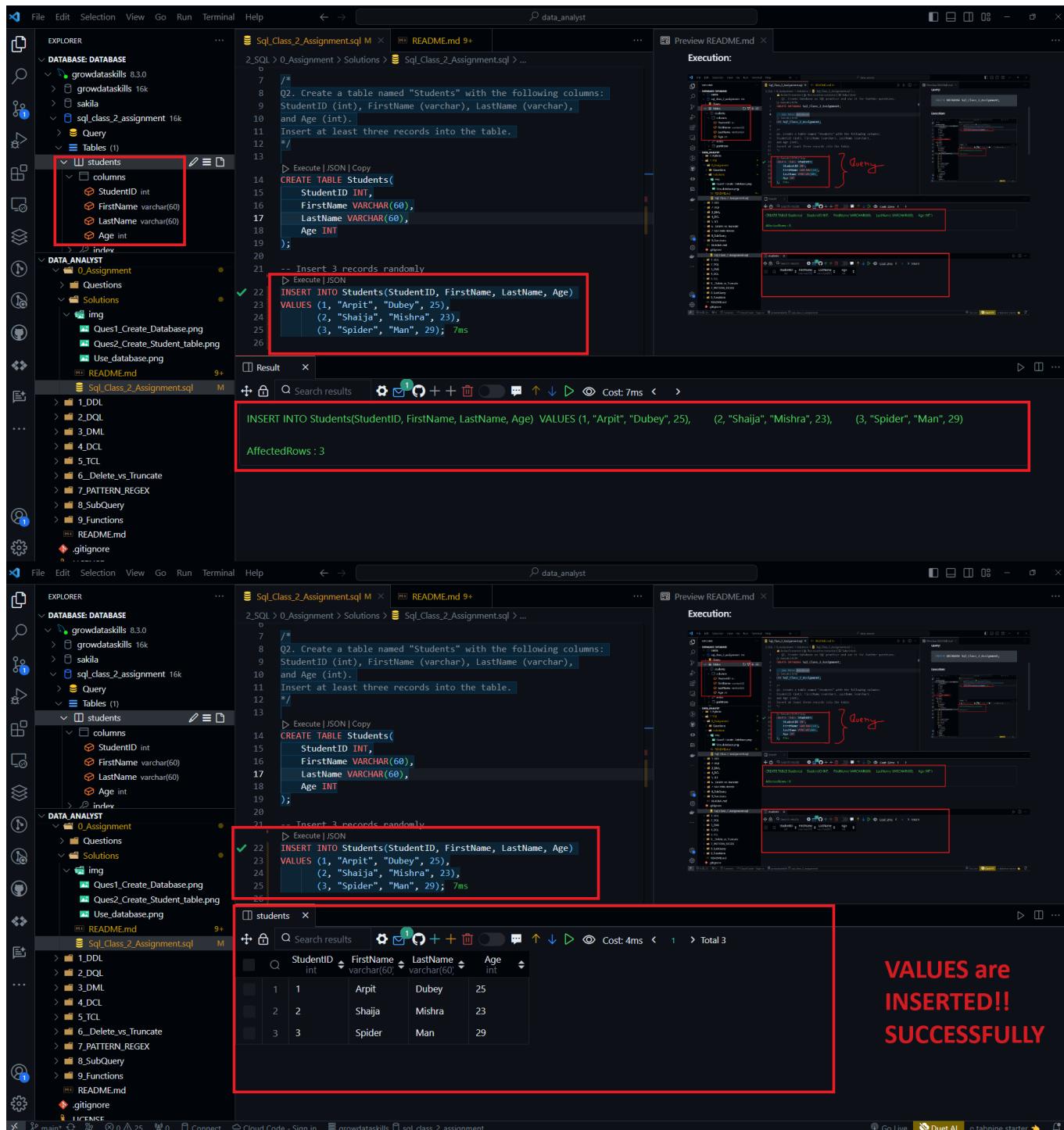
- Inserting 3 random records into the table **Students**

### Query:

```

INSERT INTO Students(StudentID, FirstName, LastName, Age)
VALUES (1, "Arpit", "Dubey", 25),
       (2, "Shaija", "Mishra", 23),
       (3, "Spider", "Man", 29);
      
```

### Execution:



The screenshot shows the Visual Studio Code interface with the following details:

- Explorer:** Shows the database structure under "DATABASE: DATABASE". A "students" table is selected, showing columns: StudentID (int), FirstName (varchar(60)), LastName (varchar(60)), and Age (int).
- Sql Class\_2\_Assignment.sql:** The code for creating the "Students" table and inserting three records is visible.
- Execution:** The "Preview README.md" tab shows the execution results of the SQL query.
- Result:** The "Result" tab displays the output of the INSERT INTO statement, showing three rows inserted successfully.

```

2_SQL > 0_Assignment > Solutions > Sql_Class_2_Assignment.sql > ...
7 /*
8 Q2. Create a table named "Students" with the following columns:
9 StudentID (int), FirstName (varchar), LastName (varchar),
10 and Age (int).
11 Insert at least three records into the table.
12 */

13
14 CREATE TABLE Students(
15     StudentID INT,
16     FirstName VARCHAR(60),
17     LastName VARCHAR(60),
18     Age INT
19 );
20
21 -- Insert 3 records randomly
22 INSERT INTO Students(StudentID, FirstName, LastName, Age)
23 VALUES (1, "Arpit", "Dubey", 25),
24 (2, "Shaija", "Mishra", 23),
25 (3, "Spider", "Man", 29); 7ms
26
  
```

A red box highlights the "Result" tab, which contains the following output:

```

INSERT INTO Students(StudentID, FirstName, LastName, Age) VALUES (1, "Arpit", "Dubey", 25), (2, "Shaija", "Mishra", 23), (3, "Spider", "Man", 29)

AffectedRows : 3
  
```

A second screenshot below shows the updated state:

**VALUES are INSERTED!! SUCCESSFULLY**

Q3. Update the age of the student with StudentID **1** to **21**. Delete the student with StudentID **3** from the "Students" table.

- **UPDATE** the column **Age** of **students** table **WHERE** the **StudentID** is equal to **1**.

**Query:**

```

UPDATE Students
SET Age = 21
WHERE StudentID = 1;
  
```

## Execution:

The screenshot shows a Visual Studio Code interface with several panes. The Explorer pane on the left lists a database named 'growdataskills' with a schema including 'sakila' and 'sql\_class\_2\_assignment'. The 'students' table is selected. The 'Sql Class 2 Assignment.sql' file in the center contains SQL code for inserting and updating student records. A red box highlights the update command:

```

32    UPDATE Students
33    SET Age = 21
34    WHERE StudentID = 1; 6ms
35
36

```

The 'Result' pane below shows the output of the update command:

```

UPDATE Students SET Age = 21 WHERE StudentID = 1
AffectedRows : 1

```

The 'Preview README.md' pane on the right shows a table with student data. A red box highlights the 'Age' column for StudentID 1, which is now 21. A message says 'VALUES are INSERTED!! SUCCESSFULLY'.

**Q3. Update the age of the student with StudentID 1 to 21. Delete the student with StudentID 3 from the "Students" table.**

- UPDATE the column Age of students table WHERE the StudentID is equal to 1.

The 'students' table in the bottom right shows the updated data. A red box highlights the 'Age' column for StudentID 1, which is now 21. A green box highlights the value 21 with the text 'Previously, it is 25'.

- **DELETE** the record of student WHERE the StudentID is 3.

