

- **ASSIGNMENT - 3**

- Patter Matching

- Question 1: How can you select all employees whose names start with the letter 'A'?
 - Question 2: How do you find all products whose names contain the word 'phone' regardless of case?
 - Question 3: How can you retrieve all email addresses from a table that end with '.com'?
 - Question 4: How do you find all phone numbers that start with the area code '555'?
 - Question 5: How can you select all cities that start with 'New' followed by any characters?
 - Question 6: How do you find all records where the value in the 'description' column contains either 'apple' or 'orange'?
 - Question 7: How can you retrieve all email addresses that follow the pattern of "user@domain.com"?
 - Question 8: How do you find all records where the 'product_code' is exactly four characters long and consists of letters and digits?
 - Question 9: How can you retrieve all phone numbers that match the pattern '###-###-####'?
 - Question 10: How do you find all records where the 'text' column contains two consecutive digits?

- **NULL VALUES:**

- Question 1: Find all employees whose birthdates are not recorded (NULL).
 - Question 2: List all orders that don't have a customer assigned (NULL customerID).

- **FUNCTIONS:**

- Question 1: Find the total quantity sold for each product.
 - Question 2: Calculate the total revenue generated from each product (Total Revenue = Quantity * Price).
 - Question 3: Determine the average price of each product.
 - Question 4: Find the product with the highest total revenue (Quantity * Price)
 - Question 5: Calculate the total quantity sold across all products.
 - Question 6: Determine the average price of all products.
 - Question 1: Determine the square root of the price for each product.
 - Question 2: Find the ceiling (smallest integer greater than or equal to) of the prices.
 - Question 3: Calculate the floor (largest integer less than or equal to) of the prices.
 - Question 1: Find the difference in days between the order date and delivery date for each order.
 - Question 2: Calculate the total delivery time in hours for all orders.
 - Question 3: Determine the day of the week when each order was placed.
 - Question 4: Find the orders that were placed on a Saturday (DayOfWeek = 7).
 - Question 5: Calculate the average delivery time in days for all orders.
 - Question 6: Find the orders that were delivered on the same day they were placed.

- **GROUP BY, HAVING, ORDER BY**

- Question 1: Consider a table named Orders with the following columns:
 - Write an SQL query to find the customer IDs of customers who have placed orders with a total amount greater than \$1,000.
 - Question 2: Consider a table named Sales with the following columns:

- Write an SQL query to find the product IDs of products that have been sold in quantities greater than 100 on at least three different sale dates.
- Question 3: Consider a table named Employees with the following columns:
- Write an SQL query to find the average salary of employees in each department, but only for departments where the average salary is greater than \$60,000.
- Question 4: Consider a table named Students with the following columns:
- Write an SQL query to find the course names in which the average score of all students is greater than or equal to 80.
- Question 5: Consider a table named Employees with the following columns:
- Write an SQL query to find the department with the highest average salary.
- Question 6: Consider a table named Sales with the following columns:
- Write an SQL query to find the product with the highest total quantity sold.
- Question 7: Consider a table named Students with the following columns:
- Write an SQL query to find the top three students with the highest average score across all courses.
- Question 8: Consider a table named Orders with the following columns:
- Write an SQL query to find the total amount of orders placed by each customer, ordered in descending order of total amount.
- Question 9: Consider a table named Books with the following columns:
- Write an SQL query to find the number of books published by each author in descending order of the count.
- Question 8: Consider a table named Orders with the following columns:
- Write an SQL query to find the customer IDs of customers who have placed orders with a total amount greater than \$1,000 and have placed at least two orders.
- Question 11: Consider a table named Products with the following columns:
- Write an SQL query to find the average price of products in each category, ordered by category name in ascending order.
- Question 12: Consider a table named Employees with the following columns:
- Write an SQL query to find the department(s) with the lowest average salary.
- Question 13: Consider a table named Orders with the following columns:
- Write an SQL query to find the customer IDs of customers who have placed orders with a total amount greater than \$500 in the year 2023, ordered by customer ID in ascending order.
- Question 14: Consider a table named Students with the following columns:
- Write an SQL query to find the course names in which the highest score achieved by any student is greater than or equal to 90, ordered by course name in ascending order
- Question 15: Consider a table named Orders with the following columns:
- Write an SQL query to find the customer IDs of customers who have placed orders with a total amount greater than \$500 in the year 2023 and have placed at least two orders
- Question 16: Consider a table named Students with the following columns:
- Write an SQL query to find the course names where the average score of students who scored less than 70 in at least one course is greater than or equal to 80.
- Question 17: Consider a table named Employees with the following columns:

- Write an SQL query to find the departments where the highest salary is greater than \$80,000 and the total number of employees in that department is at least 5.
- Question 18: Consider a table named Products with the following columns:
- Write an SQL query to find the categories where the average price of products is greater than or equal to 100.
- Question 19: Consider a table named Orders with the following columns:
- Write an SQL query to find the customer IDs of customers who have placed orders with a total amount greater than \$1,000 in any single order and have placed orders on at least three different dates.

ASSIGNMENT - 3

Patter Matching

Question 1: How can you select all employees whose names start with the letter 'A'?

Solution : Here, we have to **SELECT** all the employees whose **FirstName** starts with the letter 'A' means we have to match the initial letter 'A' using the **WHERE** and **LIKE** we can find all the employees.

Query :

```
SELECT *
FROM employees
WHERE FirstName LIKE 'A%';
```

Execution :

The screenshot shows a Visual Studio Code interface with several tabs open. In the Explorer view, there's a 'DATABASE DATABASE' section with a folder named 'growdatskills...'. Inside it, there's a 'sql_class_3_assignment' folder containing a 'Query' folder with a 'employees' table. The 'employees' table has 10 rows. In the main editor area, a SQL script is shown with a red box highlighting the WHERE clause of a query:

```
-- Question 1: How can you select all employees whose names start with the letter 'A'??
SELECT *
FROM employees
WHERE FirstName LIKE 'A%';
```

Below the editor, a preview pane shows the results of the query:

	EmployeeID	FirstName	LastName	Salary	Age	Department
1	7	Aman	Gupta	90000	45	Finance
2	8	Arpit	Dubey	90800	26	Finance

Question 2: How do you find all products whose names contain the word 'phone' regardless of case?

Let's first look at the product table

Query :

```
-- Query to view the list of products
SELECT *
FROM `sql_class_3_assignment`.`products`;
```

Execution :

Question 2: How do you find all products whose names contain the word 'phone' regardless of case?

Let's first look at the product table

```
-- Query to view the list of products
SELECT *
FROM `sql_class_3_assignment`.`products`;
```

ProductID	ProductName	Price	InStock
1	Widget A	120	1
2	Gizmo X	80.5	0
3	Thingamajig	150	1
4	Moto G Phone	15600	1
5	Whatchamacallit	110	0
6	Widget B	95	1
7	I Phone 15 Pro max	195000	0
8	Contraption Z	180	1
9	Google Nexus Phone	75000	0
10	Device Q	105	1

We can clearly observe here we have 3 - Products whose name having 'Phone'

Now, We have to find only those products whose name contain 'phone'

Query :

```
SELECT `ProductID`, `ProductName`, `Price`
FROM products
WHERE ProductName LIKE '%phone%';
```

Execution :

Now, We have to find only those products whose name contain 'phone'

Query :

```
SELECT `ProductID`, `ProductName`, `Price`
FROM products
WHERE ProductName LIKE '%phone%';
```

ProductID	ProductName	Price
4	Moto G Phone	15600
7	I Phone 15 Pro max	195000
9	Google Nexus Phone	75000

Here We found 3 Products which contains the Phone regardless of it's case in their name

Question 3: How can you retrieve all email addresses from a table that end with '.com'?

firstly, let's look at the employee table

Query :

```
SELECT *
FROM `sql_class_3_assignment`.`employee`;
```

Execution :

Here We found 3 Products which contains the Phone regardless of it's case in their name

Question 3: How can you retrieve all email addresses from a table that end with '.com'?

firstly, let's look at the employee table

```
SELECT *
FROM `sql_class_3_assignment`.`employee`;
```

emplo	last_name	first_name	title	repo	level	birth	hire	add	city	state	coun	postc	pho	fax	email
1	Adams	Andrew	General Manager	9	L6	18-02-	14-08-	11120	Edmon	AB	Canada	T5K 2N	+1 (780)	+1 (780)	andrew@chinookcorp.com
2	Edwards	Nancy	Sales Manager	1	L4	08-12-	01-05-	825 8 A	Calgary	AB	Canada	T2P 2T3	+1 (403)	+1 (403)	nancy@chinookcorp.com
3	Peacock	Jane	Sales Support Asst	2	L1	29-08-	01-04-	1111 6	Calgary	AB	Canada	T2P 5M	+1 (403)	+1 (403)	jane@chinookcorp.com
4	Park	Margaret	Sales Support Asst	2	L1	19-09-	03-05-	683 10	Calgary	AB	Canada	T2P 5G2	+1 (403)	+1 (403)	margaret@chinookcorp.com
5	Johnson	Steve	Sales Support Asst	2	L1	03-03-	17-10-	7727 B	Calgary	AB	Canada	T3B 1Y7	1 (780)	1 (780)	steve@chinookcorp.com
6	Mitchell	Michael	IT Manager	1	L3	01-07-	17-10-	5827 Br	Calgary	AB	Canada	T3B 0C9	+1 (403)	+1 (403)	michael@chinookcorp.com
7	King	Robert	IT Staff	6	L2	29-05-	02-01-	590 Co	Lethbridge	AB	Canada	T1K 5N1	+1 (403)	+1 (403)	robert@chinookcorp.com
8	Callahan	Laura	IT Staff	6	L2	09-01-	04-03-	923 7 S	Lethbridge	AB	Canada	T1H 1Y1	+1 (403)	+1 (403)	laura@chinookcorp.com

Here, we can see there are email addresses which contains '.com'

Query :

```
SELECT `email`
FROM employee
WHERE email LIKE '%.com';
```

Execution :

Question 3: How can you retrieve all email addresses from a table that end with '.com'?

```

2.SQL > O.Assignment > Solutions > Assignment3 > SQL_Class_3_Assignment.sql > 11d
30  /*
31   Question 3: How can you retrieve all email addresses from a table
32   that end with '.com'?
33  */
34
35 -- To solve this firstly we have view our employee table
36
37 ▷ Execute | JSON
38 SELECT *
39 FROM `sql_class_3_assignment`.`employee`;
40
41 -- Retrieving all email addresses from employee table
42 -- that ends with '.com'
43
44 ▷ Execute | JSON
45 SELECT `email`
46 FROM `employee`
47 WHERE `email` LIKE "%.com"; 13ms

```

Execution:

```

SELECT `email`
FROM `employee`
WHERE `email` LIKE "%.com";

```

Results:

email
andrew@chinookcorp.com
nancy@chinookcorp.com
jane@chinookcorp.com
margaret@chinookcorp.com
steve@chinookcorp.com
michael@chinookcorp.com
robert@chinookcorp.com
laura@chinookcorp.com

Got the results

Question 4: How do you find all phone numbers that start with the area code '555'?

