

ASSIGNMENT-7

REGEXP

1) Match beginning of string(^)

The screenshot shows the MySQL Workbench interface with the following details:

- Schemas:** Local instance MySQL57, ecom
- Query Editor:** SQL File 11*, containing the following code:

```
342 -- Regular Expressions
343 -- Match beginning of string(^)
344 • select * from customers
345 where firstname regexp '^Sa';
```
- Result Grid:** Displays the results of the query, showing one row:

customerID	firstname	lastName	Email	address
4	Sarah	Brown	sarah@example.com	101 Pine St, Suburb
- Action Output:** Shows the execution log:

Time	Action	Message	Duration / Fetch
69 19:45:27	select * from customers where firstname regexp '^Sa' LIMIT 0, 1000	2 row(s) returned	0.000 sec / 0.000 sec
70 19:45:45	select * from customers where firstname regexp '[in]' LIMIT 0, 1000	2 row(s) returned	0.000 sec / 0.000 sec
71 19:49:10	select * from customers where firstname regexp 'Sa' LIMIT 0, 1000	1 row(s) returned	0.015 sec / 0.000 sec

2) Match end of string(\$)

The screenshot shows the MySQL Workbench interface with the following details:

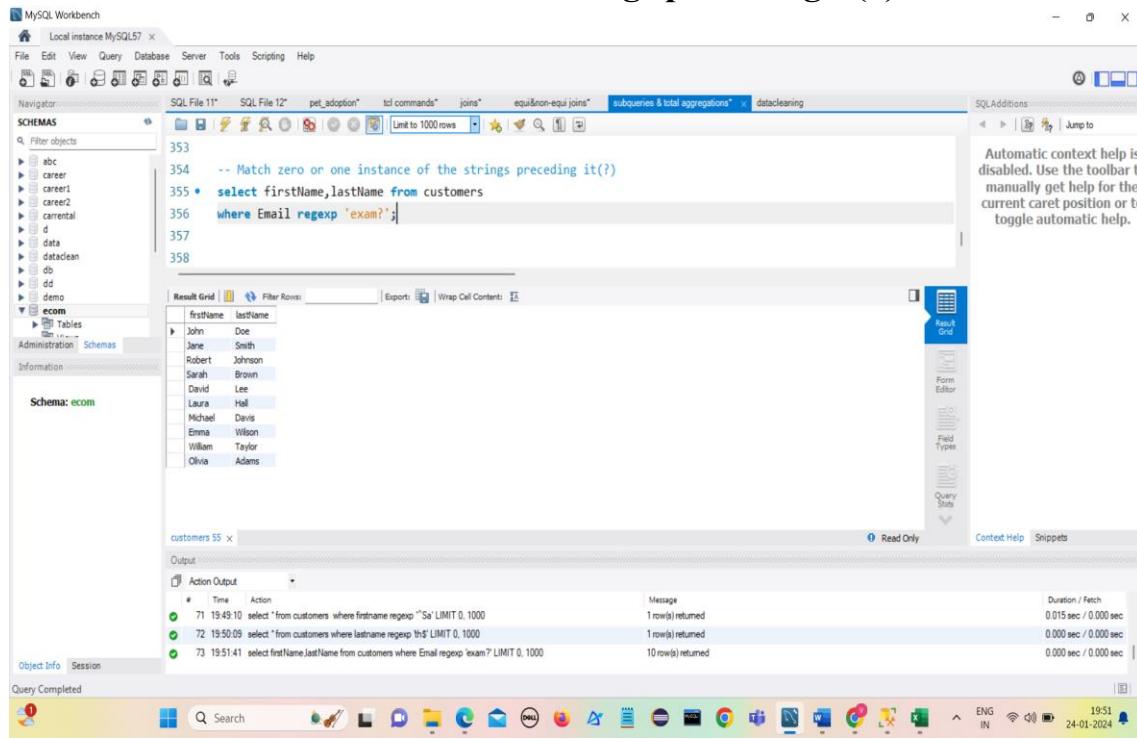
- Schemas:** Local instance MySQL57, ecom
- Query Editor:** SQL File 11*, containing the following code:

```
346 -- Match end of string($)
347 • select * from customers
348 where lastname regexp 'th$';
```
- Result Grid:** Displays the results of the query, showing one row:

customerID	firstname	lastName	Email	address
2	Jane	Smith	janesmith@example.com	456 Elm St, Town
- Action Output:** Shows the execution log:

Time	Action	Message	Duration / Fetch
70 19:45:45	select * from customers where firstname regexp '[in]' LIMIT 0, 1000	2 row(s) returned	0.000 sec / 0.000 sec
71 19:49:10	select * from customers where firstname regexp 'Sa' LIMIT 0, 1000	1 row(s) returned	0.015 sec / 0.000 sec
72 19:50:09	select * from customers where lastname regexp 'th\$' LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec

3) Match zero or one instance of the strings preceding it(?)



The screenshot shows the MySQL Workbench interface with a query editor window. The query being run is:

```
354 -- Match zero or one instance of the strings preceding it(?)
355 • select firstName,lastName from customers
356 where Email regexp 'exam?';
```

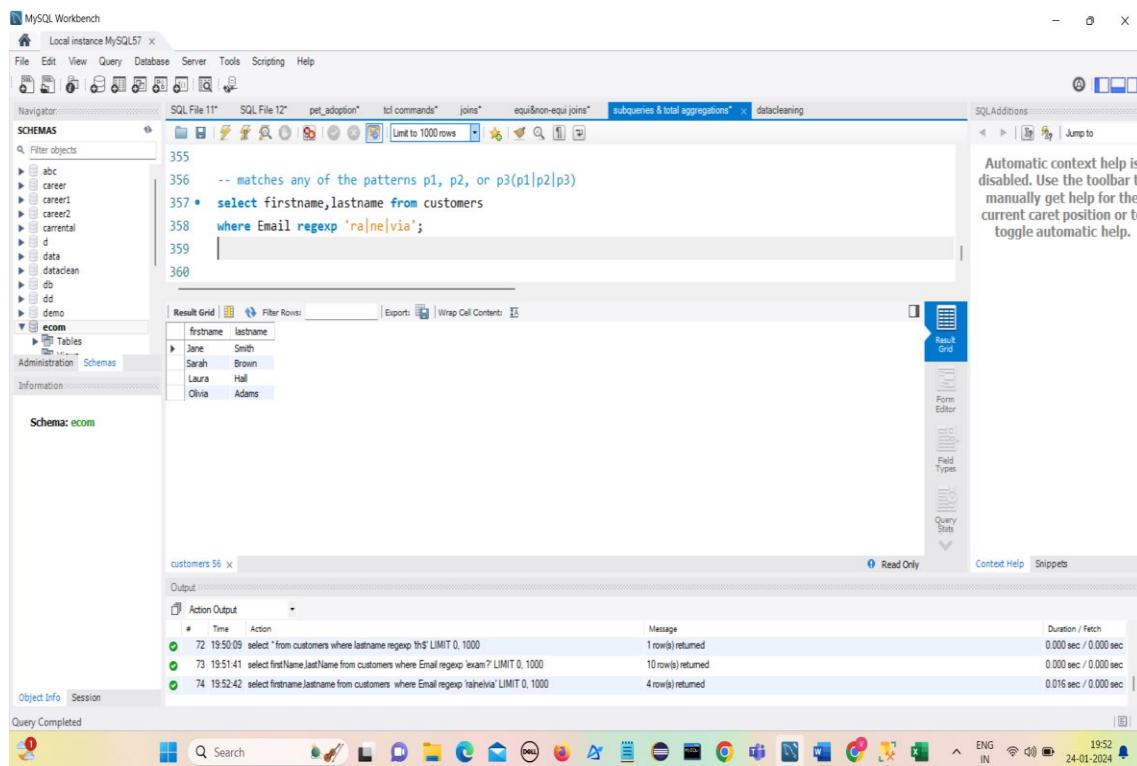
The result grid shows the following data:

firstName	lastName
John	Doe
Jane	Smith
Robert	Johnson
Sarah	Brown
David	Lee
Laura	Hall
Michael	Davis
Emma	Wilson
William	Taylor
Olivia	Adams

The output pane shows the execution log:

Action	Time	Message	Duration / Fetch
71	19:49:10	select * from customers where fname regexp 'Sa' LIMIT 0, 1000	1 row(s) returned 0.015 sec / 0.000 sec
72	19:50:09	select * from customers where lname regexp 'h\$' LIMIT 0, 1000	1 row(s) returned 0.000 sec / 0.000 sec
73	19:51:41	select FirstName,LastName from customers where Email regexp 'exam?' LIMIT 0, 1000	10 row(s) returned 0.000 sec / 0.000 sec

4) Matches any of the patterns p1, p2, or p3(p1|p2|p3)



The screenshot shows the MySQL Workbench interface with a query editor window. The query being run is:

```
355
356 -- matches any of the patterns p1, p2, or p3(p1|p2|p3)
357 • select firstname,lastname from customers
358 where Email regexp 'ra|ne|via';
```

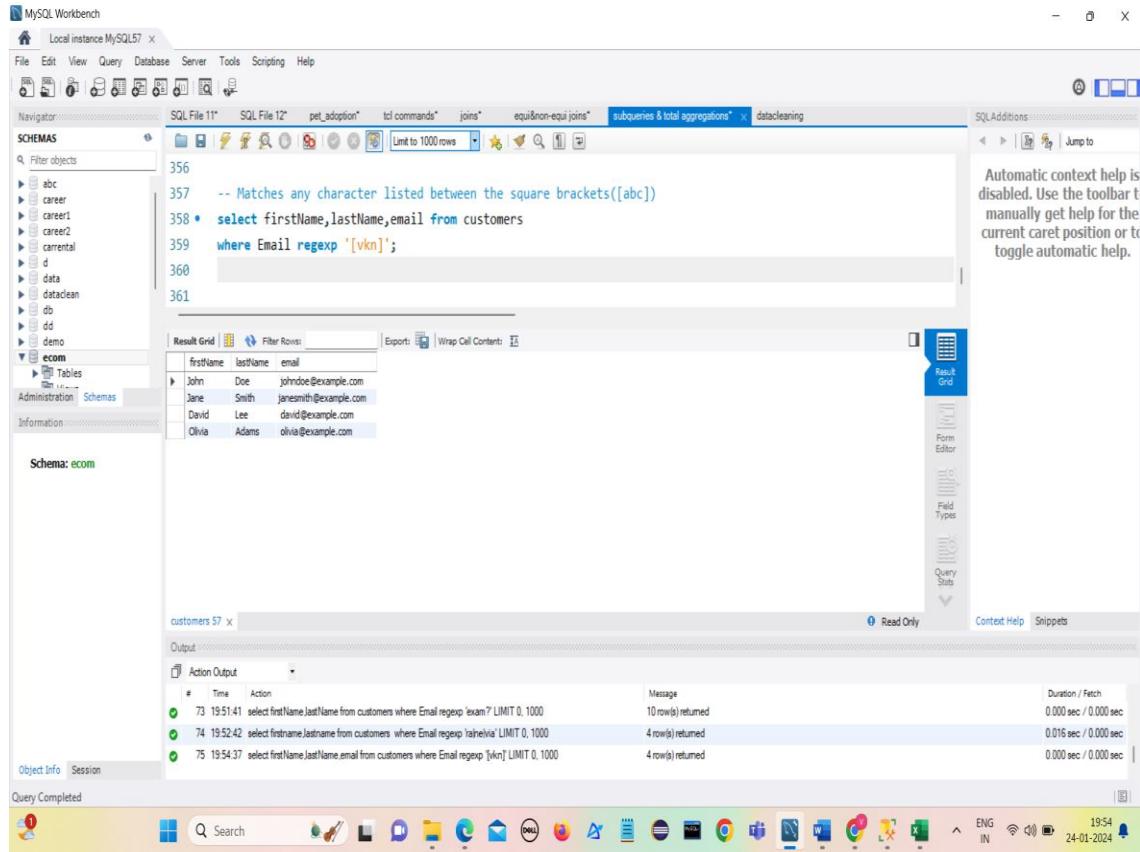
The result grid shows the following data:

firstname	lastname
Jane	Smith
Sarah	Brown
Laura	Hall
Olivia	Adams

The output pane shows the execution log:

Action	Time	Message	Duration / Fetch
72	19:50:09	select * from customers where lname regexp 'h\$' LIMIT 0, 1000	1 row(s) returned 0.000 sec / 0.000 sec
73	19:51:41	select FirstName,LastName from customers where Email regexp 'exam?' LIMIT 0, 1000	10 row(s) returned 0.000 sec / 0.000 sec
74	19:52:42	select firstname,lastname from customers where Email regexp 'ishevia' LIMIT 0, 1000	4 row(s) returned 0.016 sec / 0.000 sec

5) Matches any character listed between the square brackets([abc])



The screenshot shows the MySQL Workbench interface with the following details:

- Schemas:** Local instance MySQL57, ecom
- Query Editor:**

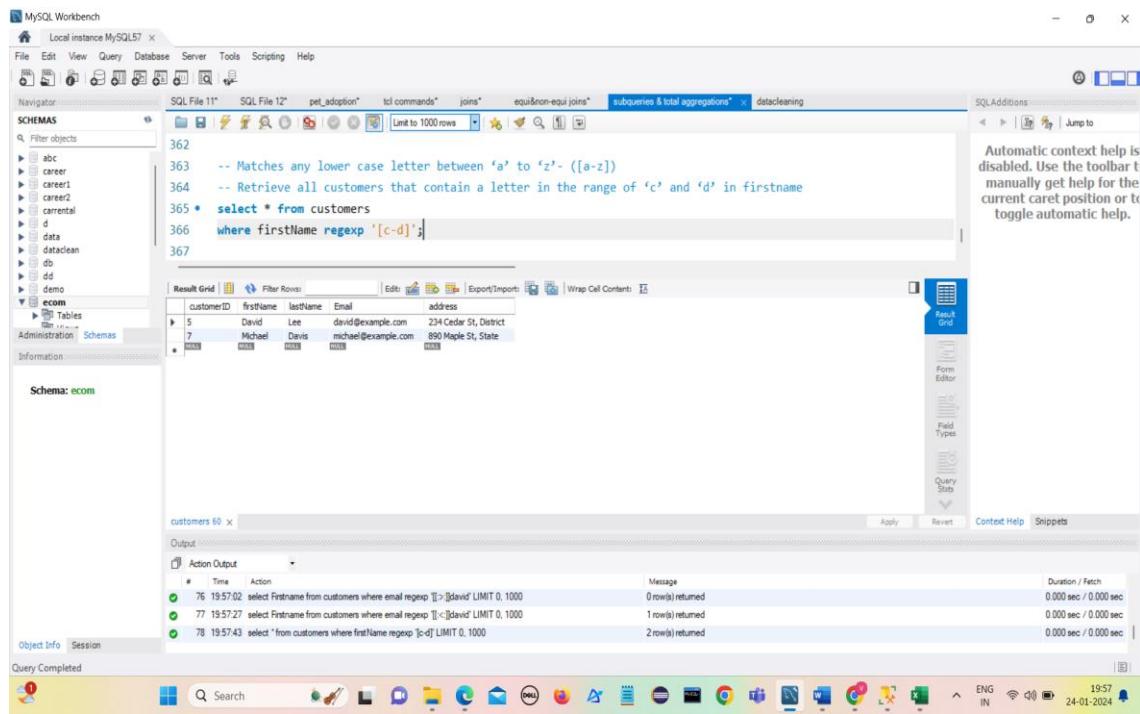
```

357 -- Matches any character listed between the square brackets([abc])
358 • select firstName,lastName,email from customers
359 where Email regexp '[vkn]';
360
361
    
```
- Result Grid:** Shows the results of the query, which are four rows of customer data:

firstName	lastName	email
John	Doe	john.doe@example.com
Jane	Smith	janesmith@example.com
David	Lee	david@example.com
Olivia	Adams	olivia@example.com
- Output:**
 - Action Output:
 - 73 19:51:41 select firstName,lastName from customers where Email regexp 'exam?' LIMIT 0, 1000 10 row(s) returned 0.000 sec / 0.000 sec
 - 74 19:52:42 select firstName,lastName from customers where Email regexp 'janevia' LIMIT 0, 1000 4 row(s) returned 0.016 sec / 0.000 sec
 - 75 19:54:37 select firstName,lastName,email from customers where Email regexp '[vkn]' LIMIT 0, 1000 4 row(s) returned 0.000 sec / 0.000 sec

6) Matches any lower case letter between ‘a’ to ‘z’- ([a-z])

Retrieve all customers that contain a letter in the range of ‘c’ and ‘d’ in firstname.



The screenshot shows the MySQL Workbench interface with the following details:

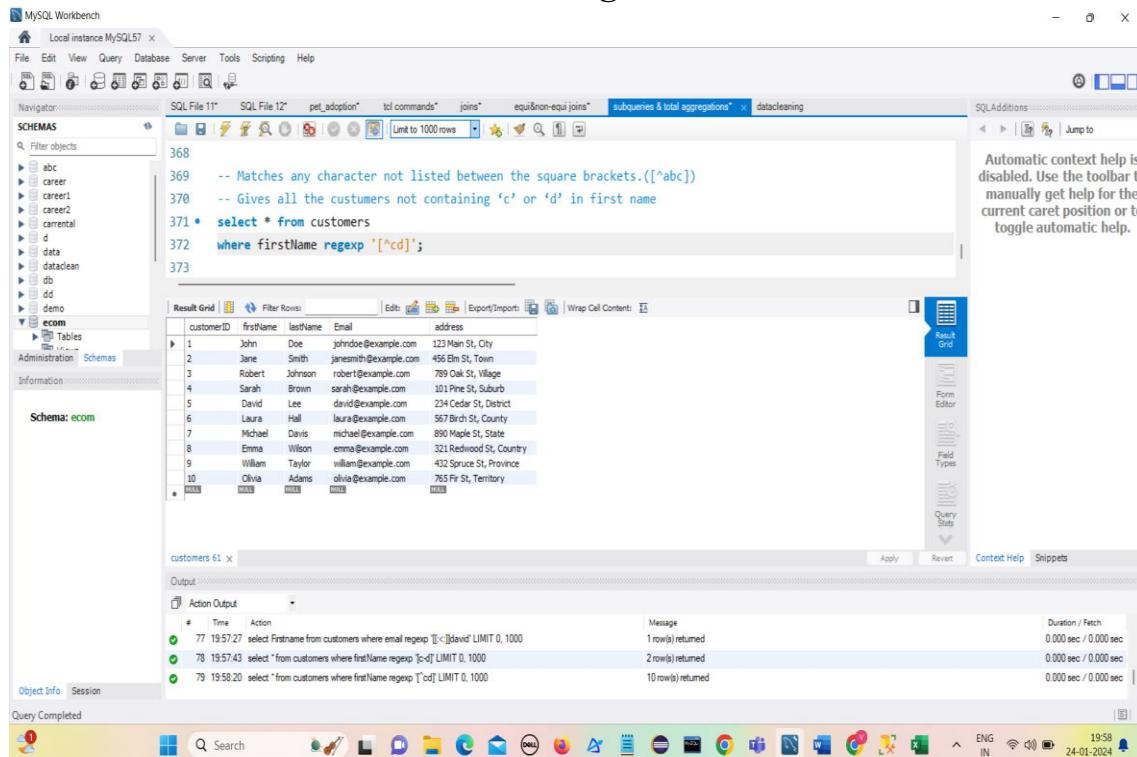
- Schemas:** Local instance MySQL57, ecom
- Query Editor:**

```

362
363 -- Matches any lower case letter between 'a' to 'z' - ([a-z])
364 -- Retrieve all customers that contain a letter in the range of 'c' and 'd' in firstname
365 • select * from customers
366 where FirstName regexp '[c-d]';
367
    
```
- Result Grid:** Shows the results of the query, which are two rows of customer data:

customerID	firstname	lastName	Email	address
5	David	Lee	david@example.com	234 Cedar St, District
7	Michael	Davis	michael@example.com	890 Maple St, State
- Output:**
 - Action Output:
 - 76 19:57:02 select FirstName from customers where email regexp 'd>[david]' LIMIT 0, 1000 0 row(s) returned 0.000 sec / 0.000 sec
 - 77 19:57:27 select FirstName from customers where email regexp 'l->[david]' LIMIT 0, 1000 1 row(s) returned 0.000 sec / 0.000 sec
 - 78 19:57:43 select * from customers where FirstName regexp '[c-d]' LIMIT 0, 1000 2 row(s) returned 0.000 sec / 0.000 sec

7) Matches any character not listed between the square brackets.([^abc])
Gives all the customers not containing ‘c’ or ‘d’ in first name

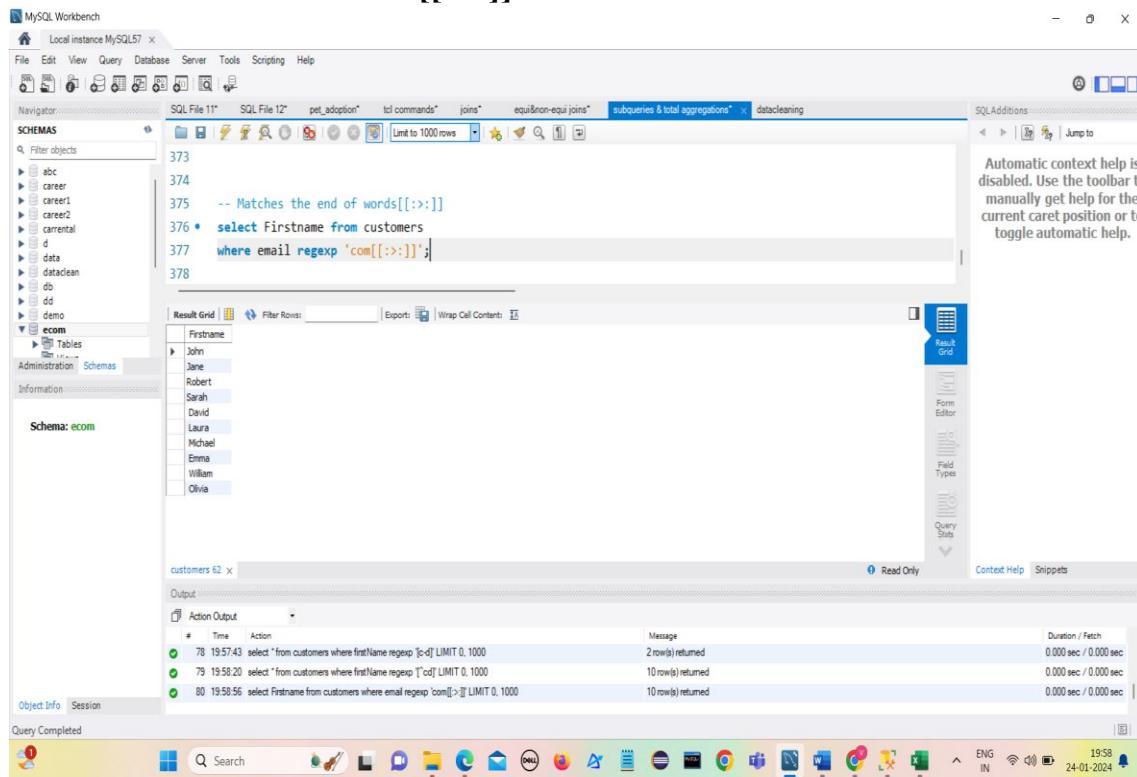


```

MySQL Workbench
File Edit View Query Database Server Tools Scripting Help
Navigator: SQL File 11* SQL File 12* pet_adoption* tcl commands* joins* equi&non-equi joins* subsequences & total aggregations* datacleaning
Schemas: abc career career1 career2 currental d data dataclean db dd demo ecom Tables Administration Schemas Information Schema: ecom
Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: |
customerID firstname lastName Email address
1 John Doe johndoe@example.com 123 Main St, City
2 Jane Smith janemsmith@example.com 456 Elm St, Town
3 Robert Johnson robert@example.com 789 Oak St, Village
4 Sarah Brown sarah@example.com 101 Pine St, Suburb
5 David Lee david@example.com 234 Cedar St, District
6 Laura Hall laura@example.com 567 Birch St, County
7 Michael Davis michael@example.com 890 Maple St, State
8 Emma Wilson emma@example.com 321 Redwood St, Country
9 William Taylor william@example.com 432 Spruce St, Province
10 Olivia Adams olivia@example.com 765 Fir St, Territory
customers 61 x
Output: Action Output
# Time Action Message Duration / Fetch
77 19:57:27 select * from customers where email regexp '[^<:;@]'; LIMIT 0, 1000 1 row(s) returned 0.000 sec / 0.000 sec
78 19:57:43 select * from customers where firstName regexp '[^cd]'; LIMIT 0, 1000 2 row(s) returned 0.000 sec / 0.000 sec
79 19:58:20 select * from customers where firstName regexp '[^cd]'; LIMIT 0, 1000 10 row(s) returned 0.000 sec / 0.000 sec
Object Info Session
Query Completed

```

8) Matches the end of words[[:>:]]



```

MySQL Workbench
File Edit View Query Database Server Tools Scripting Help
Navigator: SQL File 11* SQL File 12* pet_adoption* tcl commands* joins* equi&non-equi joins* subsequences & total aggregations* datacleaning
Schemas: abc career career1 career2 currental d data dataclean db dd demo ecom Tables Administration Schemas Information Schema: ecom
Result Grid | Filter Rows: | Export: | Wrap Cell Content: |
Firstname
John
Jane
Robert
Sarah
David
Laura
Michael
Emma
William
Olivia
customers 62 x
Output: Action Output
# Time Action Message Duration / Fetch
78 19:57:43 select * from customers where firstName regexp '[c-d]'; LIMIT 0, 1000 2 row(s) returned 0.000 sec / 0.000 sec
79 19:58:20 select * from customers where firstName regexp '[^cd]'; LIMIT 0, 1000 10 row(s) returned 0.000 sec / 0.000 sec
80 19:58:56 select FirstName from customers where email regexp 'com[[:>:]]'; LIMIT 0, 1000 10 row(s) returned 0.000 sec / 0.000 sec
Object Info Session
Query Completed

```

9) Matches the beginning of words[:<:]

The screenshot shows the MySQL Workbench interface with the following details:

- Schemas:** ecom
- Query Editor:** SQL File 11*
381 -- Matches the beginning of words[:<:]
382 • select Firstname from customers
383 where email regexp '[:<:]david';
384
- Result Grid:** Shows a single row with "Firstname" as "David".
- Action Output:** Displays three log entries for the executed queries.
- System Tray:** Shows the date and time as 24-01-2024.