## Query:

```
DELETE FROM Students
WHERE StudentID = 3;
```

## Execution:

The screenshot shows a Visual Studio Code interface with several panes:

- Explorer:** Shows a file tree with databases like `growskill 8.3.0` and `sakila`, and a `sql.class\_2\_assignment` folder containing a `students` table.
- Editor:** Displays a SQL script named `Sql\_Class\_2\_Assignment.sql` with the following content:
 

```

27 /* Q3. Update the age of the student with StudentID '1' to '21'.
28 Delete the student with StudentID '3' from the "Students" table.
29 */
30
31
32 -- UPDATE the age of the student with StudentID is equal to 1.
33 > Execute | JSON
34 UPDATE Students
35 SET Age = 21
36 WHERE StudentID = 1;
37
38 -- DELETE students record WHERE StudentID is equal to 3.
39 > Execute | JSON
40 DELETE FROM Students
41 WHERE StudentID = 3; 5ms
      
```
- Output:** Shows the execution results of the `DELETE` query:
 

```

DELETE FROM Students WHERE StudentID = 3
AffectedRows : 1
      
```
- Preview:** Shows a table named `students` with two rows:
 

	StudentID	FirstName	LastName	Age
1	1	Arpit	Dubey	21
2	2	Shaija	Mishra	23
- Bottom Status Bar:** Shows the status bar with various icons and information like "Cloud Code - Sign in", "You, 14 minutes ago", "Ln 34, Col 13", "Spaces: 4", "UTF-8", "CRLF", "SQL", "Go Live", "Duet AI", "tabnine starter", and "Prettier".

**Annotations:**

- A red box highlights the `WHERE StudentID = 3;` line in the editor.
- A red box highlights the `AffectedRows : 1` message in the output pane.
- A red box highlights the "DELETED!! RECORD" message in the preview pane.
- A callout box with a red border contains the text "Record with StudentID is equal to 3." in orange and green.
- A callout box with a red border contains the text "Deleted Record Successfully" in red and green.

## Q4. Retrieve the first names and ages of all students who are older than 20.

- Retrieving the **FirstName** and **Age** of all students without any condition.

### Query:

```

SELECT FirstName, Age
FROM Students
      
```

### Execution:

The screenshot shows the VS Code interface with the following details:

- Explorer:** Shows the database structure with the `Students` table selected.
- Sql Class\_2\_Assignment.sql:** Contains the following SQL code:
 

```

2_SQL > 0.Assignment > Solutions > Sql_Class_2_Assignment.sql > ...
28 Q3. Update the age of the student with StudentID '1' to '21'.
29 Delete the student with StudentID '3' from the "Students" table.
30 */
31
32 -- UPDATE the age of the student with StudentID is equal to 1.
33 UPDATE Students
34 SET Age = 21
35 WHERE StudentID = 1;
36
37 /* DELETE students record WHERE StudentID is equal to 3.
38 DELETE FROM Students
39 WHERE StudentID = 3;
40 */
41 /*
42 Q4. Retrieve the first names and ages of all students who are older than 20.
43 */
44
45 */
46
      
```
- Execution:** Shows the results of the query `SELECT FirstName, Age FROM Students`. The results table is highlighted with a green border and contains two rows:
 

	FirstName	Age
1	Arpit	21
2	Shaija	23
- Bottom Status Bar:** Shows the status bar with various icons and information like 'Cloud Code - Sign in', 'growthskills', 'sql\_class\_2\_assignment', 'You, 3 minutes ago', 'Ln 38, Col 21', 'Spaces: 4', 'UTF-8', 'SQL', 'Go Live', 'Duet AI', 'tabnine starter', and 'Prettier'.

- Retrieving the `FirstName` and `Age` of all students `WHERE` the `Age` is greater than `20`..

## Query:

```

SELECT FirstName, Age
FROM Students
WHERE Age > 20;
  
```

## Execution:

The screenshot shows the VS Code interface with the following details:

- Explorer:** Shows the database structure with the `Students` table selected.
- Sql Class\_2\_Assignment.sql:** Contains the following SQL code:
 

```

-- DELETE students record WHERE StudentID is equal to 3.
DELETE FROM Students
WHERE StudentID = 3;

/*
Q4. Retrieve the first names and ages of all students who are older than 20.
*/
-- Retrieving the 'FirstName' and 'Age' of all students
-- NO CONDITION
SELECT FirstName, Age
FROM Students
      
```
- Execution:** Shows the results of the query `SELECT FirstName, Age FROM Students`. The results table is highlighted with a green border and contains two rows:
 

	FirstName	Age
1	Arpit	21
2	Shaija	23
- Bottom Status Bar:** Shows the status bar with various icons and information like 'Cloud Code - Sign in', 'growthskills', 'sql\_class\_2\_assignment', 'You, 23 seconds ago', 'Ln 48, Col 14', 'Spaces: 4', 'UTF-8', 'SQL', 'Go Live', 'Duet AI', 'tabnine starter', and 'Prettier'.

A handwritten note in yellow ink on the right side of the screen says: "after applying the Condition".

**NOTE : To Perform more operation we required more values so I just inserted more values to the table.**

### Query:

```
INSERT INTO Students(StudentID, FirstName, LastName, Age)
VALUES (4, "Rahul", "Kumar", 12),
       (5, 'Alice', 'Smith', 25),
       (6, 'Bob', 'Johnson', 15),
       (7, 'Eva', 'Martinez', 16),
       (8, 'Hariharan', 'S', 26);
```

### Execution:

The screenshot shows a SQL development environment with the following details:

- Explorer:** Shows the database structure, including the `Students` table under `sql_class_2_assignment`.
- SQL Editor:** Contains the following code:
 

```
SELECT FirstName, Age
FROM Students
-- Retrieving the 'FirstName' and 'Age' of all students
-- CONDITION WHERE Age > 20
SELECT FirstName, Age
FROM Students
-- Inserting more values
INSERT INTO Students(StudentID, FirstName, LastName, Age)
VALUES (4, "Rahul", "Kumar", 12),
       (5, 'Alice', 'Smith', 25),
       (6, 'Bob', 'Johnson', 15),
       (7, 'Eva', 'Martinez', 16),
       (8, 'Hariharan', 'S', 26);
```
- Preview:** Shows the results of the `SELECT` query:
 

StudentID	FirstName	LastName	Age
1	Arpit	Dubey	21
2	Shaija	Mishra	23
3	Rahul	Kumar	12
4	Alice	Smith	25
5	Bob	Johnson	15
6	Eva	Martinez	16
7	Hariharan	S	26
- Execution:** Shows the results of the `INSERT` query, indicating 5 rows affected.
- Results Grid:** Shows the final state of the `Students` table with 7 rows.

A large yellow bracket on the right side of the results grid is overlaid with the handwritten text: **} Inserted more Values!!**

### Q5. Delete records from the same table where age < 18

- `DELETE` those records from the `Students` table `WHERE` the `Age` column `VALUES` are lesser than `18`

### Query:

```
DELETE FROM Students
WHERE Age < 18;
```

### Execution:

The screenshot shows the Visual Studio Code interface with several panes:

- EXPLORER** pane on the left showing a file tree with various database and script files.
- Sql\_Class\_2\_Assignment.sql** pane in the center containing the following SQL code:
 

```

SELECT FirstName, Age
FROM Students

-- Inserting more values
INSERT INTO Students(StudentID, FirstName, LastName, Age)
VALUES (4, "Rahul", "Kumar", 12),
(5, "Alice", "Smith", 25),
(6, "Bob", "Johnson", 15),
(7, "Eva", "Martinez", 16),
(8, "Hariharan", "S", 26);

-- Q5. Delete records from the same table where age < 18
DELETE FROM Students
WHERE Age < 18;
      
```
- Result** pane at the bottom showing the output of the DELETE query:
 

```

DELETE FROM Students WHERE Age < 18
AffectedRows : 3
      
```
- students** table preview pane showing the state of the Students table after the deletion:
 

	StudentID	FirstName	LastName	Age
1	1	Arpit	Dubey	21
2	2	Shaija	Mishra	23
3	5	Alice	Smith	25
4	8	Hariharan	S	26

**Query:**

```

DELETE FROM Students
WHERE Age < 18;
      
```

**Execution:**

**Successfully, removes all the records WHERE the students Age column have VALUES less than 18**

**Q6. Create a table named "Customers" with the following columns and constraints: CustomerID (int) as the primary key. FirstName (varchar) not null. LastName (varchar) not null. Email (varchar) unique. Age (int) check constraint to ensure age is greater than 18.**

- **CREATE** a **TABLE** named it as **Customers**
- With columns such as **CustomerID**, **FirstName**, **LastName**, **Email** and, **Age**.
- **Age** columns must having a **CHECK** constraint where it ensures the **Age** will not be less than 18.