Query :

```

SELECT *
FROM `sql_class_3_assignment`.`amazon`;

```

Execution :

Execution:

```

2.SQL > O.Assignment > Solutions > Assignment3 > SQL_Class_3_Assignment.sql > ...
54 ▷ Execute | JSON
55 SELECT *
56 FROM `sql_class_3_assignment`.`amazon`; 6ms

```

Results:

Name	Email	Address	Identity Proof	Product	Price	Phone Number
John Doe	john doe@email.com	123 Main St	License	Product A	\$100	555-123-4567
Jane Smith	jane smith@email.com	456 Elm St	Passport	Product B	\$150	(555)987-6543
Bob Johnson	bob johnson@email.com	789 Oak St	ID Card	Product C	\$200	555-876-5432
Alice Williams	alice@email.com	101 Pine St	Passport	Product D	\$175	555-234-5678
Eve Brown	eve brown@email.com	111 Oak St	License	Product E	\$120	555-789-0123
Michael Lee	michael@email.com	222 Elm St	Passport	Product F	\$130	555-555-5555
Samantha Clark	samantha@email.com	333 Pine St	ID Card	Product G	\$180	555-999-8888
Daniel Wilson	daniel@email.com	444 Oak St	Passport	Product H	\$190	555-444-3333
Sophia Garcia	sophia@email.com	555 Elm St	License	Product I	\$160	555-777-6656
Oliver Martinez	oliver@email.com	666 Pine St	ID Card	Product J	\$170	555-222-1111
Lauren Taylor	lauren@email.com	777 Oak St	License	Product K	\$110	123-456-7890
Liam Rodriguez	liam@email.com	888 Elm St	Passport	Product L	\$140	456-789-0123
Nora Hernandez	nora@email.com	999 Pine St	ID Card	Product M	\$150	789-012-3456
Emma Nguyen	emma@email.com	1010 Oak St	License	Product N	\$125	321-654-9870
William Smith	william@email.com	1111 Elm St	Passport	Product O	\$135	888-555-4444
Ava Johnson	ava@email.com	1212 Pine St	ID Card	Product P	\$155	777-555-6666
James Brown	james@email.com	1313 Oak St	License	Product Q	\$115	234-567-8901

Got the results

Area Code

Amazon Customer Data

To find out all the phone numbers that start with the area code '555' are

Query :

```
SELECT `Phone Number`  
FROM amazon  
WHERE `Phone Number` LIKE '555%';
```

Execution :

Find out all the phone numbers that start with the area code '555' are

Query :

```
SELECT `Phone Number`  
FROM amazon  
WHERE `Phone Number` LIKE '555%';
```

It is working well but here is a catch we didn't get those phone numbers whose area code were written inside parenthesis '(555)' So, I modified the query and in next execution we can see it.

Here, the query working well, but have you noticed we didn't get those phone numbers whose area code were written inside the parenthesis. like (555)

Query :

```
SELECT `Phone Number`  
FROM amazon  
WHERE `Phone Number` LIKE '555%' OR `Phone Number` LIKE '(555)%';
```

Execution :

Here, the query working well, but have you noticed we didn't get those phone numbers whose area code were written inside the parenthesis, like (555)

Query :

```
SELECT `Phone Number`
FROM `amazon`
WHERE `Phone Number` LIKE '555%' OR `Phone Number` LIKE '(555)%';
```

Now we are able to get including those phone numbers whose area code were written inside the parenthesis

Question 5: How can you select all cities that start with 'New' followed by any characters?

Query :

```
SELECT *
FROM `sql_class_3_assignment`.`phone_directory`;
```

Execution :

Now we are able to get including those phone numbers whose area code were written inside the parenthesis

Question 5: How can you select all cities that start with 'New' followed by any characters?

There are 99 rows in this table

This column is city column which includes all the cities

city	county	state	zip	phone1	phone
New Orleans	Orleans	LA	70116	504-621-8927	504-845-1427
Brighton	Livingston	MI	48116	810-292-9388	810-374-9840
Bridgeport	Gloucester	NJ	8014	856-636-8749	856-264-4130
Anchorage	Anchorage				
Hamilton	Butler				
Ashland	Ashland				
Chicago	Cook	IL	60632	773-573-6914	773-924-8565
San Jose	Santa Clara	CA	95111	408-752-3500	408-813-1105
Sioux Falls	Minnehaha	SD	57105	605-414-2147	605-794-4895
Baltimore	Baltimore City	MD	21224	410-655-8723	410-804-4694
Kulpsville	Montgomery	PA	19443	215-874-1229	215-422-8694

Now, let's filter all those cities at start with 'New'

Query :

```
SELECT `first_name`, `last_name`, `company_name`, `city`, `phone`  
FROM `phone_directory`  
WHERE `city` LIKE 'new%';
```

Execution :

The screenshot shows a code editor and a database interface. In the code editor, a file named `SQL_Class_3_Assignment.sql` contains the following SQL query:

```
SELECT *  
FROM `sql_class_3_assignment`.`phone_directory`  
WHERE `city` LIKE 'new%';
```

The database interface shows a table named `phone_directory` with columns: `first_name`, `last_name`, `company_name`, `city`, and `phone`. A red box highlights the `city` column, with a tooltip stating: "This column is city column which includes all the cities". The table data shows several entries, including multiple rows for "New York" and "Newark". A yellow box highlights the entire query in the code editor.

On the right, a preview window shows the results of the query, listing various cities and their details. A red curly brace groups the highlighted text in the code editor and the highlighted row in the preview window, indicating they represent the same data.

Question 6: How do you find all records where the value in the 'description' column contains either 'apple' or 'orange'?

Firstly, view the table

Query :

```
SELECT *  
FROM `sql_class_3_assignment`.`food_inventory`;
```

Execution :

Question 6: How do you find all records where the value in the 'description' column contains either 'apple' or 'orange'?

Firstly, view the table

Query:

```
SELECT *
FROM `sql_class_3_assignment`.`food_inventory`;
```

	id	description	price	quantity	category	expiry_date	supplier
3	10	Apple pie	\$2.00	100	Dessert	2022-11-20	Sweets Bakery
10	10	Candied oranges	\$2.50	100	Snack	2022-10-30	Snack Co.
11	11	Organic Apples	\$1.75	70	Apple	2022-10-17	Organic Farms Ltd.
12	12	Tangerines	\$1.50	120	Citrus	2022-10-22	Sunnyside Plantations
13	13	Apple Sauce	\$2.25	50	Condiment	2022-11-05	Sauce Co.
14	14	Orange Sorbet	\$3.50	40	Dessert	2022-11-10	Ice Cream Emporium
15	15	Dried Apples	\$2.00	90	Snack	2022-10-31	Healthy Snacks Ltd
16	16	Bananas	\$0.50	200	Banana	2022-10-31	Tropical Farms
17	17	Ripe Papayas	\$2.00	60	Papaya	2022-11-05	Island Delights
18	18	Sweet Strawberries	\$3.00	70	Strawberry	2022-10-25	Berry Farms
19	19	Fresh Mangoes	\$2.50	80	Mango	2022-10-30	Tropicana Farms
20	20	Juicy Pineapples	\$2.20	100	Pineapple	2022-11-15	Golden Harvest

Records where the value in the 'description' column contains either 'apple' or 'orange'

Query :

```
SELECT *
FROM `food_inventory`
WHERE `description` LIKE '%apple%' OR `description` LIKE '%orange%';
```

Execution :

Query:

```
SELECT *
FROM `food_inventory`
WHERE `description` LIKE '%apple%' OR `description` LIKE '%orange%';
```

	id	description	price	quantity	category	expiry_date	supplier
1	1	Fresh apple	\$1.00	100	Apple	2022-10-15	ABC Farms
2	2	Juicy oranges	\$0.75	150	Orange	2022-10-10	XYZ Orchards
3	3	Apple cider	\$3.50	75	Beverage	2022-11-20	Drink Co.
4	4	Sliced oranges	\$2.00	120	Orange	2022-10-20	Harvest Farms
5	5	Green apples	\$1.25	90	Apple	2022-10-16	Green Groves
6	6	Orange marmalade	\$2.50	50	Condiment	2022-11-01	Fine Foods Inc.
7	7	Red apples	\$1.50	80	Apple	2022-10-14	Orchard Delights
8	8	Orange juice	\$3.00	60	Beverage	2022-11-25	Citrus Suppliers Ltd.
9	9	Apple pie	\$5.00	25	Dessert	2022-11-28	Sweets Bakery
10	10	Candied oranges	\$2.50	100	Snack	2022-10-30	Snack Co.
11	11	Organic Apples	\$1.75	70	Apple	2022-10-17	Organic Farms Ltd.
12	12	Apple Sauce	\$2.25	50	Condiment	2022-11-05	Sauce Co.
13	13	Orange Sorbet	\$3.50	40	Dessert	2022-11-10	Ice Cream Emporium
14	14	Dried Apples	\$2.00	90	Snack	2022-10-31	Healthy Snacks Ltd

Records where values in description column contains orange.

Records where values in description column contains apple.

Question 7: How can you retrieve all email addresses that follow the pattern of "user@domain.com"?

Let's take the `phone_directory` table and view it first.

Query :

```
SELECT *
FROM `sql_class_3_assignment`.`phone_directory`;
```

Execution :

The screenshot shows a SQL development environment with several panes. On the left is the Explorer pane showing databases and tables. The center pane displays the query:

```
SELECT *
FROM `sql_class_3_assignment`.`phone_directory`;
```

The results pane on the right shows the output of the query, which contains 99 rows of data. A red box highlights the total count 'Total 99' at the bottom of the results table. A yellow box highlights the entire results table.

first_name	last_name	company_name	address	city	state	zip	phor_text	phor_text	email
James	Butt	Benton, John B Jr	6649 N E	New Orle	Orleans	LA	70116	504-62	jbuttt@gmail.com
Josephine	Darakjy	Chanay, Jeffrey A Esc	4 B Blue	Brighton	Livingston	MI	48116	810-29	josephine_darakjy@darakjy.org
Art	Venere	Chemele, James L Cpa	8 W Cerr	Bridgepor	Gloucester	NJ	8014	856-63	art@venere.org
Lenna	Paprocki	Feltz Printing Service	639 Mair	Anchorage	Anchorage	AK	99501	907-38	lpaprocki@hotmail.com
Donette	Foller	Printing Dimensions	34 Cente	Hamilton	Butler	OH	45011	513-57	donette.foller@cox.net
Simona	Morasca	Chapman, Ross E Esc	3 Mcaul	Ashland	Ashland	OH	44805	419-50	simona@morasca.com
Mitsue	Tollner	Morlong Associates	7 Eads St	Chicago	Cook	IL	60632	773-57	mitsue_tollner@yahoo.com
Leota	Dilliard	Commercial Press	7 W Jack	San Jose	Santa Clara	CA	95111	408-75	leota@hotmail.com
Sage	Wieser	Truhlar And Truhlar A	5 Boston	Sioux Fall	Minnehaha	SD	57105	605-41	sage_wieser@cox.net
Kris	Marrier	King, Christopher A E	228 Run	Baltimore	Baltimore	MD	21224	410-65	kris@gmail.com

Query :

```
SELECT `email`
FROM `phone_directory`
WHERE email LIKE '%@%.com'
```

Execution :

SELECT *
FROM `phone_directory`
WHERE email LIKE '%@%.com'

Multiple Domains
are mentioned and
total of 74 entries
are there.

Question 8: How do you find all records where the 'product_code' is exactly four characters long and consists of letters and digits?

Let's look the **product** table

Query :

```
SELECT *
FROM `sql_class_3_assignment`.`product`;
```

Execution :

Question 8: How do you find all records where the 'product_code' is exactly four characters long and consists of letters and digits?

Let's look at the `product` table.