10) Matches the beginning of all words by any character listed between the square brackets.(^abc):

Gives all the names starting with 'j' or 'm'.

The screenshot shows the MySQL Workbench interface with the following details:

- Schemas:** ecom
- Query Editor:** SQL File 11*
387 -- Matches the beginning of all words by any character listed between the square brackets.(^abc):
388 -- Gives all the names starting with 'j' or 'm'.
389 • select * from customers
390 where firstname regexp '^[jm]';
- Result Grid:** Shows two rows of customer data: John Doe and Jane Smith.
- Action Output:** Displays three log entries for the executed queries.
- System Tray:** Shows the date and time as 24-01-2024.

OVER AND PARTITION BY CLAUSE

1) Use of group by

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left lists the database structure, including tables like `employee` and `student`. The central pane contains a query window with the following SQL code:

```
-- use of group by
select age,avg(salary) as "Average Salary",sum(salary) as "Sum Salary",min(salary) as "Min Salary"
from employee
group by age;
```

The results pane displays the output of the query:

age	Average Salary	Sum Salary	Min Salary
20	35000	70000	30000
28	60000	60000	60000
30	31666	95000	20000
35	38750	150000	35000

At the bottom, a message indicates "Query executed successfully."

2) Partition by

SQL PARTITION BY clause with the OVER clause to specify the column on which we need to perform aggregation.

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left lists the database structure, including tables like `employee` and `student`. The central pane contains a query window with the following SQL code:

```
-- use of partition by
select age,avg(salary) over(partition by age) as "Average Salary",
sum(salary) over(partition by age) as "Sum Salary",
min(salary) over (partition by age) as "Min salary"
from employee;
```

The results pane displays the output of the query:

age	Average Salary	Sum Salary	Min salary
20	35000	70000	30000
20	35000	70000	30000
28	60000	60000	60000
30	31666	95000	20000
30	31666	95000	20000
35	38750	150000	35000
35	38750	150000	35000
35	38750	150000	35000

At the bottom, a message indicates "Query executed successfully."

We can add name in select with partition but we cannot add in group by

The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer, a database named 'dataclean' is selected. In the main query window, the following T-SQL code is executed:

```
-- use of partition by
select age, name, avg(salary) over(partition by age) as "Average Salary",
sum(salary) over(partition by age) as "Sum Salary",
min(salary) over (partition by age) as "Min salary"
from employee
```

The results grid displays 10 rows of data with columns: age, name, Average Salary, Sum Salary, and Min salary. The data is partitioned by age, showing the average salary for each age group.

	age	name	Average Salary	Sum Salary	Min salary
1	20	stella	35000	70000	30000
2	20	aparna	35000	70000	30000
3	28	lucky	60000	60000	60000
4	30	nimmi	31666	95000	20000
5	30	alex	31666	95000	20000
6	35	stella	38750	155000	35000
7	35	nani	38750	155000	35000
8	35	kishore	38750	155000	35000
9	35				
10	35				

Query executed successfully.

Total aggregations using over and partition clause

1) Partition by age and average of salary

The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer, a database named 'dataclean' is selected. In the main query window, the following T-SQL code is executed:

```
-- partition by age and average of salary
select * from employee
select id, name, age, email, avg(salary) over() as "AVG SALARY",
avg(salary) over(partition by age) as "AVG SALARY BY NAME"
from employee;
```

The results grid displays 10 rows of data with columns: id, name, age, email, AVG SALARY, and AVG SALARY BY NAME. The data is partitioned by age, showing the average salary for each age group.

	id	name	age	email	AVG SALARY	AVG SALARY BY NAME
1	105	stella	20	stella@gmail.com	38000	35000
2	110	aparna	20	aparna@gmail.com	38000	35000
3	112	lucky	28	lucky@gmail.com	38000	60000
4	108	nimmi	30	nimmi@gmail.com	38000	31666
5	109	krish	30	krish@gmail.com	38000	31666
6	101	alex	30	alex@gmail.com	38000	31666
7	104	stella	35	stella@gmail.com	38000	38750
8	106	stella	35	stella@gmail.com	38000	38750
9	107	nani	35	nani@gmail.com	38000	38750
10	111	kishore	35	kishore@gmail.com	38000	38750

--use group by

Query executed successfully.

2) Use of group by

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows the database structure for 'DESKTOP-HPGQ4S\SQLEXPRESS'. The central pane displays a T-SQL script:

```
--use group by  
--average and top salary for each employee  
select age,avg(salary) as "AVG Salary",MAX(salary) as "Top Salary"  
from employee  
group by age  
  
select * from employee;
```

The results pane shows the output of the query:

age	AVG Salary	Top Salary
1	20	35000
2	28	60000
3	30	31666
4	35	45000

A status bar at the bottom indicates 'Query executed successfully.' and the system tray shows the date and time as 24-01-2024.

3) Use partition by

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows the database structure for 'DESKTOP-HPGQ4S\SQLEXPRESS'. The central pane displays a T-SQL script:

```
group by age  
  
-- use partition by  
select age,avg(salary) over(partition by age) as "AVG SALARY",  
max(salary) over(partition by age) as "TOP SALARY"  
from employee  
  
select * from employee;  
-- create view  
create view Emp_details as
```

The results pane shows the output of the query:

age	AVG SALARY	TOP SALARY
1	20	35000
2	20	35000
3	28	60000
4	30	31666
5	30	31666
6	30	31666
7	35	45000
8	35	38750
9	35	38750
10	35	38750

A status bar at the bottom indicates 'Query executed successfully.' and the system tray shows the date and time as 24-01-2024.