### Query:

```

CREATE TABLE Customers(
    CustomerID INT PRIMARY KEY,
    FirstName VARCHAR(60) NOT NULL,
    LastName VARCHAR(60) NOT NULL,
    Email VARCHAR(80) UNIQUE,
    Age INT CHECK (Age > 18)
);
      
```

### Execution:

```

CREATE TABLE Customers(
    CustomerID INT PRIMARY KEY,
    FirstName VARCHAR(60) NOT NULL,
    LastName VARCHAR(60) NOT NULL,
    Email VARCHAR(80) UNIQUE,
    Age INT CHECK (Age > 18)
);

```

**Successfully Executed!!**

Q7. You have a table named "Orders" with columns: OrderID (int), CustomerID (int), OrderDate (date), and TotalAmount (decimal). Create a foreign key constraint on the "CustomerID" column referencing the "Customers" table.

- Create Orders Table with Foreign Key Constraint: Now, create the "Orders" table while referencing the "Customers" table by including a foreign key constraint. Here's an example SQL script to create the "Orders" table with the foreign key constraint:

#### Query:

```

CREATE TABLE Orders(
    OrderID INT,
    CustomerID INT,
    OrderDate DATE,
    TotalAmount DECIMAL(10,2),
    CONSTRAINT FK_CustomerID
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);

```

**Introduction:** Provide an overview of the requirement to create the "Orders" table and to establish a relationship with the "Customers" table.

**Step 1:** Explain the creation of the "Customers" table with columns like CustomerID and CustomerName. Include the SQL script for creating the table.

**Step 2:** Describe the creation of the "Orders" table with columns like OrderID, CustomerID, OrderDate, and TotalAmount. Clarify the addition of a foreign key constraint on the "CustomerID" column, referencing the "CustomerID" column in the "Customers" table. Include the SQL script for creating the table along with the foreign key constraint.

```
ALTER TABLE Orders
ADD CONSTRAINT FK_Customer
FOREIGN KEY (CustomerID)
REFERENCES Customers(CustomerID);
```

## Execution:

The screenshot shows a code editor interface with several panes. On the left, the Explorer pane shows a database structure with a table named 'orders' selected. The main pane contains an SQL script for creating the 'Orders' table, which includes a foreign key constraint 'FK\_Customer' referencing the 'CustomerID' column in the 'Customers' table. The 'Execution' pane on the right shows the results of the query execution, including the successful creation of the table and its constraints. A red box highlights the SQL code for the table creation, and another red box highlights the 'Successfully Executed!!' message in the execution results.

```
CREATE TABLE Orders(
    OrderID INT,
    CustomerID INT,
    OrderDate DATE,
    TotalAmount DECIMAL(10,2),
    CONSTRAINT FK_Customer
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
```

**Q8.** Create a table named "Employees" with columns: EmployeeID (int) as the primary key. FirstName (varchar) not null. LastName (varchar) not null. Salary (decimal) check constraint to ensure salary is between 20000 and 100000.

- **CREATE TABLE Employees:** This line initiates the creation of the "Employees" table.
- **(EmployeeID int PRIMARY KEY):** Defines the "EmployeeID" column as an integer primary key.
- **(FirstName varchar(80) NOT NULL):** Specifies the "FirstName" column as a non-null varchar datatype.
- **(LastName varchar(80) NOT NULL):** Specifies the "LastName" column as a non-null varchar datatype.

- **Salary DECIMAL(10,2) CHECK (Salary BETWEEN 20000 and 100000):** SET the "Salary" column as a decimal type with a precision of 10 digits and 2 decimal places and adds a check constraint to ensure that the salary is between 20000 and 100000.

## Query:

```
CREATE TABLE Employees(
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(80) NOT NULL,
    LastName VARCHAR(80) NOT NULL,
    Salary DECIMAL(10,2) CHECK (Salary BETWEEN 20000 and 100000)
);
```

## Execution:

The screenshot shows a SQL development environment with the following details:

- Explorer:** Shows the database structure with a table named "employees" selected.
- Code Editor:** Displays the SQL code for creating the "Employees" table. The entire code block is highlighted with a red box.
- Output:** Shows the executed query and its results. The output is highlighted with a red box. It includes the table definition and an "AffectedRows : 0" message.
- Table View:** Shows the "employees" table structure with columns: EmployeeID (int), FirstName (varchar(80)), LastName (varchar(80)), and Salary (decimal(10,2)). The table view is also highlighted with a red box.

**Q9. Create a table named "Books" with columns: BookID (int) as the primary key. Title (varchar) not null. ISBN (varchar) unique.**

- **CREATE TABLE Books:** Initiates the creation of a table named "Books" in the database.
- **(BookID INT PRIMARY KEY):** Defines the "BookID" column as an integer primary key, ensuring its uniqueness and serving as the identifier for each book.

- (**Title VARCHAR(80) NOT NULL**): Specifies the "Title" column as a non-null varchar data type, allowing up to 80 characters for the book title.
- (**ISBN VARCHAR(80) UNIQUE**): Defines the "ISBN" column as a varchar data type with a maximum length of 80 characters, and adds a constraint to ensure that each ISBN value is unique.

### Query:

```
CREATE TABLE Books(
    BookID INT PRIMARY KEY,
    Title VARCHAR(80) NOT NULL,
    ISBN VARCHAR(80) UNIQUE
);
```

### Execution:

The screenshot shows a SQL development environment with the following details:

- Left Panel (EXPLORER):** Shows the database structure. A folder named "books" is expanded, revealing "columns" (BookID int, Title varchar(80), ISBN varchar(80)), "index", "partitions", and "customers".
- Middle Panel (SQL Editor):** Displays the SQL code for creating the Books table. The entire code block is highlighted with a red box.
- Right Panel (Results):** Shows the execution results. The output includes the created table definition: `CREATE TABLE Books( BookID INT PRIMARY KEY, Title VARCHAR(80) NOT NULL, ISBN VARCHAR(80) UNIQUE )`. Below it, the message `AffectedRows : 0` is displayed.
- Bottom Status Bar:** Shows connection information, including the database name `sql_class_2_assignment`.