Query:

```
SELECT *
FROM `sql_class_3_assignment`.`product`;
```

Total 30

	id	product_code	product_name	price
1	1	A123	Premium T-Shirt	\$24.99
2	2	B456C	Wireless Headphones	\$89.99
3	3	C789	Leather Wallet	\$34.99
4	4	D890	Stainless Steel Watch	\$149.99
5	5	E234	Yoga Mat	\$29.99
6	6	F567	Portable Bluetooth Speaker	\$44.99
7	7	G901D	Coffee Maker	\$79.99
8	8	H345	Running Shoes	\$59.99
9	9	I678V	Cooking Pan Set	\$99.99
10	10	J912	Backpack	\$39.99

We have total 30 values and rows.

Let's solve the problem using REGEXP

Query :

```
SELECT *
FROM `product`
WHERE `product_code` REGEXP '^[a-zA-Z0-9]{4}$'
```

Execution :

values and rows

lets solve the problem using REGEXP

Query:

```
SELECT *
FROM `product`
WHERE `product_code` REGEXP '^[a-zA-Z0-9]{4}$'
```

Execution :

(total 30 rows)

Product_code len == 4

	id	product_code	product_name	price
1	1	A123	Premium T-Shirt	\$24.99
2	3	C789	Leather Wallet	\$34.99
3	4	D890	Stainless Steel Watch	\$149.99
4	5	E234	Yoga Mat	\$29.99
5	6	F567	Portable Bluetooth Speaker	\$44.99
6	8	H345	Running Shoes	\$59.99
7	10	J912	Backpack	\$39.99
8	11	K234	Drone with HD Camera	\$199.99
9	12	L678	Bluetooth Earbuds	\$69.99
10	14	N567	Smart Home Security Camera	\$129.99

Question 9: How can you retrieve all phone numbers that match the pattern '###-###-####'?

Query :

```
SELECT *  
FROM `sql_class_3_assignment`.`contacts`;
```

Execution :

```
SELECT *  
FROM `sql_class_3_assignment`.`contacts`;
```

	id	full_name	phone_number	email
1	1	John Doe	123-456-7890	johndoe@email.com
2	2	Jane Smith	987-654-3210	janesmith@email.com
3	3	Bob Johnson	555123-4567	bobjohnson@email.com
4	4	Alice Williams	333-555-7777	alicewilliams@email.com
5	5	Eve Brown	8889997777	evebrown@email.com
6	6	Michael Davis	444-333-2222	michaeldavis@email.com
7	7	Samantha Clark	666-777-8888	samanthaclark@email.com
8	8	Daniel	111-222-3333	danielwilson@email.com
9	9	Sophia Garcia	777-888-9999	sophiagarcia@email.com
10	10	Olivia Martinez	444-555-6666	oliviamartinez@email.com

lets solve the problem using REGEXP

Query :

```
SELECT `Phone_number`  
FROM `contacts`  
WHERE `Phone_number` REGEXP '^[0-9]{3}-[0-9]{3}-[0-9]{4}$';
```

Execution :

The screenshot shows a code editor interface with several windows open:

- EXPLORER**: Shows the project structure with a folder named "growingdataskills" containing sub-folders like "Views", "Procedures", and "Functions".
- SQL Class 3 Assignment.sql**: A code editor window with the following SQL query:

```
/*
Question 9: How can you retrieve all phone numbers
that match the pattern '###-##-##'?
*/
-- To view the table
SELECT *
FROM `contacts`;
```
- Preview README.md**: A preview window showing a table of contacts with columns "id", "name", and "phone_number". A red arrow points from the text "NOT FOLLOW the format" to the "phone_number" column.
- Query:** A search results window showing a table with one column "phone_number" and 10 rows of data. A green curly brace highlights the entire table.
- Search results**: A search results window showing the same table as the Query window.

Handwritten notes on the right side of the screen:

Phone numbers
which follows the
Format $XXX-XXX-XXXX$

```
SELECT `phone_number`
FROM `contacts`
WHERE `phone_number` REGEXP '^[0-9]{3}-[0-9]{3}-[0-9]{4}$';
```

Question 10: How do you find all records where the 'text' column contains two consecutive digits?

Let's first view the table `corpus`

Query :

```
SELECT *
FROM `sql_class_3_assignment`.`corpus`;
```

Execution :

File Edit Selection View Go Run Terminal Help

SQL_Class_3_Assignment.sql M

```

133 -- Execute | JSON
134 SELECT `phone_number`
135 FROM `contacts`
136 WHERE `phone_number` REGEXP '^\d{3}(\d{3})\d{4}(\d{3})$'
137 /*
138 Question 10: How do you find all records where the 'text' column contains two consecutive digits?
139 */
140 /*
141 -- To view the table
142 */
143 -- Execute | JSON
144 SELECT *
145 FROM `sql_class_3_assignment`.`corpus`;

```

Preview README.md

	Category	Text	Count
1	A	This is a sample text with 123 consecutive digits	5
2	B	Another example with no consecutive digits	3
3	C	789xyz contains consecutive digits	7
4	D	NoConsecutiveDigitsHere	2
5	E	The number 45 is present here	6
6	F	Only single digits in this sentence	4
7	G	6789 has four consecutive digits	9
8	H	There are 33 reasons for this	8
9	I	This sentence has 678 consecutive digits	1
10	K	PU678ZZ	12

Query :

```

SELECT *
FROM `corpus`
WHERE `text` REGEXP '[0-9]{2}';

```

Now let's solve the Question here

Query :

```

SELECT *
FROM `corpus`
WHERE `text` REGEXP '[0-9]{2}';

```

Execution :

File Edit Selection View Go Run Terminal Help

SQL_Class_3_Assignment.sql M

```

140 /*
141 Question 10: How do you find all records where the 'text' column contains two consecutive digits?
142 */
143 /*
144 -- To view the table
145 */
146 /*
147 -- Solution
148 */
149 SELECT *
150 FROM `sql_class_3_assignment`.`corpus`;

```

Preview README.md

	Category	Text	Count
1	A	This is a sample text with 123 consecutive digits	5
2	B	Another example with no consecutive digits	3
3	C	789xyz contains consecutive digits	7
4	D	NoConsecutiveDigitsHere	2
5	E	The number 45 is present here	6
6	G	6789 has four consecutive digits	9
7	H	There are 33 reasons for this	8
8	I	This sentence has 678 consecutive digits	1
9	K	PU678ZZ	12

Now let's solve the Question here

Query :

```

SELECT *
FROM `corpus`
WHERE `text` REGEXP '[0-9]{2}';

```

Execution :

	Category	Text	Count
1	A	This is a sample text with 123 consecutive digits	5
2	C	789xyz contains consecutive digits	7
3	E	The number 45 is present here	6
4	G	6789 has four consecutive digits	9
5	H	There are 33 reasons for this	8
6	I	This sentence has 678 consecutive digits	1
7	K	PU678ZZ	12
8	L	ABC123DEF456	15

NUL VALUES:

Question 1: Find all employees whose birthdates are not recorded (NULL).

let's see the table

Query :

```
SELECT *
FROM `sql_class_3_assignment`.`employees`;
```

Execution :

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

```
-- To view the table
-- Execute | JSON
SELECT *
FROM `sql_class_3_assignment`.`employees`;
```

The results table displays 22 rows of employee data. The 'birthdate' column is highlighted in red for all rows where it contains the value '(NULL)'. A yellow curly brace on the right side of the table indicates that there are some values which are NULL.

EmployeeID	FirstName	LastName	Salary	Age	Birthdate	Department
1	Shivcharan	Das	45000	32	1990-05-21	Sales
2	Krish	Naik	60000	35	1988-12-15	Sales
3	Rohit	Kapoor	75000	42	(NULL)	Marketing
4	Ekta	Dubey	95000	36	1989-09-10	Finance
5	Shailja	Mishra	80000	23	1999-09-16	Sales
6	Hariharan	S	65000	25	1996-04-03	Marketing
7	Aman	Gupta	90000	45	1988-05-20	Finance
8	Arpit	Dubey	90800	26	(NULL)	Finance
9	Shreya	Richharia	42000	27	1996-01-20	HR
10	Sudhanshu	Kumar	30000	40	1987-08-11	Marketing
11	John	Doe	60000	31	1990-05-21	Sales
12	Jane	Smith	52000	25	(NULL)	Marketing
13	Bob	Johnson	75000	33	1988-12-15	HR

let's solve this problem

Query :

```
SELECT *
FROM `employees`
WHERE `birthdate` IS NULL;
```

Execution :

File Edit Selection View Go Run Terminal Help

READER.MD SQL_Class_3_Assignment.sql Preview README.MD

EXPLORER

- DATABASE
 - growsk... +
 - sql_class_3_assignment 160k
 - Query
- DATA ANALYST
- 1_Python
- 2_SQL
 - 0_Assignment
 - Questions
 - Solutions
 - Assignment2
 - Assignment3
 - img
- README.MD
- SQL_Class_3_Assignment.M
- tables
- 1.DDL
- 2.DQL
- 3.DML
- 4.DCL
- 5.TCL
- 6_Delete_vs_Truncate
- 7.PATTERN_REGEX
- 8.SubQuery
- 9.Functions
- README.MD
- 3_MSExcel
- .gitignore
- LICENSE

155 Question 1: Find all employees whose birthdates are not recorded (NULL).
156
157 */
158 -- To view the table
159 Execute | JSON
160 SELECT *
161 FROM `sql_class_3_assignment`.`employees`;

163 -- Solution You, 10 seconds ago • Uncommitted changes
164 Execute | JSON
165 SELECT *
166 FROM `employees`
167 WHERE `birthdate` IS NULL; 8ms

employees

EmployeeID	FirstName	LastName	Salary	Age	Birthdate	Department
1	Rohit	Kapoor	75000	42	(NULL)	Marketing
2	Arpit	Dubey	90800	26	(NULL)	Finance
3	Jane	Smith	52000	25	(NULL)	Marketing
4	Alice	Williams	68000	26	(NULL)	Finance
5	Samantha	Clark	48000	30	(NULL)	PR
6	Sophia	Garcia	69000	25	(NULL)	Accounting
7	Liam	Rodriguez	72000	40	(NULL)	IT

let's solve this problem

Query :

```
SELECT *
FROM `employees`
WHERE `birthdate` IS NULL;
```

Employees whose birthdate values are not recorded or NULL.