Materialized views

- A materialized view is a duplicate data table created by combining data from multiple existing tables for faster data retrieval.
- Materialized views work by precomputing and storing the results of a specific query as a physical table in the database.
- The database performs the precomputation at regular intervals, or users can trigger it by specific events.
- The data in the materialized view is precomputed and stored, meaning that the results are already available without the need to recompute the query each time the view is accessed.

Creating a view:

The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer, the database 'dataclean' is selected. In the center pane, a query window contains the following T-SQL code:

```
select * from employee;
-- create view
create view Emp_details_as
select id,name,age from employee
where department='sales';

--execute view
select * from Emp_details;
```

The 'Results' tab shows the output of the query:

id	name	age
104	steve	35
106	steff	35
108	krish	33
110	lucky	20

A status bar at the bottom indicates 'Query executed successfully.'

Execute view:

select * from viewname;

create view which shows employees with salary higher than avg salary

The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer, the database 'dataclean' is selected. In the center pane, a query window contains the following T-SQL code:

```
-- create view which shows employees with salary higher than avg salary
create view Emp_Sal as
select id,name from employee
where salary > ( select avg(salary) from employee);

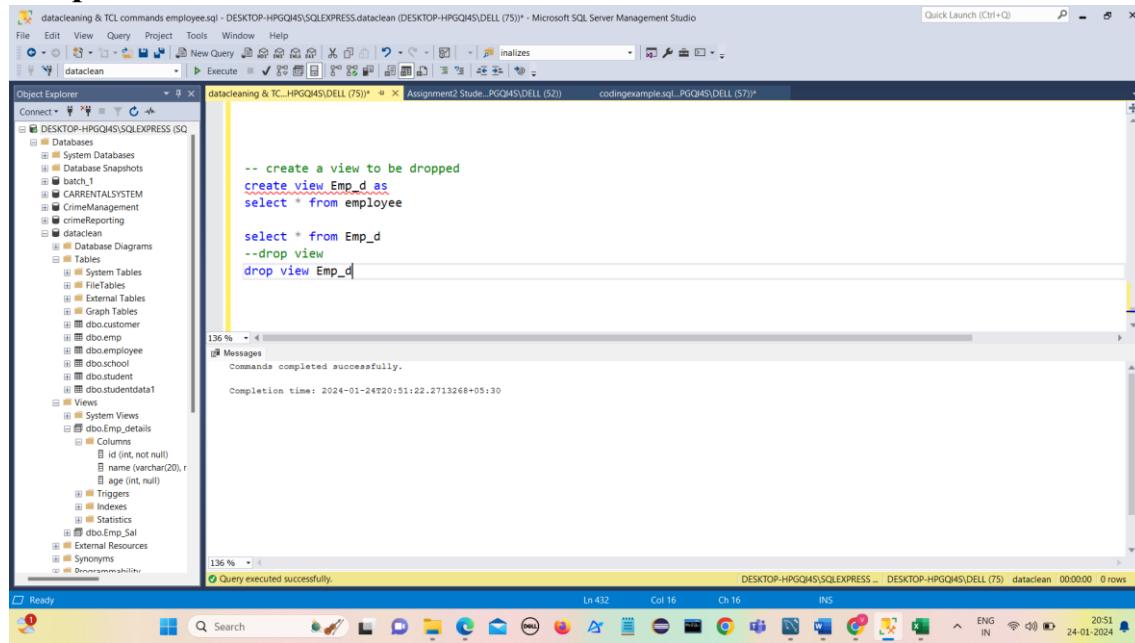
select * from Emp_Sal;
```

The 'Results' tab shows the output of the query:

id	name
105	steve
107	raju
109	krish
112	lucky

A status bar at the bottom indicates 'Query executed successfully.'

Drop view



The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer, the database 'dataclean' is selected. In the center pane, a query window displays the following T-SQL code:

```
-- create a view to be dropped
create view Emp_d as
select * from employee

select * from Emp_d
--drop view
drop view Emp_d
```

The status bar at the bottom indicates 'Query executed successfully.' and 'Completion time: 2024-01-24T20:51:22.2713268+05:30'. The taskbar at the bottom of the screen shows various application icons.

Star Schema:

A star schema consists of a central fact table that references multiple dimension tables. Each dimension table is denormalized ("flattened") to avoid the query overhead that comes with a highly normalized schema, which can require a large number of joins to retrieve the necessary data.

The fact table contains multiple foreign keys that reference the dimension tables. The data in each dimension table is denormalized, making queries fast and efficient.

For instance, the dimTerritory table includes the TerritoryName, TerritoryCountry and TerritoryRegion columns. Together, these columns form the hierarchy TerritoryRegion > TerritoryCountry > TerritoryName.

The more rows, the lower the cardinality(which refers to the number of unique values relative to the number of rows) and the more redundant data. Although data queries are generally faster in a star schema, the schema can also be more prone to data integrity issues and require more disk space than a more normalized structure.

Snowflake schema:

In a snowflake schema, a fact is surrounded by its associated dimensions (as in a star schema), but those dimensions are further related to other dimensions, branching out into a snowflake pattern. Snowflaking normalizes the dimensions by moving attributes with low cardinality into separate dimension tables.