Q10. Consider a table named "Employees" with columns: EmployeeID, FirstName, LastName, and Age. Write an SQL query to retrieve the first name and last name of employees who are older than 30

- Adding Age column to pre-exist Employees TABLE

### Query:

```
ALTER TABLE Employees
ADD Age INT;
```

## Execution:

```
ALTER TABLE Employees
ADD Age INT;
```

- Inserting some random values to Employees TABLE

## Query:

```
INSERT INTO Employees (EmployeeID, FirstName, LastName, Salary, Age)
VALUES
(1, 'Shivcharan', 'Das', 45000.00, 32),
(2, 'Krish', 'Naik', 60000.00, 35),
(3, 'Rohit', 'Kapoor', 75000.00, 42),
(4, 'Ekta', 'Dubey', 95000.00, 36),
(5, 'Shailja', 'Mishra', 80000.00, 23),
(6, 'Hariharan', 'S', 65000.00, 25),
(7, 'Aman', 'Gupta', 90000.00, 45),
(8, 'Arpit', 'Dubey', 90800.00, 26),
(9, 'Shreya', 'Richharia', 42000.00, 27),
(10, 'Sudhanshu', 'Kumar', 30000.00, 40);
```

## Execution:

```

ALTER TABLE Employees
ADD Age INT;

-- Inserting some random values to Employees TABLE
INSERT INTO Employees (EmployeeID, FirstName, LastName, Salary, Age)
VALUES
(1, 'Shivcharan', 'Das', 45000.00, 32),
(2, 'Krish', 'Naik', 60000.00, 35),
(3, 'Rohit', 'Kapoor', 75000.00, 42),
(4, 'Ekta', 'Dubey', 95000.00, 36),
(5, 'Shailja', 'Mishra', 80000.00, 23),
(6, 'Hariharan', 'S', 65000.00, 25),
(7, 'Aman', 'Gupta', 90000.00, 45),
(8, 'Arpit', 'Dubey', 90800.00, 26),
(9, 'Shreya', 'Richharia', 42000.00, 27),
(10, 'Sudhanshu', 'Kumar', 30000.00, 40);
  
```

AffectedRows : 10

EmployeeID	FirstName	LastName	Salary	Age
1	Shivcharan	Das	45000.00	32
2	Krish	Naik	60000.00	35
3	Rohit	Kapoor	75000.00	42
4	Ekta	Dubey	95000.00	36
5	Shailja	Mishra	80000.00	23
6	Hariharan	S	65000.00	25
7	Aman	Gupta	90000.00	45
8	Arpit	Dubey	90800.00	26
9	Shreya	Richharia	42000.00	27
10	Sudhanshu	Kumar	30000.00	40

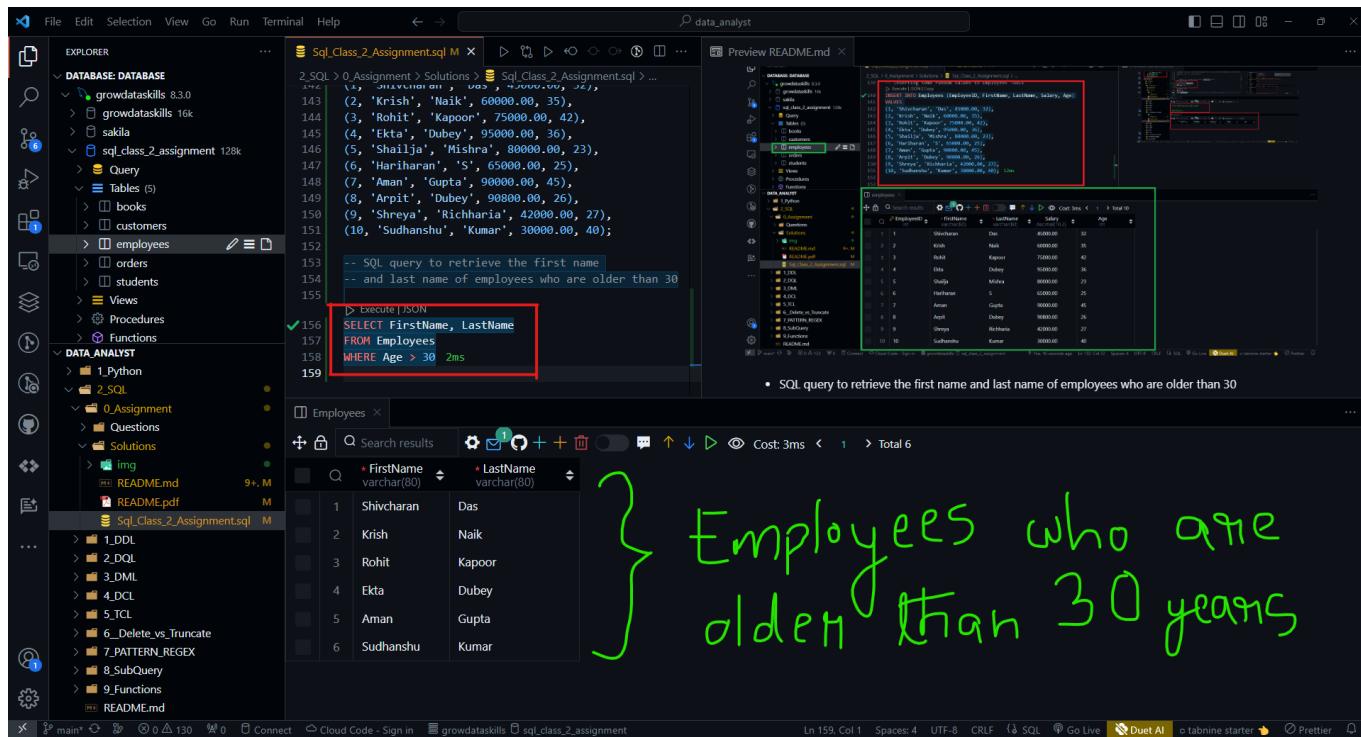
- SQL query to retrieve the first name and last name of employees who are older than 30

### Query:

```

SELECT FirstName, LastName
FROM Employees
WHERE Age > 30
  
```

### Execution:



The screenshot shows a code editor interface with several windows open. On the left, the Explorer pane shows a database named 'growingdataskills' with tables like 'employees', 'orders', 'students', and 'customers'. The main editor window contains the following SQL query:

```

2.SQL > 0.Assignment > Solutions > Sql_Class_2_Assignment.sql M
-- SQL query to retrieve the first name
-- and last name of employees who are older than 30
156 SELECT FirstName, LastName
157 FROM Employees
158 WHERE Age > 30 2ms
159
  
```

The results of this query are displayed in a table below:

	FirstName	LastName
1	Shivcharan	Das
2	Krish	Naik
3	Rohit	Kapoor
4	Ekta	Dubey
5	Aman	Gupta
6	Sudhanshu	Kumar

On the right side of the interface, there is a preview window showing the results of the query. A large green curly brace is drawn over the table and the preview window, grouping them together. Handwritten text next to the brace reads: "Employees who are older than 30 years".

Q11. Using the same "Employees" table, write an SQL query to retrieve the first name, last name, and age of employees whose age is between 20 and 30.

**Query:**

```

SELECT FirstName, LastName, Age
FROM Employees
WHERE Age BETWEEN 20 and 30;
  
```

**Execution:**

```

154 -- SQL query to retrieve the first name
155 -- and last name of employees who are older than 30
156 SELECT FirstName, LastName
157 FROM Employees
158 WHERE Age > 30
159 /*
160 Q11. Using the same "Employees" table, write an SQL query to retrieve the f
161 /*
162 /*
163 /*
164 */
165 /*
166 */
167 WHERE Age BETWEEN 20 AND 30; 2ms
  
```

Execution:

```

SELECT FirstName, LastName, Age
FROM Employees
WHERE Age BETWEEN 20 AND 30;
  
```

Age b/w 20 and 30

**Q12.** Given a table named "Products" with columns: ProductID, ProductName, Price, and InStock (0- for out of stock, 1- for in stock). Write an SQL query to retrieve the product names and prices of products that are either priced above \$100 or are out of stock

- Create a table named "Products"

#### Query:

```

CREATE TABLE Products(
    ProductID INT,
    ProductName VARCHAR(80),
    Price DECIMAL(10, 2),
    InStock INT
);
  
```

#### Execution:

The screenshot shows the Visual Studio Code interface with several panes:

- Explorer:** Shows the project structure with a red box around the 'products' folder.
- Sql Class 2 Assignment.sql:** A code editor pane containing SQL code. A red box highlights the table creation statement:
 

```
CREATE TABLE Products(
    ProductID INT,
    ProductName VARCHAR(80),
    Price DECIMAL(10, 2),
    InStock INT
);
```
- Preview README.md:** A preview pane showing the content of the README.md file, which includes a question about retrieving product names and prices.
- Result:** A results pane showing the output of the table creation command:
 

```
CREATE TABLE Products( ProductID INT, ProductName VARCHAR(80), Price DECIMAL(10, 2), InStock INT )
AffectedRows : 0
```
- products:** A data viewer pane showing the 'products' table with 10 rows of data. A red box highlights the table header.