Question 2: List all orders that don't have a customer assigned (NULL customerId).

let's see the table

Query :

```
SELECT *
FROM `sql_class_3_assignment`.`customers`;
```

Execution :

File Edit Selection View Go Run Terminal Help ↵ → 🔍 data_analyst

EXPLORER

- DATABASE: DATABASE
 - growdatab... 160k
 - sql_class_3_assignment 160k
 - Tables (10)
 - amazon 17
 - contacts 12
 - corpus 11
 - customers 10
 - employee 8
 - employees 10

DATA ANALYST

 - 1.Python
 - 2.SQL
 - 0_Assignment
 - Questions
 - Solutions
 - Assignment2
 - Assignment3
 - img
 - README.md 9+, M
 - tables
 - 1_DDL
 - 2_DQL
 - 3_DML
 - 4_DCL
 - 5_TCL
 - 6_Delete_vs_Truncate
 - 7_PATTERN_REGEX
 - 8_SubQuery
 - 9_Functions
 - READMEmd
 - 3_MSEExcel
 - .gitignore
 - LICENSE

main.js 0 0 56 0 0 Connect Cloud Code - Sign in growdatabskills sql_class_3_assignment You, 1 second ago Ln 180, Col 19 Spaces: 4 UTF-8 CRLF SQL Go Live Duet AI tabnine starter Prettier

Question 2: List all orders that don't have a customer assigned (NULL customerID).

let's see the table

Query:

```
SELECT *
FROM `sql_class_3_assignment`.`customers`;
```

sql.class_3_assignment.customers

CustomerID	FirstName	LastName	Email	Age	city	PostalCode	Country	Address	OrderText	OrderID
1	(NULL)	John	johnoe@exa...	35	New York	10001	USA	123 Oak St	Shoes	1
2	1002	Jane	jane.smith@e...	28	Los Angeles	20002	USA	456 Pine St	Furniture	2
3	(NULL)	Michael	michael.john...	42	London	WC1A 1AA	UK	789 Elm St	Electronics	3
4	1004	Emily	emily.davis@...	29	Sydney	2000	Australia	1010 Maple A	Books	4
5	(NULL)	Andrew	andrew.wilso...	37	Tokyo	100-0005	Japan	1212 Cedar B	Clothing	5
6	1006	Jessica	jessica.brow...	31	Paris	75001	France	1414 Ash Ln	Home Decor	6
7	1007	William	william.marti...	45	Berlin	10115	Germany	1616 Birch Dr	Appliances	7
8	(NULL)	Olivia	olivia.garcia...	26	Toronto	M5V 2Z5	Canada	1818 Pinecr...	Toys	8
9	1009	David	david.anders...	33	Dubai	12345	UAE	2020 Spruce...	Garden Supplies	9
10	1010	Sophia	sophia.lopez...	40	Mumbai	400001	India	2222 Oakwo...	Sports Equipment	10
11	1011	Christopher	christopher.r...	39	San Francisco	94133	USA	2323 Elm St	Outdoor Gear	11
12	(NULL)	Daniel	daniel.griffin...	36	Seattle	98101	USA	2525 Maple A	Pet Supplies	12

let's solve this problem

Query :

```
SELECT `Order`, `OrderID`, `Country`
FROM `customers`
WHERE `CustomerID` IS NULL;
```

Execution :

File Edit Selection View Go Run Terminal Help ↵ → 🔍 data_analyst

EXPLORER

- DATABASE: DATABASE
 - growdatab... 160k
 - sql_class_3_assignment 160k
 - Tables (10)
 - amazon 17
 - contacts 12
 - corpus 11
 - customers 10
 - employee 8
 - employees 10

DATA ANALYST

 - 1.Python
 - 2.SQL
 - 0_Assignment
 - Questions
 - Solutions
 - Assignment2
 - Assignment3
 - img
 - README.md 9+, M
 - tables
 - 1_DDL
 - 2_DQL
 - customers
 - 3_DML
 - 4_DCL
 - 5_TCL
 - 6_Delete_vs_Truncate
 - 7_PATTERN_REGEX
 - 8_SubQuery
 - 9_Functions
 - READMEmd
 - 3_MSEExcel
 - .gitignore
 - LICENSE

main.js 0 0 56 0 0 Connect Cloud Code - Sign in growdatabskills sql_class_3_assignment You, 7 minutes ago Ln 175, Col 43 Spaces: 4 UTF-8 CRLF SQL Go Live Duet AI tabnine starter Prettier

let's solve this problem

Query:

```
SELECT `Order`, `OrderID`, `Country`
FROM `customers`
WHERE `CustomerID` IS NULL;
```

customers

Order	OrderID	Country
1	Shoes	USA
2	Electronics	UK
3	Clothing	Japan
4	Toys	Canada
5	Pet Supplies	USA
6	Beauty Prod	USA
7	Art Supplies	USA
8	Home Improv	USA
9	Electronic Acc	USA
10	Fitness Equipr	USA

Activate Windows
Go to Settings to activate Windows.

FUNCTIONS:

- Consider a table named Sales with the following columns:

SaleID (integer): The unique identifier for each sale.

Product (string): The name of the product sold.

Quantity (integer): The quantity of the product sold.

Price (decimal): The price per unit of the product.

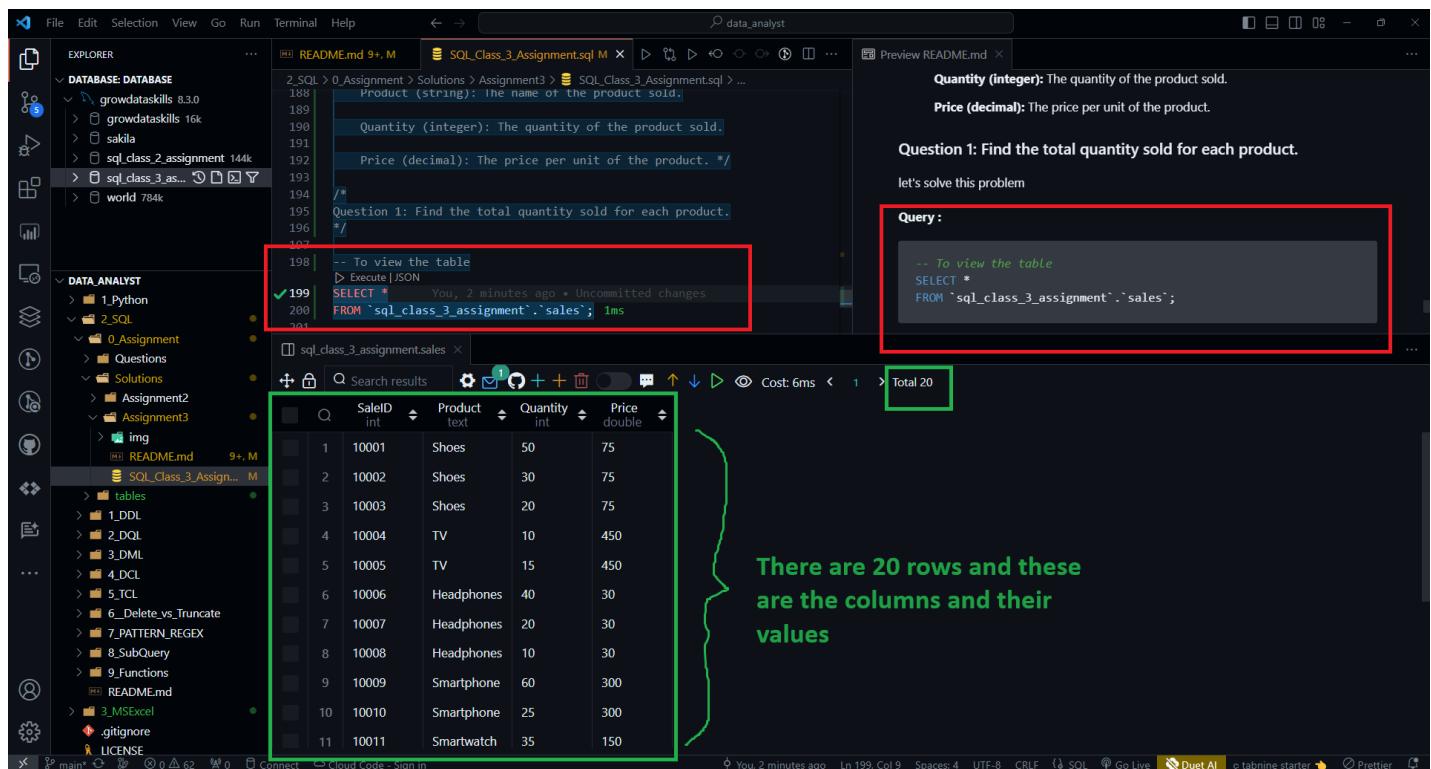
Question 1: Find the total quantity sold for each product.

let's view **sales** table

Query :

```
-- To view the table
SELECT *
FROM `sql_class_3_assignment`.`sales`;
```

Execution :



The screenshot shows a code editor interface with several tabs open. On the left, there's an Explorer sidebar showing project files like README.md, Python scripts, and various assignment folders. The main editor area has a SQL file open with the following content:

```
-- To view the table
SELECT *
FROM `sql_class_3_assignment`.`sales`;
```

The results pane shows a table with 20 rows of data:

	SaleID	Product	Quantity	Price
1	10001	Shoes	50	75
2	10002	Shoes	30	75
3	10003	Shoes	20	75
4	10004	TV	10	450
5	10005	TV	15	450
6	10006	Headphones	40	30
7	10007	Headphones	20	30
8	10008	Headphones	10	30
9	10009	Smartphone	60	300
10	10010	Smartphone	25	300
11	10011	Smartwatch	35	150

A green curly brace on the right side of the table highlights the entire row set, with the caption "There are 20 rows and these are the columns and their values".

let's solve this problem

Query :

```
-- Solution
SELECT `Product`, SUM(Quantity) AS `Total_Qty_Sold`
```

```
FROM `sql_class_3_assignment`.`sales`
GROUP BY `Product`;
```

Execution :

Code Editor Screenshot:

```

194 /*
195 Question 1: Find the total quantity sold for each product.
196 */
197
198 -- To view the table
199 > Execute | JSON
SELECT *
200 FROM `sql_class_3_assignment`.`sales`;
201
202 -- Solution
203 > Execute | JSON
SELECT `Product`, SUM(Quantity) AS `Total_Qty_Sold`
204 FROM `sql_class_3_assignment`.`sales`
205 GROUP BY `Product`;
206

```

Preview Window:

let's solve this problem

Query:

```

-- Solution
SELECT `Product`, SUM(Quantity) AS `Total_Qty_Sold`
FROM `sql_class_3_assignment`.`sales`
GROUP BY `Product`;

```

Table View:

Product	Total_Qty_Sold
Shoes	100
TV	25
Headphones	70
Smartphone	85
Smartwatch	50
Laptop	30
Speaker	80
Desk	25
Chair	60

*Sales of Products
Grouping each categories
and find total Qty Sold*

Question 2: Calculate the total revenue generated from each product (Total Revenue = Quantity * Price).

Query :

```

SELECT `Product`, `Price`, SUM(Quantity) AS 'Total Quantity Sold', SUM(Quantity * Price) AS `Total
Revenue`
FROM `sql_class_3_assignment`.`sales`
GROUP BY `Product`, `Price`;

```

Execution :

Question 2: Calculate the total revenue generated from each product (Total Revenue = Quantity * Price).

Query:

```
SELECT `Product`, `Price`, SUM(Quantity) AS 'Total Quantity Sold', SUM(Quantity * Price) AS 'Total Revenue'
FROM `sql_class_3_assignment`.`sales`
GROUP BY `Product`, `Price`;
```

Total Revenue = Price * SUM(Quantity)

(OR)

Total Revenue = Total Quantity Sold * Price

Product	Price	Total Quantity Sold	Total Revenue
Shoes	75	100	7500
TV	450	25	11250
Headphones	30	70	2100
Smartphone	300	85	25500
Smartwatch	150	50	7500
Laptop	850	30	25500
Speaker	100	80	8000
Desk	200	25	5000
Chair	75	60	4500

Question 3: Determine the average price of each product.

Query :

```
SELECT `Product`, AVG(`Price`) AS 'Average Price'
FROM `sql_class_3_assignment`.`sales`
GROUP BY `Price`;
```

Execution :

Question 3: Determine the average price of each product.