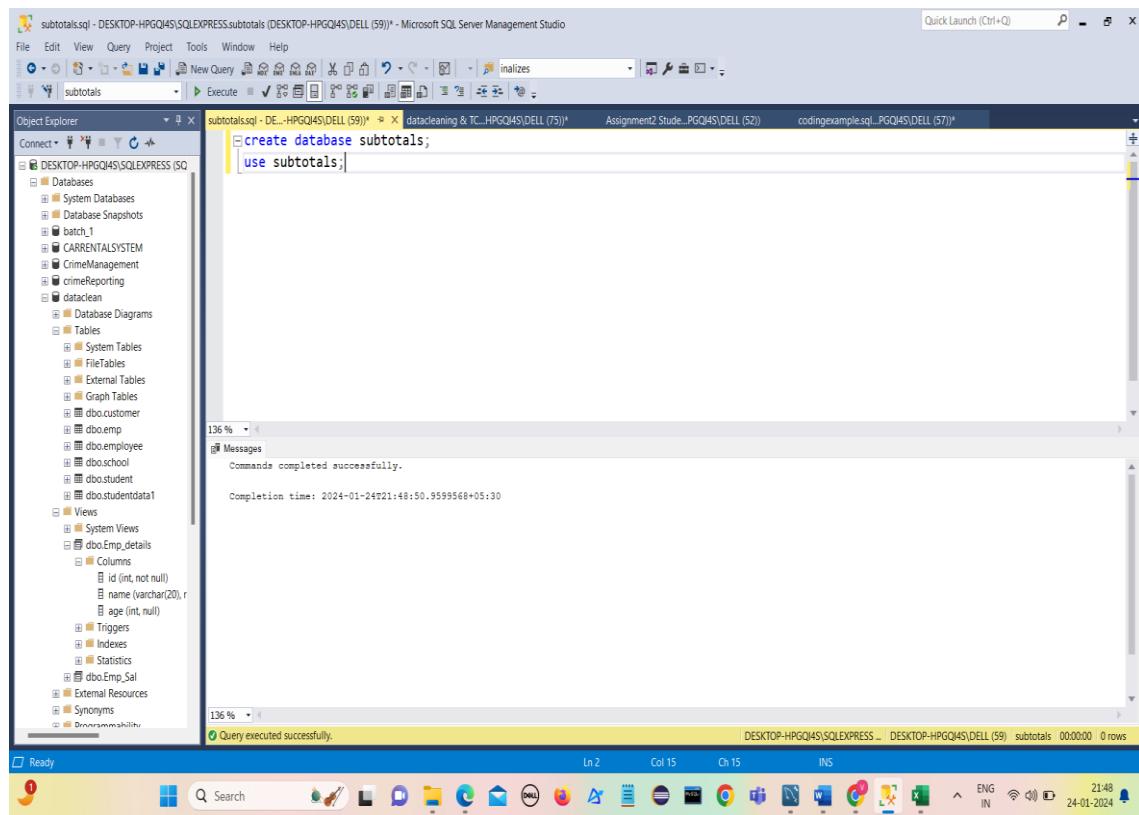
The fact table in a snowflake schema uses foreign keys to reference the core dimensions, which in turn use foreign keys to reference the dimensions at the next level.

The normalized dimensions in a snowflake schema reduce the amount of redundant data, making them less susceptible to data integrity issues than a star schema.

Sub Totals:

A subtotal is a figure that shows the sum of similar sets of data but it does not indicate the final total.

Calculating a subtotal in SQL query can be a bit complicated than the common aggregate queries



The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer on the left, a connection to 'DESKTOP-HPGQ4S\SQLEXPRESS' is selected, showing databases like 'batch_1', 'CARRENTALSYSTEM', 'CrimeManagement', 'crimeReporting', 'dataclean', 'datacleaner', 'dbo.customer', 'dbo.emp', 'dbo.employee', 'dbo.school', 'dbo.student', and 'dbo.studentdata1'. The 'Messages' pane at the bottom displays the command 'use subtotals;' and the message 'Commands completed successfully.' with a completion time of '2024-01-24T21:48:50.9599568+05:30'. The status bar at the bottom right shows 'Ready', 'Ln 2 Col 15 Ch 15 INS', and a date/time of '24-01-2024 21:48'.

```

CREATE TABLE SalesList
(
    SalesMonth NVARCHAR(20),
    SalesQuarters VARCHAR(5),
    SalesYear SMALLINT,
    SalesTotal MONEY
);
-- insert records into SalesList table
INSERT INTO SalesList(SalesMonth,SalesQuarters,SalesYear,SalesTotal) VALUES ('N'March','Q1',2019,60)
INSERT INTO SalesList(SalesMonth,SalesQuarters,SalesYear,SalesTotal) VALUES ('N'March','Q1',2020,50)
INSERT INTO SalesList(SalesMonth,SalesQuarters,SalesYear,SalesTotal) VALUES ('N'May','Q2',2019,30)
INSERT INTO SalesList(SalesMonth,SalesQuarters,SalesYear,SalesTotal) VALUES ('N'July','Q3',2020,10)
INSERT INTO SalesList(SalesMonth,SalesQuarters,SalesYear,SalesTotal) VALUES ('N'November','Q4',2019,120)
INSERT INTO SalesList(SalesMonth,SalesQuarters,SalesYear,SalesTotal) VALUES ('N'October','Q4',2019,150)
INSERT INTO SalesList(SalesMonth,SalesQuarters,SalesYear,SalesTotal) VALUES ('N'November','Q4',2019,180)
INSERT INTO SalesList(SalesMonth,SalesQuarters,SalesYear,SalesTotal) VALUES ('N'November','Q4',2020,120)
INSERT INTO SalesList(SalesMonth,SalesQuarters,SalesYear,SalesTotal) VALUES ('N'July','Q3',2019,160)
INSERT INTO SalesList(SalesMonth,SalesQuarters,SalesYear,SalesTotal) VALUES ('N'March','Q1',2020,170)

```

136 rows affected.

Query executed successfully.

```

CREATE TABLE SalesList
(
    SalesMonth NVARCHAR(20),
    SalesQuarters VARCHAR(5),
    SalesYear SMALLINT,
    SalesTotal MONEY
);
-- insert records into SalesList table
INSERT INTO SalesList(SalesMonth,SalesQuarters,SalesYear,SalesTotal) VALUES ('N'March','Q1',2019,60)
INSERT INTO SalesList(SalesMonth,SalesQuarters,SalesYear,SalesTotal) VALUES ('N'March','Q1',2020,50)
INSERT INTO SalesList(SalesMonth,SalesQuarters,SalesYear,SalesTotal) VALUES ('N'May','Q2',2019,30)
INSERT INTO SalesList(SalesMonth,SalesQuarters,SalesYear,SalesTotal) VALUES ('N'July','Q3',2020,10)
INSERT INTO SalesList(SalesMonth,SalesQuarters,SalesYear,SalesTotal) VALUES ('N'November','Q4',2019,120)
INSERT INTO SalesList(SalesMonth,SalesQuarters,SalesYear,SalesTotal) VALUES ('N'October','Q4',2019,150)
INSERT INTO SalesList(SalesMonth,SalesQuarters,SalesYear,SalesTotal) VALUES ('N'November','Q4',2019,180)
INSERT INTO SalesList(SalesMonth,SalesQuarters,SalesYear,SalesTotal) VALUES ('N'November','Q4',2020,120)
INSERT INTO SalesList(SalesMonth,SalesQuarters,SalesYear,SalesTotal) VALUES ('N'July','Q3',2019,160)
INSERT INTO SalesList(SalesMonth,SalesQuarters,SalesYear,SalesTotal) VALUES ('N'March','Q1',2020,170)

select * from SalesList;

```

SalesMonth	SalesQuarters	SalesYear	SalesTotal
1 March	Q1	2019	60.00
2 March	Q1	2020	50.00
3 May	Q2	2019	30.00
4 July	Q3	2020	10.00
5 November	Q4	2019	120.00
6 October	Q4	2019	150.00
7 November	Q4	2019	180.00
8 November	Q4	2020	120.00
9 July	Q3	2019	160.00
10 March	Q1	2020	170.00

136 rows affected.