- Inserting some random values to the table

### Query:

```
INSERT INTO Products (ProductID, ProductName, Price, InStock)
VALUES
(1, 'Widget A', 120.00, 1),
(2, 'Gizmo X', 80.50, 0),
(3, 'Thingamajig', 150.00, 1),
(4, 'Doohickey', 90.00, 1),
(5, 'Whatchamacallit', 110.00, 0),
(6, 'Widget B', 95.00, 1),
(7, 'Gadget Y', 200.00, 0),
(8, 'Contraption Z', 180.00, 1),
(9, 'Doodad', 75.00, 0),
(10, 'Device Q', 105.00, 1);
```

### Execution:

The screenshot shows two instances of the Visual Studio Code interface. Both instances have the same setup: the left pane shows a file tree with a 'products' folder selected (highlighted with a red box). The right pane contains a code editor and a terminal.

**Top Instance (Terminal View):**

- Code Editor:** Shows the following SQL code:
 

```
184 -- Inserting some random values
185 INSERT INTO Products (ProductID, ProductName, Price, InStock)
VALUES
(1, 'Widget A', 120.00, 1),
(2, 'Gizmo X', 80.50, 0),
(3, 'Thingamajig', 150.00, 1),
(4, 'Doohickey', 90.00, 1),
(5, 'Whatchamacallit', 110.00, 0),
(6, 'Widget B', 95.00, 1),
(7, 'Gadget Y', 200.00, 0),
(8, 'Contraption Z', 180.00, 1),
(9, 'Doodad', 75.00, 0),
(10, 'Device Q', 105.00, 1); 6ms
```
- Terminal:** Shows the output of the query:
 

```
Execution:
INSERT INTO Products (ProductID, ProductName, Price, InStock)
VALUES
(1, 'Widget A', 120.00, 1),
(2, 'Gizmo X', 80.50, 0),
(3, 'Thingamajig', 150.00, 1),
(4, 'Doohickey', 90.00, 1),
(5, 'Whatchamacallit', 110.00, 0),
(6, 'Widget B', 95.00, 1),
(7, 'Gadget Y', 200.00, 0),
(8, 'Contraption Z', 180.00, 1),
(9, 'Doodad', 75.00, 0),
(10, 'Device Q', 105.00, 1)

AffectedRows : 10
```

**Bottom Instance (Table View):**

- Code Editor:** Shows the same SQL code as the top instance.
- Terminal:** Shows the output of the query, which is displayed as a table in the terminal window:
 

ProductID	ProductName	Price	InStock
1	Widget A	120.00	1
2	Gizmo X	80.50	0
3	Thingamajig	150.00	1
4	Doohickey	90.00	1
5	Whatchamacallit	110.00	0
6	Widget B	95.00	1
7	Gadget Y	200.00	0
8	Contraption Z	180.00	1
9	Doodad	75.00	0
10	Device Q	105.00	1

- SQL query to retrieve the product names and prices of products that are either priced above \$100 or are out of stock

## Query:

```
SELECT ProductName, Price
FROM Products
WHERE Price > 100 OR InStock = 0;
```

## Execution:

The screenshot shows the Visual Studio Code interface with several tabs open:

- EXPLORER**: Shows the file structure of the project, including databases like `growdataskills` and tables like `products`.
- Sql\_Class\_2\_Assignment.sql**: The active code editor tab containing an SQL query.
- README.md**: Another code editor tab.
- Preview README.md**: A preview pane showing the results of the query.

The SQL query in the editor is:

```

2_SQL > 0.Assignment > Solutions > Sql_Class_2_Assignment.sql ...
191 (5, 'Whatchamacallit', 110.00, 0),
192 (6, 'Widget B', 95.00, 1),
193 (7, 'Gadget Y', 200.00, 0),
194 (8, 'Contraption Z', 180.00, 1),
195 (9, 'Doodad', 75.00, 0),
196 (10, 'Device Q', 105.00, 1);

-- SQL query to retrieve the product names and prices of products
-- that are either priced above $100 or are out of stock
200 SELECT ProductName, Price
FROM Products
WHERE Price > 100 OR InStock = 0; 2ms
201
202
203
  
```

The preview pane shows the results of the query:

	ProductName	Price
1	Widget A	120.00
2	Gizmo X	80.50
3	Thingamajig	150.00
4	Whatchamacallit	110.00
5	Gadget Y	200.00
6	Contraption Z	180.00
7	Doodad	75.00
8	Device Q	105.00

Handwritten annotations on the preview pane highlight specific rows:

- A red box surrounds the row for "Gizmo X" with the handwritten note "out of stock".
- A green box surrounds the row for "Doodad" with the handwritten note "Price is greater than \$100".

Q13. Using the "Products" table, write an SQL query to retrieve the product names and prices of products that are in stock and priced between 50 and 150.

#### Query:

```

SELECT ProductName, Price
FROM Products
WHERE InStock = 1 AND Price BETWEEN 50 AND 150;
  
```

#### Execution:

```

SELECT ProductName, Price
FROM Products
WHERE InStock = 1 AND Price BETWEEN 50 AND 150;
  
```

	ProductName	Price
1	Widget A	120.00
2	Thingamajig	150.00
3	Doochickey	90.00
4	Widget B	95.00
5	Device Q	105.00

O/P results  
Products → in stocks  
Prices are b/w 50 & 150

Q14. Consider a table named "Orders" with columns: OrderID, OrderDate, TotalAmount, and CustomerID. Write an SQL query to retrieve the order IDs and total amounts of orders placed by customer ID 1001 after January 1, 2023, or orders with a total amount exceeding \$500

- Our **Customers** table is empty, so we have to put some values in that because our **Orders** table have a Foreign Key referencing to **Customers** table.

#### Query:

```

INSERT INTO Customers (CustomerID, FirstName, LastName, Email, Age)
VALUES
(1001, 'John', 'Doe', 'john.doe@example.com', 35),
(1002, 'Jane', 'Smith', 'jane.smith@example.com', 28),
(1003, 'Michael', 'Johnson', 'michael.johnson@example.com', 42),
(1004, 'Emily', 'Davis', 'emily.davis@example.com', 29),
(1005, 'Andrew', 'Wilson', 'andrew.wilson@example.com', 37),
(1006, 'Jessica', 'Brown', 'jessica.brown@example.com', 31),
(1007, 'William', 'Martinez', 'william.martinez@example.com', 45),
(1008, 'Olivia', 'Garcia', 'olivia.garcia@example.com', 26),
(1009, 'David', 'Anderson', 'david.anderson@example.com', 33),
(1010, 'Sophia', 'Lopez', 'sophia.lopez@example.com', 40);
  
```

#### Execution:

The screenshot shows the Visual Studio Code interface with the following details:

- Explorer:** Shows the project structure with files like `1\_Python`, `2\_SQL`, `0\_Assignment`, `Questions`, `Solutions`, and `Sql\_Class\_2\_Assignment.sql`.
- Sql Class 2 Assignment.sql:** The active code editor window contains an SQL script for inserting data into the `Customers` table. A red box highlights the `VALUES` clause and its corresponding values.
- Result:** Below the editor, the results of the executed query are shown in a table. A red box highlights the table header and the first 10 rows of data.
- Preview README.md:** A preview pane on the right shows the generated content of the `README.md` file, which includes the inserted data and some handwritten notes about products and prices.