Query:

```
SELECT `Product`, AVG(`Price`) AS 'Average Price'
FROM `sql_class_3_assignment`.`sales`
GROUP BY `Price`;
```

Product	Average Price
Shoes	75
TV	450
Headphones	30
Smartphone	300
Smartwatch	150
Laptop	850
Speaker	100
Desk	200
Chair	75

Question 4: Find the product with the highest total revenue (Quantity * Price)

Query :

```
SELECT `Product`, `Price`, SUM(`Quantity` * `Price`) AS `Total Quantity Sold`
FROM `sql_class_3_assignment`.`sales`
GROUP BY `Product`, `Price`
ORDER BY `Total Quantity Sold` DESC
LIMIT 1;
```

Execution :

Question 5: Calculate the total quantity sold across all products.

Query :

```
-- Solution
SELECT SUM(Quantity) AS OverAllQuantitySold
FROM `sql_class_3_assignment`.`sales`;
```

Execution :

```
-- Solution
SELECT SUM(Quantity) AS OverAllQuantitySold
FROM `sql_class_3_assignment`.`sales`;
```

OverAllQuantitySold
525

Question 6: Determine the average price of all products.

Query :

```
SELECT AVG(`Price`) AS `OverALLAveragePrice`  
FROM `SQL_class_3_assignment`.`sales`;
```

Execution :

The screenshot shows a code editor interface with several tabs and panes. On the left, there's an 'EXPLORER' pane showing a database structure with tables like 'employees', 'food.inventory', 'phone_directory', 'product', and 'sales'. In the center, there's a code editor tab with a file named 'SQL_Class_3_Assignment.sql'. The code contains several comments and queries. A specific section is highlighted with a red box:

```
243 SELECT AVG(`Price`) AS `OverALLAveragePrice`  
FROM `SQL_class_3_assignment`.`sales`;
```

Below the code editor is a results pane showing the output of the query:

OverALLAveragePrice	double
1	228.25

On the right side of the interface, there's a 'Preview README.md' tab and a 'Execution' tab. The 'Execution' tab displays the query and its result, with some parts highlighted by red boxes. It also shows a preview of the 'README.md' file.

- Consider a table named Products with the following columns:

ProductID (integer): The unique identifier for each product.

ProductName (string): The name of the product.

Price (decimal): The price of the product.

Question 1: Determine the square root of the price for each product.

let's view **products** table

Query :

```
-- To view the table  
SELECT *  
FROM `sql_class_3_assignment`.`products`;
```

Execution :

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

- Explorer:** Shows the database structure under "DATABASE: DATABASE".
- Terminal:** Displays the SQL query:

```
2_SQL > 0.Assignment > Solutions > Assignment3 > SQL_Class_3_Assignment.sql > ...
214   SELECT `Product`, AVG(`Price`) AS 'Average Price'
215   FROM `sql_class_3_assignment`.`sales`
216   GROUP BY `Product`;
217
218 /*
219 Question 1: Determine the square root of the price for each product.
220 */
223 -- To view the table
224   SELECT *
225   FROM `sql_class_3_assignment`.`products`;
```
- Preview README.md:** Shows the question and the query to view the table.
- Table View:** Shows the "products" table with 11 rows of data.

Query :

```
-- Solution
SELECT `ProductName`, `Price`, SQRT(`Price`) AS `Square Root`
FROM `sql_class_3_assignment`.`products`
GROUP BY `ProductName`, `Price`;
```

Execution :

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

- Explorer:** Shows the database structure under "DATABASE: DATABASE".
- Terminal:** Displays the SQL query:

```
2_SQL > 0.Assignment > Solutions > Assignment3 > SQL_Class_3_Assignment.sql > ...
218 /*
219 Question 1: Determine the square root of the price for each product.
220 */
223 -- To view the table
224   SELECT *
225   FROM `sql_class_3_assignment`.`products`;
226
227 -- Solution
228   SELECT `ProductName`, `Price`, SQRT(`Price`) AS `Square Root`
229   FROM `sql_class_3_assignment`.`products`
230   GROUP BY `ProductName`, `Price`;
231
```
- Preview README.md:** Shows the question and the query to view the table.
- Table View:** Shows the "products" table with 11 rows of data, including the calculated "Square Root" column.

A large yellow curly brace on the right side of the table highlights the "Square Root" column, with handwritten text "Correctly finds the Square Root of Price" written over it.

Question 2: Find the ceiling (smallest integer greater than or equal to) of the prices.

Query :

```
SELECT `ProductName`, `Price`, CEIL(`Price`) AS `CeiledPrice`  
FROM `sql_class_3_assignment`.`products`
```

Execution :

Question 2: Find the ceiling (smallest integer greater than or equal to) of the prices.

Query :

```
SELECT `ProductName`, `Price`, CEIL(`Price`) AS `CeiledPrice`  
FROM `sql_class_3_assignment`.`products`
```

ProductName	Price	CeiledPrice
Shoes	75	75
Headphones	30.2	31
TV	450.6	451
Smartphone	300	300
Smartwatch	150	150
Laptop	850	850
Speaker	100.4	101
Desk	200	200
Chair	75	75
Tablet	400	400

Question 3: Calculate the floor (largest integer less than or equal to) of the prices.

Query :

```
-- Solution  
SELECT `ProductName`, `Price`, FLOOR(`Price`) AS `FlooredPrice`  
FROM `sql_class_3_assignment`.`products`;
```

Execution :

Question 3: Calculate the floor (largest integer less than or equal to) of the prices.

Query :

```
-- Solution
SELECT `ProductName`, `Price`, FLOOR(`Price`) AS `FlooredPrice`
FROM `sql_class_3_assignment`.`products`;
```

	ProductName	Price	FlooredPrice
1	Shoes	75	75
2	Headphones	30.2	30
3	TV	450.6	450
4	Smartphone	300	300
5	Smartwatch	150	150
6	Laptop	850	850
7	Speaker	100.4	100
8	Desk	200	200
9	Chair	75	75
10	Tablet	400	400

- Consider a table named Orders with the following columns:

OrderID (integer): The unique identifier for each order.

OrderDate (datetime): The date and time when the order was placed.

DeliveryDate (datetime): The date and time when the order was delivered.

Question 1: Find the difference in days between the order date and delivery date for each order.

let's view **orders** table

Query :

```
SELECT *
FROM `sql_class_3_assignment`.`orders`;
```

Execution :

Query :

```
SELECT *
FROM `sql_class_3_assignment`.`orders`;
```

Execution :

```
SELECT `OrderDate`, `DeliveryDate`, DATEDIFF(`DeliveryDate`, `OrderDate`) AS DelayInDeliveryInDays,
TIMEDIFF(`DeliveryDate`, `OrderDate`) AS DelayInHours
FROM `sql_class_3_assignment`.`orders`;
```

OrderID	ProductID	ProductName	Price	CustomerID	CustomerName	OrderDate	DeliveryDate
1	100001	Shoes	75	1001	John Doe	2022-01-01 08:00:00	2022-01-04 08:00:00
2	100002	Headphones	30	1005	Andrew	2022-01-02 12:00:00	2022-01-07 12:00:00
3	100003	TV	450	1004	Emily Davis	2022-01-03 10:00:00	2022-01-11 10:00:00
4	100004	Smartphone	300	1009	David Anderson	2022-01-04 12:00:00	2022-01-06 14:00:00
5	100005	Smartwatch	150	1006	Jessica Brown	2022-01-05 11:00:00	2022-01-05 15:00:00
6	100006	Laptop	850	1002	Jane Smith	2022-01-06 16:00:00	2022-01-09 16:00:00
7	100007	Speaker	100	1007	William Martinez	2022-01-07 09:00:00	2022-01-11 09:00:00
8	100008	Desk	200	1003	Michael Johnson	2022-01-08 08:00:00	2022-01-12 13:00:00
9	100009	Chair	75	1008	Olivia Garcia	2022-01-09 10:00:00	2022-01-13 17:00:00
10	100010	Tablet	400	1010	Sophia Lopez	2022-01-10 08:00:00	2022-01-14 08:00:00

Query :

```
SELECT `OrderDate`, `DeliveryDate`, DATEDIFF(`DeliveryDate`, `OrderDate`) AS DelayInDeliveryInDays,
TIMEDIFF(`DeliveryDate`, `OrderDate`) AS DelayInHours
FROM `sql_class_3_assignment`.`orders`;
```

Execution :

Query :

```
SELECT `OrderDate`, `DeliveryDate`, DATEDIFF(`DeliveryDate`, `OrderDate`) AS DelayInDeliveryInDays,
TIMEDIFF(`DeliveryDate`, `OrderDate`) AS DelayInHours
FROM `sql_class_3_assignment`.`orders`;
```

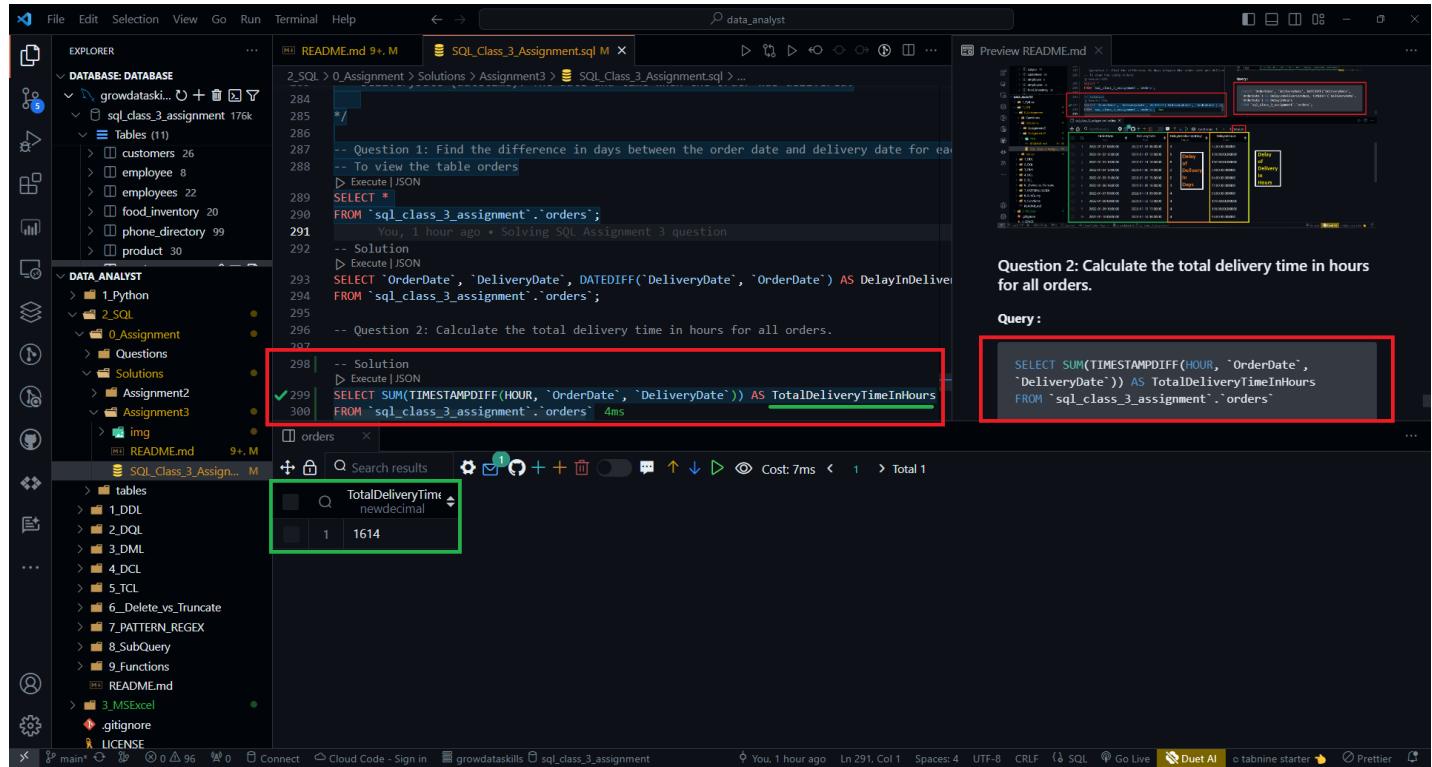
OrderDate	DeliveryDate	DelayInDeliveryInDays	DelayInHours
2022-01-01 08:00:00	2022-01-04 08:00:00	3	72:00:00.000000
2022-01-02 12:00:00	2022-01-07 12:00:00	5	120:00:00.000000
2022-01-03 10:00:00	2022-01-11 10:00:00	8	192:00:00.000000
2022-01-04 12:00:00	2022-01-06 14:00:00	2	50:00:00.000000
2022-01-05 11:00:00	2022-01-05 15:00:00	0	04:00:00.000000
2022-01-06 16:00:00	2022-01-09 16:00:00	3	72:00:00.000000
2022-01-07 09:00:00	2022-01-11 09:00:00	4	96:00:00.000000
2022-01-08 08:00:00	2022-01-12 13:00:00	4	101:00:00.000000
2022-01-09 10:00:00	2022-01-13 17:00:00	4	103:00:00.000000
2022-01-10 08:00:00	2022-01-14 08:00:00	4	96:00:00.000000

Question 2: Calculate the total delivery time in hours for all orders.