**SQL Script (Sql Class 2 Assignment.sql):**

```

-- Inserting VALUES for Customers table:
INSERT INTO Customers (CustomerID, FirstName, LastName, Email, Age)
VALUES
(1001, 'John', 'Doe', 'john.doe@example.com', 35),
(1002, 'Jane', 'Smith', 'jane.smith@example.com', 28),
(1003, 'Michael', 'Johnson', 'michael.johnson@example.com', 42),
(1004, 'Emily', 'Davis', 'emily.davis@example.com', 29),
(1005, 'Andrew', 'Wilson', 'andrew.wilson@example.com', 37),
(1006, 'Jessica', 'Brown', 'jessica.brown@example.com', 31),
(1007, 'William', 'Martinez', 'william.martinez@example.com', 45),
(1008, 'Olivia', 'Garcia', 'olivia.garcia@example.com', 26),
(1009, 'David', 'Anderson', 'david.anderson@example.com', 33),
(1010, 'Sophia', 'Lopez', 'sophia.lopez@example.com', 40);

```

**Result Table:**

	CustomerID	FirstName	LastName	Email	Age
1	1001	John	Doe	john.doe@example.com	35
2	1002	Jane	Smith	jane.smith@example.com	28
3	1003	Michael	Johnson	michael.johnson@example.com	42
4	1004	Emily	Davis	emily.davis@example.com	29
5	1005	Andrew	Wilson	andrew.wilson@example.com	37
6	1006	Jessica	Brown	jessica.brown@example.com	31
7	1007	William	Martinez	william.martinez@example.com	45
8	1008	Olivia	Garcia	olivia.garcia@example.com	26
9	1009	David	Anderson	david.anderson@example.com	33
10	1010	Sophia	Lopez	sophia.lopez@example.com	40

**Preview README.md Content:**

o/p results  
Products → Prices are between 100 & 150

**VALUES inserted Successfully !!**

- Now, Inserting some random values in pre-created Orders TABLE

```

(1, 1001, '2023-01-15', 450.00),
(2, 1002, '2023-02-20', 700.00),
(3, 1003, '2023-01-05', 300.00),
(4, 1001, '2023-03-10', 800.00),
(5, 1001, '2023-02-28', 550.00),
(6, 1004, '2023-01-20', 480.00),
(7, 1001, '2023-02-10', 650.00),
(8, 1001, '2023-04-05', 350.00),
(9, 1005, '2023-03-15', 900.00),
(10, 1001, '2023-01-25', 750.00);

```

## Execution:

The screenshot shows two instances of the Visual Studio Code interface. Both instances have the same setup: Explorer on the left, a central code editor with an SQL script, and a Preview tab on the right displaying a table of data.

**Top Instance (SQL Class 2 Assignment):**

- Explorer:** Shows a database named "growthskills 8.3.0" containing tables like "customers" and "orders".
- Code Editor:** Contains the following SQL script:
 

```
-- Inserting some random values in pre-created Orders TABLE
INSERT INTO Orders (OrderID, CustomerID, OrderDate, TotalAmount)
VALUES
(1, 1001, '2023-01-15', 450.00),
(2, 1002, '2023-02-20', 700.00),
(3, 1003, '2023-01-05', 300.00),
(4, 1001, '2023-03-10', 800.00),
(5, 1001, '2023-02-28', 550.00),
(6, 1004, '2023-01-20', 480.00),
(7, 1001, '2023-02-10', 650.00),
(8, 1001, '2023-04-05', 350.00),
(9, 1005, '2023-03-15', 900.00),
(10, 1001, '2023-01-25', 750.00); 8ms
```
- Preview Tab:** Shows a table titled "Preview README.md" with columns "OrderID", "CustomerID", "OrderDate", and "TotalAmount". It displays 10 rows of data inserted from the script. A red box highlights the last row, and a message "VALUES inserted Successfully !!".
- Status Bar:** Shows "Cost: 8ms".

**Bottom Instance (SQL Class 2 Assignment):**

- Explorer:** Shows a database named "growthskills 8.3.0" containing tables like "customers" and "orders". The "orders" table is selected and highlighted with a red box.
- Code Editor:** Contains the same SQL script as the top instance.
- Preview Tab:** Shows a table titled "orders" with columns "OrderID", "CustomerID", "OrderDate", and "TotalAmount". It displays 10 rows of data inserted from the script. A red box highlights the last row, and a message "VALUES inserted Successfully !!".
- Status Bar:** Shows "Cost: 3ms" and "Total 10".

- SQL query to retrieve the order IDs and total amounts of orders placed by customer ID 1001 after January 1, 2023, or orders with a total amount exceeding \$500

## Query:

```
SELECT OrderID, TotalAmount
FROM Orders
WHERE (CustomerID = 1001 AND OrderDate > '2023-01-01')
OR TotalAmount > 500;
```

**Execution:**

```

SELECT OrderID, TotalAmount
FROM Orders
WHERE (CustomerID = 1001 AND OrderDate > '2023-01-01')
    OR TotalAmount > 500;
  
```

CustomerID = 1001 AND  
OrderDate > '2023-01-01'  
OR TotalAmount > 500

OrderID	TotalAmount
1	450.00
2	700.00
3	800.00
4	550.00
5	650.00
6	350.00
7	900.00
8	750.00

Q15. Retrieve the ProductName of products from the "Products" table that have a price between \$50 and \$100.

**Query:**

```

SELECT ProductName
FROM Products
WHERE Price BETWEEN 50 AND 100;
  
```

**Execution:**

The screenshot shows a SQL development environment with the following details:

- EXPLORER:** Shows the database structure for "sakila" including tables like "books", "customers", "employees", "orders", and "products".
- SQl\_Class\_2\_Assignment.sql:** Contains the following code for Q15:
 

```

253 | 2.SQL > 0.Assignment > Solutions > Sql_Class_2_Assignment.sql ...
254 | 254 | D Execute | JSON
255 | 255 | SELECT OrderID, TotalAmount
256 | 256 | FROM Orders
257 | 257 | WHERE (CustomerID = 1001 AND OrderDate > '2023-01-01')
258 | 258 | OR TotalAmount > 500;      You, 45 seconds ago * Uncommitted changes
259 |
260 | 259 | /*
261 | 261 | Q15. Retrieve the ProductName of products from the "Products" table that
262 | 262 | between $50 and $100.
263 | 263 | */
      
```
- Preview README.md:** Shows the execution results for the query, displaying a table with columns "OrderID", "CustomerID", "EmployeeID", "OrderDate", "TotalAmount", and "Status". A yellow box highlights the WHERE clause of the query.
- Products Table:** Shows a list of products with their names: Gizmo X, Doohickey, Widget B, and Doodad.
- Handwritten Note:** A large yellow bracket groups the four products and points to the text "Products → price are b/w 50 & 100".

Q16. Retrieve the names of employees from the "Employees" table who are both from the "Sales" department and have an age greater than 25, or they are from the "Marketing" department.

- There is no Department column in the "Employees" table so, let's first alter the "Employees" table and add up one more column named it as "Department"

#### Query:

```
ALTER TABLE Employees
ADD COLUMN Department varchar(255);
```

#### Execution:

The screenshot shows a code editor interface with multiple tabs and panes. On the left, there's a sidebar with project navigation and file lists. The main area contains several SQL scripts and their execution results.

- Sql\_Class\_2\_Assignment.sql:** Contains various SQL statements, including a query for products between \$50 and \$100 and a script to add a 'Department' column to the 'Employees' table.
- ALTER TABLE Employees ADD COLUMN Department varchar(255);**: The execution result shows the command was run successfully with a cost of 19ms and 0 affected rows.
- employees:** A table viewer showing 10 rows of employee data. The 'Department' column for all rows is currently null.

**Q16. Retrieve the names of employees from the "Employees" table who are both from the "Sales" department and have an age greater than 25, or they are from the "Marketing" department.**

**WE can SEE that  
Department column have  
NULL values so we have to  
insert some values for  
each record**

- let's now filled all the NULL values in Department column

```
UPDATE Employees
SET Department = 'Sales' WHERE EmployeeID IN (1, 2, 5);
UPDATE Employees
SET Department = 'Marketing' WHERE EmployeeID IN (3, 6, 10);
UPDATE Employees
SET Department = 'Finance' WHERE EmployeeID IN (4, 7, 8);
UPDATE Employees
SET Department = 'HR' WHERE EmployeeID IN (9);
```

## Execution:

The screenshot shows a SQL development environment with the following components:

- Explorer:** Shows the database structure under "DATA ANALYST".
- Query Editor:** Displays a script for updating the "Employees" table based on department.
- Results Grid:** Shows the "Employees" table with 10 rows of data.
- Preview Pane:** Shows the updated state of the "Employees" table with the "Department" column populated.

Handwritten annotations:

- A curly brace on the right side of the screen groups the "Query Editor" and "Preview Pane" sections, with the word "Query" written above it.
- A large curly brace on the right side of the screen groups the "Results Grid" and "Preview Pane" sections, with the text "Now Department column have NO! NULL values" written above it.

EmployeeID	FirstName	LastName	Salary	Age	Department
1	Shivcharan	Das	45000.00	32	Sales
2	Krish	Naik	60000.00	35	Sales
3	Rohit	Kapoor	75000.00	42	Marketing
4	Ekta	Dubey	95000.00	36	Finance
5	Shailja	Mishra	80000.00	23	Sales
6	Hariharan	S	65000.00	25	Marketing
7	Aman	Gupta	90000.00	45	Finance
8	Arpit	Dubey	90800.00	26	Finance
9	Shreya	Richharia	42000.00	27	HR
10	Sudhanshu	Kumar	30000.00	40	Marketing

- Retrieving, who are both from the "Sales" department and have an age greater than 25, or they are from the "Marketing" department.

### Query:

```
SELECT FirstName, LastName
FROM Employees
WHERE (`Department` = 'Sales' AND `Age` > 25)
OR `Department` = 'Marketing';
```

### Execution:

The screenshot shows a SQL development environment with the following details:

- EXPLORER:** Shows the database structure for 'sakila' and 'sql\_class\_2\_assignment'.
- SQL Editor:** Contains a script for updating employees in the 'Finance' and 'HR' departments, followed by a query to select employees from 'Sales' or 'Marketing' departments where age is greater than 25.
- Execution Results:** Displays the results of the query, showing five rows of employee data.
- Data Grid:** Shows the same five rows of employee data with columns labeled 'FirstName' and 'LastName'.
- Handwritten Note:** A yellow bracket on the right side of the screen groups the last two rows of the grid and points to the handwritten note below.
- Handwritten Note Content:**

Names of Employees  
who are from "Sales"  
and age > 25 or they are  
from "Marketing" department

Q17. Retrieve the names of customers from the "Customers" table who are not from the city 'New York' or 'Los Angeles.

- Our Customers table does not have **city** column so let add that column in Customers table.

#### Query:

```
ALTER TABLE Customer
ADD COLUMN city varchar(255);
```

#### Execution:

Q17. Retrieve the names of customers from the "Customers" table who are not from the city 'New York' or 'Los Angeles.'

ALTER TABLE Customers  
ADD COLUMN city varchar(255);

CustomerID	FirstName	LastName	Email	Age	city
1001	John	Doe	john.doe@example.com	35	(NULL)
1002	Jane	Smith	jane.smith@example.com	28	(NULL)
1003	Michael	Johnson	michael.johnson@example.com	42	(NULL)
1004	Emily	Davis	emily.davis@example.com	29	(NULL)
1005	Andrew	Wilson	andrew.wilson@example.com	37	(NULL)
1006	Jessica	Brown	jessica.brown@example.com	31	(NULL)
1007	William	Martinez	william.martinez@example.com	45	(NULL)
1008	Olivia	Garcia	olivia.garcia@example.com	26	(NULL)
1009	David	Anderson	david.anderson@example.com	33	(NULL)
1010	Sophia	Lopez	sophia.lopez@example.com	40	(NULL)

A added  
city column  
Successfully

- Updating city column because it contains NULL values.

## Query:

```
UPDATE Customers
SET City = 'New York'
WHERE CustomerID = 1001;

UPDATE Customers
SET City = 'Los Angeles'
WHERE CustomerID = 1002;

UPDATE Customers
SET City = 'London'
WHERE CustomerID = 1003;

UPDATE Customers
SET City = 'Sydney'
WHERE CustomerID = 1004;

UPDATE Customers
SET City = 'Tokyo'
WHERE CustomerID = 1005;

UPDATE Customers
SET City = 'Paris'
WHERE CustomerID = 1006;

UPDATE Customers
SET City = 'Berlin'
WHERE CustomerID = 1007;
```

```

UPDATE Customers
SET City = 'Toronto'
WHERE CustomerID = 1008;

UPDATE Customers
SET City = 'Dubai'
WHERE CustomerID = 1009;

UPDATE Customers
SET City = 'Mumbai'
WHERE CustomerID = 1010;

```

## Execution:

The screenshot shows the Visual Studio Code interface with the following details:

- Explorer View:** Shows the project structure under "DATA: DATABASE" and "DATA ANALYST".
- Sql\_Class\_2\_Assignment.sql File:** Contains three UPDATE statements:
  - Line 335: `UPDATE Customers SET City = 'Toronto' WHERE CustomerID = 1008;`
  - Line 340: `UPDATE Customers SET City = 'Dubai' WHERE CustomerID = 1009;`
  - Line 345: `UPDATE Customers SET City = 'Mumbai' WHERE CustomerID = 1010;` (highlighted with a yellow box)
- Preview README.md View:** Shows a preview of the README.md file with some code snippets.
- Data Grid View:** Displays a table of customer data with columns: CustomerID, FirstName, LastName, Email, Age, and city. The data is as follows:
 

CustomerID	FirstName	LastName	Email	Age	city
1001	John	Doe	john.doe@example.com	35	New York
1002	Jane	Smith	jane.smith@example.com	28	Los Angeles
1003	Michael	Johnson	michael.johnson@example.com	42	London
1004	Emily	Davis	emily.davis@example.com	29	Sydney
1005	Andrew	Wilson	andrew.wilson@example.com	37	Tokyo
1006	Jessica	Brown	jessica.brown@example.com	31	Paris
1007	William	Martinez	william.martinez@example.com	45	Berlin
1008	Olivia	Garcia	olivia.garcia@example.com	26	Toronto
1009	David	Anderson	david.anderson@example.com	33	Dubai
1010	Sophia	Lopez	sophia.lopez@example.com	40	Mumbai

- Retrieve the names of customers who are not from the city 'New York' or 'Los Angeles'

## Query:

```

SELECT FirstName, LastName
FROM Customers
WHERE city NOT IN ('New York', 'Los Angeles');

```

## Execution:

The screenshot shows a SQL development environment with the following details:

- EXPLORER:** Shows the database structure for 'sakila' including tables like books, customers, employees, orders, products, and students.
- SQLEditor:** Contains two queries:
  - Query 1 (Line 341): `SET City = 'Dubai'`
  - Query 2 (Line 350): `SELECT FirstName, LastName FROM Customers WHERE city NOT IN ('New York', 'Los Angeles');`
- Results:** A table titled 'Customers' displays 8 rows of data with columns 'FirstName' and 'LastName'. The data is:
 

	FirstName	LastName
1	Michael	Johnson
2	Emily	Davis
3	Andrew	Wilson
4	Jessica	Brown
5	William	Martinez
6	Olivia	Garcia
7	David	Anderson
8	Sophia	Lopez
- Notes:** A large yellow bracket on the right side of the interface groups the results table and the explanatory text below it, which reads: "Customers Records who are NOT from the cities 'New York' or 'Los Angeles'".