Query :

```
SELECT SUM(TIMESTAMPDIFF(HOUR, `OrderDate`, `DeliveryDate`)) AS TotalDeliveryTimeInHours
FROM `sql_class_3_assignment`.`orders`
```

Execution :



```
SELECT SUM(TIMESTAMPDIFF(HOUR, `OrderDate`, `DeliveryDate`)) AS TotalDeliveryTimeInHours
FROM `sql_class_3_assignment`.`orders`;
```

Question 3: Determine the day of the week when each order was placed.

let's solve this problem

Query :

```
SELECT `OrderID`, `ProductName`, DAYOFWEEK(`OrderDate`) AS DayOfWeek
FROM `sql_class_3_assignment`.`orders`;
```

Execution :

```

SELECT `OrderID`, `ProductName`, `OrderDate`
FROM `sql_class_3_assignment`.`orders`
WHERE DAYOFWEEK(`OrderDate`) = 7;

```

	OrderID	ProductName	DayOfWk
1	100001	Shoes	7
2	100002	Headphones	1
3	100003	TV	2
4	100004	Smartphone	3
5	100005	Smartwatch	4
6	100006	Laptop	5
7	100007	Speaker	6
8	100008	Desk	7
9	100009	Chair	-

Question 4: Find the orders that were placed on a Saturday (DayOfWeek = 7).

Query :

```

SELECT `OrderID`, `ProductName`, `OrderDate`
FROM `sql_class_3_assignment`.`orders`
WHERE DAYOFWEEK(`OrderDate`) = 7;

```

Execution :

```

SELECT `OrderID`, `ProductName`, `OrderDate`
FROM `sql_class_3_assignment`.`orders`
WHERE DAYOFWEEK(`OrderDate`) = 7;

```

	OrderID	ProductName	OrderDate
1	100001	Shoes	2022-01-01 08:00:00
2	100008	Desk	2022-01-08 08:00:00
3	100015	Printer	2022-01-15 14:00:00

Question 5: Calculate the average delivery time in days for all orders.

Query :

```
SELECT AVG(DATEDIFF(`DeliveryDate`, `OrderDate`)) AS AverageDeliveryTimeInDays
FROM `sql_class_3_assignment`.`orders`;
```

Execution :

The screenshot shows a SQL development environment with several tabs open. The main tab contains the SQL query for Question 5. The results pane shows a single row with the average delivery time. A red box highlights the query in the code editor, and a green box highlights the result table.

```
SELECT AVG(DATEDIFF(`DeliveryDate`, `OrderDate`)) AS AverageDeliveryTimeInDays
FROM `sql_class_3_assignment`.`orders`;
```

AverageDeliveryTimeInDays
3.3000

Question 6: Find the orders that were delivered on the same day they were placed.

Query :

```
SELECT `OrderId`, `ProductName`, `OrderDate`, DATEDIFF(`DeliveryDate`, `OrderDate`) AS
`DeliveryDelayInDays`
FROM `sql_class_3_assignment`.`orders`
WHERE DATEDIFF(`DeliveryDate`, `OrderDate`) = 0;
```

Execution :

```

SELECT `OrderID`, `ProductName`, `OrderDate`, DATEDIFF(`DeliveryDate`, `OrderDate`) AS `DeliveryDelayInDays`
FROM `sql_class_3_assignment`.`Orders`
WHERE DATEDIFF(`DeliveryDate`, `OrderDate`) = 0;

```

Execution :

let's solve this problem

Query :

```

SELECT `OrderID`, `ProductName`, `OrderDate`, DATEDIFF(`DeliveryDate`, `OrderDate`) AS `DeliveryDelayInDays`
FROM `sql_class_3_assignment`.`Orders`
WHERE DATEDIFF(`DeliveryDate`, `OrderDate`) = 0;

```

Q	OrderID	ProductName	OrderDate	DeliveryDelayInDays
1	100005	Smartwatch	2022-01-05 11:00:00	0
2	100015	Printer	2022-01-15 14:00:00	0
3	100019	USB Drive	2022-01-19 10:00:00	0

GROUP BY, HAVING, ORDER BY

Question 1: Consider a table named Orders with the following columns:

OrderID (integer): The unique identifier for each order.

CustomerID (integer): The unique identifier for each customer.

TotalAmount (decimal): The total amount of the order.

Write an SQL query to find the customer IDs of customers who have placed orders with a total amount greater than \$1,000.

Query :

```

SELECT `CustomerID`, `CustomerName`, `TotalAmount`
FROM `sql_class_3_assignment`.`orders`
GROUP BY `CustomerID`, `CustomerName`, `TotalAmount`
HAVING SUM(`TotalAmount`) > 1000;

```

Execution :

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

```

2.SQL > 0.Assignment > Assignment3 > SQL_Class_3_Assignment.sql M
332 OrderID (integer): The unique identifier for each order.
333 CustomerID (integer): The unique identifier for each customer.
334 TotalAmount (decimal): The total amount of the order.
335 Write an SQL query to find the customer IDs of customers who have placed orders with
336 a total amount greater than $1,000.
337 */
338
339 -- To view the table
340 > Execute JSON
341 SELECT * You, 1 minute ago + Uncommitted changes
FROM `sql_class_3_assignment`.`orders`;
342
343 -- Solution
344 > Execute JSON
345 SELECT `CustomerID`, `CustomerName`, `TotalAmount`
FROM `sql_class_3_assignment`.`orders`
GROUP BY `CustomerID`, `CustomerName`, `TotalAmount`
HAVING SUM(`TotalAmount`) > 1000; 4ms
346
347
348
349
350

```

The results of the query are displayed in a table:

CustomerID	CustomerName	TotalAmount
1	John Doe	1500
2	Emily Davis	1200
3	Jessica Brown	2000
4	Jane Smith	1350
5	William Martinez	1100
6	Olivia Garcia	1250
7	John Doe	1800
8	Emily Davis	1600
9	David Anderson	1050

The total count of rows is indicated as "Total 20".

Question 2: Consider a table named Sales with the following columns:

ProductID (integer): The unique identifier for each product.

SaleDate (date): The date of the sale.

QuantitySold (integer): The quantity of the product sold on that date.

Write an SQL query to find the product IDs of products that have been sold in quantities greater than 100 on at least three different sale dates.

Query :

```

SELECT `ProductID`, `ProductName`, `Category`
FROM `sql_class_3_assignment`.`sales`
GROUP BY `ProductID`, `ProductName`, `Category`
HAVING SUM(`QuantitySold`) > 100 AND COUNT(DISTINCT `SaleDate`) >= 3;

```

Execution :

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

```

SELECT `ProductID`, `ProductName`, `Category`
FROM `sql_class_3_assignment`.`sales`
GROUP BY `ProductID`, `ProductName`, `Category`
HAVING SUM(`QuantitySold`) > 100 AND COUNT(DISTINCT `SaleDate`) >= 3;

```

The results of this query are displayed in a table below, showing 16 rows of data from the 'sales' table.

ProductID	ProductName	Category
101	Shoes	Footwear
102	Headphones	Electronics
103	Shirt	Apparel
104	Trousers	Apparel
105	Watch	Accessories
106	Laptop	Electronics
108	Smartphone	Electronics
109	Desk	Office Furniture
110	Chair	Office Furniture
111	Printer	Electronics
112	Monitor	Electronics
113	Keyboard	Accessories
114	Mouse	Accessories
115	Sofa	Home Goods
116	Bed	Home Goods
117	Couch	Home Goods
118	Chair	Home Goods

Question 3: Consider a table named Employees with the following columns:

EmployeeID (integer): The unique identifier for each employee.

Department (string): The department in which the employee works.

Salary (decimal): The salary of the employee.

Write an SQL query to find the average salary of employees in each department, but only for departments where the average salary is greater than \$60,000.

Query :

```

SELECT `Department`, AVG(Salary) AS `Average Salary`
FROM `sql_class_3_assignment`.`employee_s`
GROUP BY `Department`
HAVING AVG(Salary) > 60000

```

Execution :

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

```
/*
Question 3: Consider a table named Employees with the following columns:
EmployeeID (integer): The unique identifier for each employee.
Department (string): The department in which the employee works.
Salary (decimal): The salary of the employee.
Write an SQL query to find the average salary of employees in each department, but only for departments where the average salary is greater than $60,000.
*/
-- To view table
-- Execute | JSON
SELECT *
FROM `sql_class_3_assignment`.`employee_s`;
```

A red box highlights the section from line 382 to line 389. Below the editor, a results table is displayed:

Department	Average Salary	
1	Engineering	74000
2	Sales	61400
3	Marketing	64000
4	HR	60500

A green box highlights the results table. On the right side of the interface, there is a preview window showing the same SQL query and its results.

Question 4: Consider a table named Students with the following columns:

StudentID (integer): The unique identifier for each student.

Course (string): The course name.

Score (integer): The score obtained by the student in the course.

Write an SQL query to find the course names in which the average score of all students is greater than or equal to 80.

Query :

```
SELECT `Course`, AVG(Score) AS `Average Score`
FROM `sql_class_3_assignment`.`student_s`
GROUP BY `Course`
HAVING AVG(`Score`) >= 80;
```

Execution :

The screenshot shows a Visual Studio Code interface with several tabs open. The main tab contains an SQL query:

```

391 Question 4: Consider a table named Students with the following columns:
392 StudentID (integer): The unique identifier for each student.
393 Course (string): The course name.
394 Score (integer): The score obtained by the student in the course.
395 Write an SQL query to find the course names in which the average score of all students
396 is greater than or equal to 80.
397 */
398 You, 25 minutes ago • Solving SQL Assignment 3
399 -- To view table
400 ▶ Execute | JSON
401 SELECT *
402 FROM `sql_class_3_assignment`.`student_s`;
403 -- Solution
404 ▶ Execute | JSON
405 SELECT `Course`, AVG(`Score`) AS `Average Score`
406 FROM `sql_class_3_assignment`.`student_s`
407 GROUP BY `Course`
408 HAVING AVG(`Score`) >= 80; 1ms
409

```

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

	Course	Average Score
1	Mathematics	88.5000
2	Physics	82.8000
3	Biology	87.7500
4	Chemistry	83.0000

On the right side of the interface, there is a preview window for the README.md file, which contains the question and its solution.