Q18. Retrieve the names of employees from the "Employees" table who are either from the "HR" department and have an age less than 30, or they are from the "Finance" department and have an age greater than or equal to 35.

#### Query:

```
SELECT FirstName, LastName
FROM employees
WHERE (`Department` = 'HR' AND `Age` < 30)
OR (`Department` = 'Finance' AND `Age` >= 35)
```

#### Execution:

**Q18. Retrieve the names of employees from the "Employees" table who are either from the "HR" department and have an age less than 30, or they are from the "Finance" department and have an age greater than or equal to 35.**

**Query:**

```
SELECT FirstName, LastName
FROM employees
WHERE (Department = 'HR' AND Age < 30)
OR (Department = 'Finance' AND Age >= 35) 3ms
```

**Diagram:** A handwritten diagram shows the logic for the query. It starts with 'employees' at the top, which branches into 'Dept' (Department) and 'Age'. 'Dept' further branches into 'HR' and 'Finance'. 'Age' branches into '< 30' and '>= 35'.

Employee ID	First Name	Last Name
1	Ekta	Dubey
2	Aman	Gupta
3	Shreya	Richharia

**Q19. Retrieve the names of customers from the "Customers" table who are not from the city 'London' and either have a postal code starting with '1' or their country is not 'USA'.**

- We donot have country or postal code in Customers table so let's add them FIRST!

#### Query:

```
ALTER TABLE Customers
ADD COLUMN PostalCode varchar(20),
ADD COLUMN Country varchar(255);
```

#### Execution:

The screenshot shows a code editor interface with several tabs open. The main tab contains a SQL query:

```

2. SQL > 0_Assignment > Solutions > Sql_Class_2_Assignment.sql > ...
365 /**
366 Q19. Retrieve the names of customers from the "Customers" table
367 who are not from the city 'London' and either have a postal code starting with '1'
368 or their country is not 'USA'.
369 */
370 -- We don't have country or postal code in Customers table so
371 -- let's add them FIRST
372
373 /*
374 Execute | JSON
375 ALTER TABLE Customers
376 ADD COLUMN PostalCode varchar(20),
377 ADD COLUMN Country varchar(255);

```

A red box highlights the ALTER TABLE statement. To the right, a preview window shows the 'customers' table with two new columns: 'PostalCode' and 'Country', both containing NULL values.

**Q19. Retrieve the names of customers from the "Customers" table who are not from the city 'London' and either have a postal code starting with '1' or their country is not 'USA'.**

**AS WE CAN SEE:**  
**PostalCode and Country columns are added successfully but having NULL values so let's UPDATE these values.**

- Updating Values for PostalCode and Country columns in Customers table

### Query:

```

UPDATE Customers
SET PostalCode = '10001', Country = 'USA'
WHERE CustomerID = 1001;

UPDATE Customers
SET PostalCode = '20002', Country = 'USA'
WHERE CustomerID = 1002;

UPDATE Customers
SET PostalCode = 'WC1A 1AA', Country = 'UK'
WHERE CustomerID = 1003;

UPDATE Customers
SET PostalCode = '2000', Country = 'Australia'
WHERE CustomerID = 1004;

UPDATE Customers
SET PostalCode = '100-0005', Country = 'Japan'
WHERE CustomerID = 1005;

UPDATE Customers
SET PostalCode = '75001', Country = 'France'
WHERE CustomerID = 1006;

UPDATE Customers
SET PostalCode = '10115', Country = 'Germany'
WHERE CustomerID = 1007;

```

```

UPDATE Customers
SET PostalCode = 'M5V 2Z5', Country = 'Canada'
WHERE CustomerID = 1008;

UPDATE Customers
SET PostalCode = '12345', Country = 'UAE'
WHERE CustomerID = 1009;

UPDATE Customers
SET PostalCode = '400001', Country = 'India'
WHERE CustomerID = 1010;

```

## Execution:

The screenshot shows a SQL development environment with the following details:

- Explorer:** Shows the database structure with tables like books, customers, employees, orders, products, and views.
- SQL Editor:** Contains three UPDATE statements:
  - Line 408: UPDATE Customers SET PostalCode = 'M5V 2Z5', Country = 'Canada' WHERE CustomerID = 1008;
  - Line 412: UPDATE Customers SET PostalCode = '12345', Country = 'UAE' WHERE CustomerID = 1009;
  - Line 416: UPDATE Customers SET PostalCode = '400001', Country = 'India' WHERE CustomerID = 1010; (highlighted with a red box)
- Preview:** Shows the results of the last UPDATE statement (CustomerID 1010):
 

CustomerID	FirstName	LastName	Email	Age	city	PostalCode	Country
1001	John	Doe	john.doe@example.com	35	New York	10001	USA
1002	Jane	Smith	jane.smith@example.com	28	Los Angeles	20002	USA
1003	Michael	Johnson	michael.johnson@example.com	42	London	WC1A 1AA	UK
1004	Emily	Davis	emily.davis@example.com	29	Sydney	2000	Australia
1005	Andrew	Wilson	andrew.wilson@example.com	37	Tokyo	100-0005	Japan
1006	Jessica	Brown	jessica.brown@example.com	31	Paris	75001	France
1007	William	Martinez	william.martinez@example.com	45	Berlin	10115	Germany
1008	Olivia	Garcia	olivia.garcia@example.com	26	Toronto	M5V 2Z5	Canada
1009	David	Anderson	david.anderson@example.com	33	Dubai	12345	UAE
1010	Sophia	Lopez	sophia.lopez@example.com	40	Mumbai	400001	India
- Table View:** Shows the same data from the preview in a standard table format.
- Note:** A red box highlights the third UPDATE statement, and a red arrow points to the table with the text: "Now we have values for both the columns PostalCode and Country column apart from NULL VALUES".

- Retrieve the names of customers, who are not from the city 'London' and either have a postal code starting with '1' or their country is not 'USA'.

## Query:

```

SELECT FirstName, LastName
FROM Customers
WHERE `city` != 'London'
AND (`PostalCode` LIKE '1%' OR `Country` != 'USA');

```

## Execution:

The screenshot shows a Visual Studio Code interface with several tabs open. The main tab contains a SQL script named `Sql_Class_2_Assignment.sql`. The script includes DDL statements for creating tables like `employees`, `orders`, and `products`, and DML statements for inserting data into `customers` and `order_details`. A specific query is highlighted:

```
SELECT FirstName, LastName
FROM Customers
WHERE city != 'London'
AND (PostalCode LIKE '1%' OR Country != 'USA');
```

The results of this query are displayed in a table titled "Customers" in the bottom pane, showing 8 rows of data. A callout box highlights the results with the text: "WE Got the customers names who where not from London city and their postal code start with 1 and who are not from USA".

---

**Submitted By: Arpit Dubey**

**Email ID: aarpitdubey@gmail.com**

**LinkedIn: [click here](#)**

**Github : [click here](#)**

---