Question 5: Consider a table named Employees with the following columns:

EmployeeID (integer): The unique identifier for each employee.

Department (string): The department in which the employee works.

Salary (decimal): The salary of the employee.

Write an SQL query to find the department with the highest average salary.

Query :

```
-- Solution
SELECT `Department`
FROM `sql_class_3_assignment`.`employee_s`
GROUP BY `Department`
ORDER BY AVG(`Salary`) DESC
LIMIT 1;
```

Execution :

The screenshot shows a Visual Studio Code interface with several panes. The Explorer pane on the left lists various projects and files, including 'DATA ANALYST' and 'SQL Class 3 Assignment'. The Editor pane at the top contains an SQL script named 'SQL_Class_3_Assignment.sql' with code related to an 'Employees' table. The Terminal pane at the bottom shows the execution of a query to find the department with the highest average salary, resulting in a single row for 'Engineering'. A red box highlights the query in the terminal.

```
EmployeeID (integer): The unique identifier for each employee.  
Department (string): The department in which the employee works.  
Salary (decimal): The salary of the employee.  
  
Write an SQL query to find the department with the highest average salary.  
  
Query :  
  
-- Solution  
SELECT `Department`  
FROM `sql_class_3_assignment`.`employee_s`  
GROUP BY `Department`  
ORDER BY AVG(`Salary`) DESC  
LIMIT 1;  
  
+-----+  
| Department |  
+-----+  
| Engineering |  
+-----+
```

Question 6: Consider a table named Sales with the following columns:

ProductID (integer): The unique identifier for each product.

SaleDate (date): The date of the sale.

QuantitySold (integer): The quantity of the product sold on that date.

Write an SQL query to find the product with the highest total quantity sold.

Query :

```
SELECT `ProductName`  
FROM `sql_class_3_assignment`.`sales`  
GROUP BY `ProductName`  
ORDER BY SUM(`QuantitySold`) DESC  
LIMIT 1;
```

Execution :

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

```

2.SQL > 0.Assignment > Solutions > Assignment3 > SQL_Class_3_Assignment.sql > ...
428 /*
429 Question 6: Consider a table named Sales with the following columns:
430 ProductID (integer): The unique identifier for each product.
431 SaleDate (date): The date of the sale.
432 QuantitySold (integer): The quantity of the product sold on that date.
433 Write an SQL query to find the product with the highest total quantity sold.
434 */
435 You, 41 minutes ago • Solving SQL Assignment 3
436 -- To view table
437 > Execute | JSON
438 SELECT *
439   FROM `sql_class_3_assignment`.`sales`;
440 -- Solution
441 > Execute | JSON | Copy
442 SELECT `ProductName`
443   FROM `sql_class_3_assignment`.`sales`
444 GROUP BY `ProductName`
445 ORDER BY SUM(`QuantitySold`) DESC
446 LIMIT 1; 2ms

```

The results of the query are shown in a table:

ProductName	text
1	Laptop

A green arrow points from the word "Laptop" in the table to the word "PRODUCT" below it.

Execution :

```

SELECT `ProductName`
FROM `sql_class_3_assignment`.`sales`
GROUP BY `ProductName`
ORDER BY SUM(`QuantitySold`) DESC
LIMIT 1;

```

Activate Windows
Go to Settings to activate Windows.

Question 7: Consider a table named Students with the following columns:

StudentID (integer): The unique identifier for each student.

Course (string): The course name.

Score (integer): The score obtained by the student in the course.

Write an SQL query to find the top three students with the highest average score across all courses.

Query :

```

SELECT `StudentID`, `Course`, AVG(Score) AS `Average Score`
FROM `sql_class_3_assignment`.`student_s`
GROUP BY `StudentID`, `Course`
ORDER BY `Average Score` DESC
LIMIT 3;

```

Execution :

```

SELECT `StudentID`, `Course`, AVG(`Score`) AS `Average Score`
FROM `sql_class_3_assignment`.`student_s`
GROUP BY `StudentID`, `Course`
ORDER BY `Average Score` DESC
LIMIT 3;

```

	StudentID	Course	Average Score
1	5	Biology	95.0000
2	18	Mathematics	94.0000
3	7	Biology	92.0000

Question 8: Consider a table named Orders with the following columns:

OrderID (integer): The unique identifier for each order.

CustomerID (integer): The unique identifier for each customer.

TotalAmount (decimal): The total amount of the order.

Write an SQL query to find the total amount of orders placed by each customer, ordered in descending order of total amount.

Query :

```

SELECT `CustomerID`, `CustomerName`, SUM(`TotalAmount`) AS `TotalAmountOfOrders`
FROM `sql_class_3_assignment`.`orders`
GROUP BY `CustomerID`, `CustomerName`
ORDER BY `TotalAmountOfOrders` DESC;

```

Execution :

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

```

SELECT * 
FROM `sql_class_3_assignment`.`orders`;

```

Below the query, the results are displayed in a table:

	CustomerID	CustomerName	TotalAmountOfOrders
1	104	Olivia Garcia	2750
2	102	William Martinez	2650
3	102	Sophia Lopez	2250
4	103	Michael Johnson	2100
5	103	Jessica Brown	2000
6	103	David Anderson	2000
7	101	David Anderson	1930
8	103	John Doe	1800

The total number of rows is indicated as "Total 26".

Question 9: Consider a table named Books with the following columns:

BookID (integer): The unique identifier for each book.

Author (string): The author of the book.

PublicationYear (integer): The year the book was published.

Write an SQL query to find the number of books published by each author in descending order of the count.

Query :

```

SELECT `Author`, COUNT(`BookID`) AS `NumberOfBooks` 
FROM `sql_class_3_assignment`.`books` 
GROUP BY `Author` 
ORDER BY `NumberOfBooks` DESC;

```

Execution :

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

```

SELECT `Author`, COUNT(`BookID`) AS `NumberOfBooks`
FROM `sql_class_3_assignment`.`books`
GROUP BY `Author`
ORDER BY `NumberOfBooks` DESC;

```

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

	Author	NumberOfBooks
1	JK Rowling	4
2	Stephen King	4
3	Agatha Christie	4
4	JRR Tolkien	4
5	Dan Brown	4

Question 8: Consider a table named Orders with the following columns:

OrderID (integer): The unique identifier for each order.

CustomerID (integer): The unique identifier for each customer.

TotalAmount (decimal): The total amount of the order.

Write an SQL query to find the customer IDs of customers who have placed orders with a total amount greater than \$1,000 and have placed at least two orders.

Query :

```

SELECT `CustomerID`, `CustomerName`
FROM `sql_class_3_assignment`.`orders`
GROUP BY `CustomerID`, `CustomerName`
HAVING SUM(`TotalAmount`) > 1000 AND COUNT(`OrderID`) >= 2;

```

Execution :

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

```

2.SQL > 0.Assignment > Solutions > Assignment3 > SQL_Class_3_Assignment.sql M
507 CUSTOMERID (integer): the unique identifier for each customer.
508 OrderDate (date): The date of the order.
509 TotalAmount (decimal): The total amount of the order.
510 Write an SQL query to find the customer IDs of customers who have placed orders with
511 a total amount greater than $1,000 and have placed at least two orders.
512 */
513
514 -- To view TABLE You, 59 seconds ago * Uncommitted changes
515 > Execute JSON
516 SELECT *
517 FROM `sql_class_3_assignment`.`orders`;
518
519 -- Solution
520 > Execute JSON
521 SELECT `CustomerID`, `CustomerName`
522 FROM `sql_class_3_assignment`.`orders`
523 GROUP BY `CustomerID`, `CustomerName`
524 HAVING SUM(`TotalAmount`) > 1000 AND COUNT(`OrderID`) >= 2;

```

A red box highlights the solution query. Below it, a table titled "sql_class_3_assignment.orders" is displayed:

	CustomerID	CustomerName
1	101	David Anderson
2	102	William Martinez
3	104	Olivia Garcia
4	102	Sophia Lopez
5	105	John Doe

The results pane on the right shows the output of the query:

```

SELECT `CustomerID`, `CustomerName`
FROM `sql_class_3_assignment`.`orders`
GROUP BY `CustomerID`, `CustomerName`
HAVING SUM(`TotalAmount`) > 1000 AND COUNT(`OrderID`) >=
2;

```

Question 11: Consider a table named Products with the following columns:

ProductID (integer): The unique identifier for each product.

Category (string): The category of the product.

Price (decimal): The price of the product.

Write an SQL query to find the average price of products in each category, ordered by category name in ascending order.

Query :

```

SELECT `Category`, AVG(`Price`) AS AveragePrice
FROM `sql_class_3_assignment`.`product_s`
GROUP BY `Category`
ORDER BY `Category`;

```

Execution :

The screenshot shows a Visual Studio Code interface with several windows open. In the center, a code editor displays an SQL query:

```

2.SQL > 0.Assignment > Solutions > Assignment3 > SQL_Class_3_Assignment.sql M ...
527 Category (string): The category of the product.
528 Price (decimal): The price of the product.
529 Write an SQL query to find the average price of products in each category, ordered by
category name in ascending order.
531 */
533 -- to view TABLE
534 D Execute | JSON
535 SELECT *
536 FROM `sql_class_3_assignment`.`product_s`;
537 -- Solution
538 D Execute | JSON
539 SELECT `Category` ,AVG(`Price`) AS AveragePrice
540 FROM `sql_class_3_assignment`.`product_s`
541 GROUP BY `Category`
542 ORDER BY `Category`;

```

A red box highlights the solution part of the query. To the right, a preview window shows the results of the query:

Query :

```

SELECT `Category` ,AVG(`Price`) AS AveragePrice
FROM `sql_class_3_assignment`.`product_s`
GROUP BY `Category`
ORDER BY `Category`;

```

Execution :

Category	AveragePrice
Accessories	30.75
Clothing	57.5
Electronics	612.5

Question 12: Consider a table named Employees with the following columns:

EmployeeID (integer): The unique identifier for each employee.

Department (string): The department in which the employee works.

Salary (decimal): The salary of the employee.

Write an SQL query to find the department(s) with the lowest average salary.

Query :

```

-- Solution
SELECT `Department` ,AVG(`Salary`) AS AverageSalary
FROM `sql_class_3_assignment`.`employee_s`
GROUP BY `Department`
ORDER BY AVG(`Salary`) ASC;

```

Execution :

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

```
SELECT `Department`, AVG(`Salary`) AS AverageSalary
FROM `sql_class_3_assignment`.`employee_s`
GROUP BY `Department`
ORDER BY AVG(`Salary`) ASC;
```

A red box highlights the solution part of the code. Below the code, a green box highlights the resulting table from the query:

	Department	AverageSalary
1	HR	60500
2	Sales	61400
3	Marketing	64000
4	Engineering	74000

Question 13: Consider a table named Orders with the following columns:

OrderID (integer): The unique identifier for each order.

CustomerID (integer): The unique identifier for each customer.

OrderDate (date): The date of the order.

TotalAmount (decimal): The total amount of the order.

Write an SQL query to find the customer IDs of customers who have placed orders with a total amount greater than \$500 in the year 2023, ordered by customer ID in ascending order.

Query :

```
SELECT `CustomerID`, `CustomerName`, YEAR(`OrderDate`) AS `RequiredYear`
FROM `sql_class_3_assignment`.`orders`
GROUP BY `CustomerID`, `CustomerName`, `RequiredYear`
HAVING SUM(`TotalAmount`) > 500 AND RequiredYear = 2023;
```

Execution :

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

```

565 TotalAmount (decimal): The total amount of the order.
566 Write an SQL query to find the customer IDs of customers who have placed orders with
567 a total amount greater than $500 in the year 2023, ordered by customer ID in ascending
568 order.
569 */
570 -- To view table
571 > Execute | JSON
572 SELECT *
573 FROM `sql_class_3_assignment`.`orders`;
574
575 -- Solution
576 > Execute | JSON
577 SELECT `CustomerID`, `CustomerName`, YEAR(`OrderDate`) AS `RequiredYear`
578 FROM `sql_class_3_assignment`.`orders`
579 GROUP BY `CustomerID`, `CustomerName`, `RequiredYear`
580 HAVING SUM(`TotalAmount`) > 500 AND RequiredYear = 2023; 2ms

```

The results of the query are displayed in a table:

CustomerID	CustomerName	RequiredYear
1	Sophia Lopez	2023
2	John Doe	2023
3	Michael Johnson	2023

The right side of the interface shows the output of the query, which is identical to the results table.

Question 14: Consider a table named Students with the following columns:

StudentID (integer): The unique identifier for each student.

Course (string): The course name.

Score (integer): The score obtained by the student in the course.

Write an SQL query to find the course names in which the highest score achieved by any student is greater than or equal to 90, ordered by course name in ascending order

Query :

```

SELECT `Course`, MAX(`Score`) AS `max_score`
FROM `sql_class_3_assignment`.`student_s`
GROUP BY `Course`
HAVING `max_score` >= 90
ORDER BY `Course` ASC;

```

Execution :

```

585 Score (integer): The score obtained by the student in the course.
586 Write an SQL query to find the course names in which the highest score achieved by
587 any student is greater than or equal to 90, ordered by course name in ascending order.
588 *
589 -- To view table
590 > Execute|JSON
591 SELECT *
592 FROM `sql_class_3_assignment`.`student_s`;
593
594 -- Solution
595 > Execute|JSON|Copy
596 SELECT `Course` , MAX(`Score`) AS `max_score`
597 FROM `sql_class_3_assignment`.`student_s`
598 GROUP BY `Course`
599 HAVING `max_score` >= 90
600 ORDER BY `Course` ASC; 4ms

```

Course	max_score
Biology	95
Chemistry	90
Mathematics	94

Question 15: Consider a table named Orders with the following columns:

OrderID (integer): The unique identifier for each order.

CustomerID (integer): The unique identifier for each customer.

OrderDate (date): The date of the order.

TotalAmount (decimal): The total amount of the order.

Write an SQL query to find the customer IDs of customers who have placed orders with a total amount greater than \$500 in the year 2023 and have placed at least two orders

in that year. **Query :**

```

SELECT `CustomerID` , `ProductName` 
FROM `sql_class_3_assignment`.`orders` 
WHERE EXTRACT(YEAR FROM `OrderDate` )=2023 
GROUP BY `CustomerID` , `ProductName` 
HAVING SUM(`TotalAmount` ) > 500 AND COUNT(`OrderID` ) >= 2;

```

Execution :

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

```

609 a total amount greater than $500 in the year 2023 and have placed at least two orders
610 in that year.
611 */
612
613 -- to view table
614
615
616
617
618 -- Solution
619
620
621
622
623
624

```

The query is:

```

SELECT * FROM `sql_class_3_assignment`.`orders`

```

Below this, another query is shown:

```

SELECT `CustomerID`, `ProductName`
FROM `sql_class_3_assignment`.`orders`
WHERE EXTRACT(YEAR FROM `OrderDate`)=2023
GROUP BY `CustomerID`, `ProductName`
HAVING SUM(`TotalAmount`) > 500 AND COUNT(`OrderID`) >= 2;

```

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

CustomerID	ProductName
1	105
	Smartwatch

Question 16: Consider a table named Students with the following columns:

StudentID (integer): The unique identifier for each student.

Course (string): The course name.

Score (integer): The score obtained by the student in the course.

Write an SQL query to find the course names where the average score of students who scored less than 70 in at least one course is greater than or equal to 80.

Query :

```

SELECT `Course`
FROM `sql_class_3_assignment`.`student_s`
GROUP BY `Course`
HAVING COUNT(CASE WHEN Score < 70 THEN 1 ELSE NULL END) > 0 AND AVG(Score) >= 80;

```

Execution :

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

```

2.SQL > 0.Assignment > Solutions > Assignment3 > SQL_Class_3_Assignment.sql ...
628 Course (string): The course name.
629 Score (integer): The score obtained by the student in the course.
630 Write an SQL query to find the course names where the average score of students who
631 scored less than 70 in at least one course is greater than or equal to 80.
632 */
633
634 -- To view table
635 D Execute | JSON
636 SELECT *
637 FROM `sql_class_3_assignment`.`student_s`;
638 You, 7 seconds ago + Uncommitted changes
639 -- Solution
640 D Execute | JSON
641 SELECT "Course"
642 FROM `sql_class_3_assignment`.`student_s`
643 GROUP BY "Course"
644 HAVING COUNT(CASE WHEN Score < 70 THEN 1 ELSE NULL END) > 0 AND AVG(Score) >= 80; 2ms
645 /*

```

A red box highlights the solution part of the query. Below the editor, a search results panel shows a single result:

Course
Mathematics

The results panel has a green border.

Question 17: Consider a table named Employees with the following columns:

EmployeeID (integer): The unique identifier for each employee.

Department (string): The department in which the employee works.

Salary (decimal): The salary of the employee.

Write an SQL query to find the departments where the highest salary is greater than \$80,000 and the total number of employees in that department is at least 5.

Query :

```

SELECT `Department`
FROM `sql_class_3_assignment`.`employee_s`
GROUP BY `Department`
HAVING MAX(`Salary`) > 80000 AND COUNT(`EmployeeID`) >= 5;

```

Execution :

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

```
llass_3_assignment.sql > SELECT `Department` FROM `sql_class_3_assignment`.`employee_s` GROUP BY `Department` HAVING MAX(`Salary`) > 80000 AND COUNT(`EmployeeID`) >= 5;
```

The query is highlighted with a red box. Below the code editor, there is a search results panel with a single result:

Department
Engineering

A green box highlights the search results panel.

On the right side of the screen, there is a preview window for 'README.md' and a 'Query' section with the following text:

```
SELECT `Department`  
FROM `sql_class_3_assignment`.`employee_s`  
GROUP BY `Department`  
HAVING MAX(`Salary`) > 80000 AND COUNT(`EmployeeID`) >= 5;
```

Question 18: Consider a table named Products with the following columns:

ProductID (integer): The unique identifier for each product.

Category (string): The category of the product.

Price (decimal): The price of the product.

Write an SQL query to find the categories where the average price of products is greater than or equal to 50, and the maximum price within that category is greater than 100.

Query :

```
-- Solution  
SELECT `Category`  
FROM `sql_class_3_assignment`.`product_s`  
GROUP BY `Category`  
HAVING AVG(`Price`) >= 50 AND MAX(`Price`) > 100;
```

Execution :

The screenshot shows a code editor interface with several tabs and panes. On the left, there's a sidebar with icons for file operations like 'File', 'Edit', 'Selection', 'View', 'Go', 'Run', 'Terminal', and 'Help'. Below this is a 'DATABASE' tree view showing 'DATA BASE D...', 'growing...', 'sql_class_3_assignment...', and 'Tables (18)' which include 'product', 'product_s', 'products', 'sale', and 'sales'. To the right of the tree is a main code editor pane with a tab titled 'SQL_Class_3_Assignment.sql M'. The code in this tab is:

```
665 Price (decimal): The price of the product.
666 Write an SQL query to find the categories where the average price of products is
667 greater than or equal to $50, and the maximum price within that category is greater
668 than $100.
669 */
670
671 -- To view table
672 > Execute | JSON
673 SELECT *
674 FROM `sql_class_3_assignment`.`product_s`
675
676 -- Solution
677 > Execute | JSON
678 SELECT `Category`
679 FROM `sql_class_3_assignment`.`product_s`
680 GROUP BY `Category`
681 HAVING AVG(`Price`) >= 50 AND MAX(`Price`) > 100; 2ms
```

Below the code editor is a search results panel with a table titled 'Category' containing one row: '1 Electronics'. A red box highlights the entire code area from line 665 to line 680. To the right of the code editor is a preview pane with the title 'Preview README.md X'. It contains the following text:

Category (string): The category of the product.
Price (decimal): The price of the product.

Write an SQL query to find the categories where the average price of products is greater than or equal to \$50, and the maximum price within that category is greater than \$100.

Query :

```
-- Solution
SELECT `Category`
FROM `sql_class_3_assignment`.`product_s`
GROUP BY `Category`
HAVING AVG(`Price`) >= 50 AND MAX(`Price`) > 100;
```

Question 19: Consider a table named Orders with the following columns:

OrderID (integer): The unique identifier for each order.

CustomerID (integer): The unique identifier for each customer.

OrderDate (date): The date of the order.

TotalAmount (decimal): The total amount of the order.

Write an SQL query to find the customer IDs of customers who have placed orders with a total amount greater than \$1,000 in any single order and have placed orders on at least three different dates.

Query :

```
-- Solution
SELECT `CustomerID`
FROM `sql_class_3_assignment`.`orders`
GROUP BY `CustomerID`
HAVING MAX(`TotalAmount`) AND COUNT(DISTINCT `OrderDate`) >= 3;
```

Execution :

The screenshot shows a code editor interface with several tabs and panes. The top navigation bar includes File, Edit, Selection, View, Go, Run, Terminal, Help, and a preview pane for 'README.md'. The left sidebar has sections for EXPLORER, DATABASE, and DATA ANALYST. The main area contains two tabs: 'SQL_Class_3_Assignment.sql' and 'SQL_Class_3_Assignment.sql M'. The code editor displays an SQL query:`2.SQL > 0.Assignment > Solutions > Assignment3 > SQL_Class_3_Assignment.sql > ...
688 TotalAmount (decimal): The total amount of the order.
689 Write an SQL query to find the customer IDs of customers who have placed orders with
690 a total amount greater than $1,000 in any single order and have placed orders on at
691 least three different dates.
692 */
693 -- To view table
694 D Execute | JSON
695 SELECT *
696 FROM `sql_class_3_assignment`.`orders`;
697
698 -- Solution
699 D Execute | JSON
SELECT `CustomerID` You, 4 seconds ago * Uncommitted changes
FROM `sql_class_3_assignment`.`orders`
GROUP BY `CustomerID`
HAVING MAX(`TotalAmount`) AND COUNT(DISTINCT `OrderDate`) >= 3; 1ms`

A red box highlights the solution part of the code. Below the code editor is a results pane titled 'sql.class_3_assignment.orders' showing a table with CustomerID and OrderID. A green box highlights this table.

CustomerID	OrderID
1	101
2	102
3	103
4	104
5	105
6	108

At the bottom, there are status bars for 'Cost: 1ms', 'Total 6', and file details like 'You, 4 seconds ago', 'Ln 699, Col 20', 'Spaces: 4', 'UTF-8', 'CRLF', 'SQL', 'Go Live', 'Duet AI', 'tabnine starter', and 'Prettier'.

Submitted By: Arpit Dubey
Email ID: aarpitdubey@gmail.com
LinkedIn: [click here](#)
Github : [click here](#)