Query executed successfully.

ROLLUP extension

ROLLUP, CUBE, and GROUPING SETS are extensions of the GROUP BY statement and add the extra subtotal and grand total rows to the resultset.

The **ROLLUP** extension allows us to generate hierarchical subtotal rows according to its input columns and it also adds a grand total row to the result set.

1)

The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer, the database 'DESKTOP-HPGQ4S\SQLEXPRESS.subtotals' is selected. In the center pane, a query window displays the following SQL code:

```
select * from SalesList;
--1)
SELECT SalesYear, SUM(SalesTotal) AS SalesTotal FROM SalesList
GROUP BY ROLLUP(SalesYear)
```

The results pane shows the output of the query:

SalesMonth	SalesQuarters	SalesYear	SalesTotal	
1	March	Q1	2019	60.00
2	March	Q1	2020	50.00
3	May	Q2	2019	30.00
4	July	Q3	2020	10.00
5	November	Q4	2019	120.00
6	October	Q4	2019	150.00
7	November	Q4	2019	180.00
8	November	Q4	2020	120.00
9	July	Q3	2019	160.00
10	March	Q1	2020	170.00

At the bottom of the results, there is a summary row:

SalesQuarters	SalesTotal
Q1	280.00
Q2	30.00
Q3	170.00
Q4	570.00
NULL	1050.00

A message at the bottom of the results pane says "Query executed successfully."

The extra row at the end of the result set, this row shows the grand total sales of the years without considering the sales month and quarters.

2)

The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer, the database 'DESKTOP-HPGQ4S\SQLEXPRESS.subtotals' is selected. In the center pane, a query window displays the following SQL code:

```
--2)
select SalesQuarters,sum(salesTotal) as SalesTotal from SalesList
group by rollup(salesQuarters);
```

The results pane shows the output of the query:

SalesQuarters	SalesTotal
Q1	280.00
Q2	30.00
Q3	170.00
Q4	570.00
NULL	1050.00

A message at the bottom of the results pane says "Query executed successfully."

3) pass two different columns as a parameter to the ROLLUP

```

subtotals.sql - DESKTOP-HPGQI4S\SQLEXPRESS.subtotals (DESKTOP-HPGQI4S\DELL (59)) - Microsoft SQL Server Management Studio
File Edit View Query Project Tools Window Help
New Query Execute
subtotals.sql : DE...HPGQI4S\DELL (59)* + datacleaning & TC_HPGQI4S\DELL (75)* Assignment2 Stude...PGQI4S\DELL (52)* codingexample.sql_PQI4S\DELL (57)*
--2)
select SalesQuarters,sum(salesTotal) as SalesTotal from SalesList
group by rollup(salesQuarters);

--3) PASS two different parameters to roll up
select SalesYear,SalesQuarters,sum(salestotal) as SalesTotal from SalesList
group by rollup(SalesYear,SalesQuarters);

Results Messages
SalesYear SalesQuarters SalesTotal
1 2019 Q1 60.00
2 2019 Q2 30.00
3 2019 Q3 160.00
4 2019 Q4 450.00
5 NULL 700.00
6 2020 Q1 220.00
7 2020 Q3 10.00
8 2020 Q4 120.00
9 2020 NULL 350.00
10 NULL NULL 1050.00
Query executed successfully.

```

700 and 350 are sub totals by years and 1050 is the grand total.

pass 3 columns into the ROLLUP extension and then this extension will generate subtotal rows for all hierarchies.

```

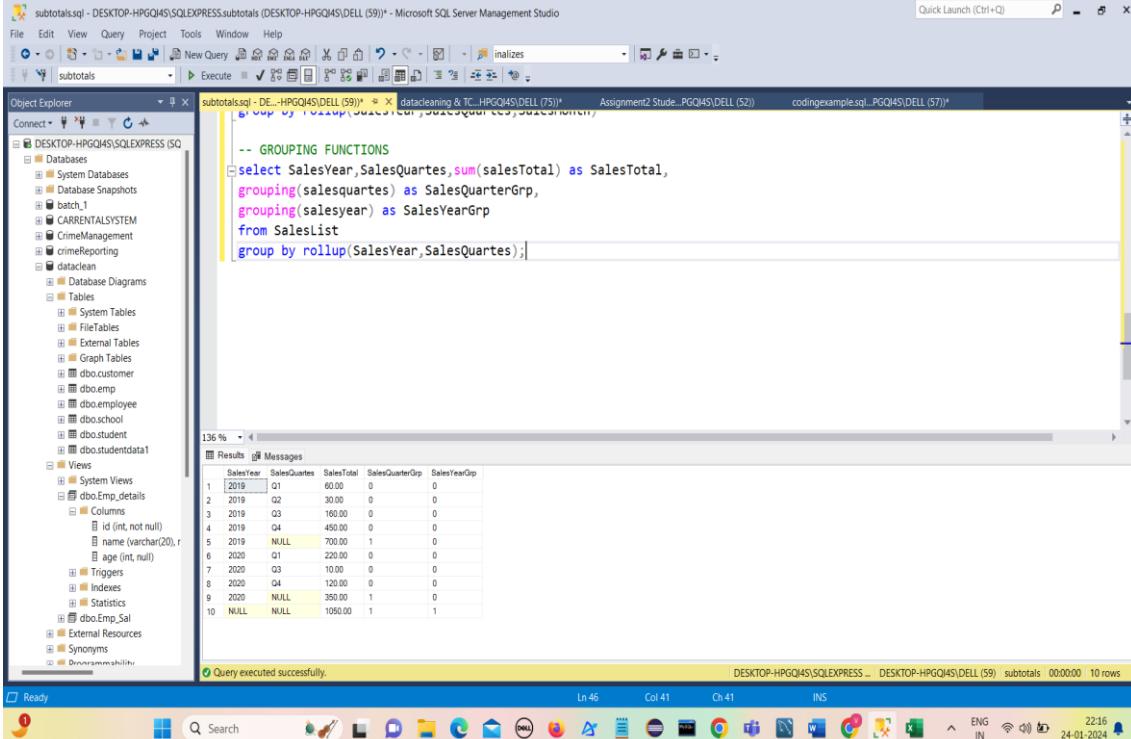
subtotals.sql - DESKTOP-HPGQI4S\SQLEXPRESS.subtotals (DESKTOP-HPGQI4S\DELL (59)) - Microsoft SQL Server Management Studio
File Edit View Query Project Tools Window Help
New Query Execute
subtotals.sql : DE...HPGQI4S\DELL (59)* + datacleaning & TC_HPGQI4S\DELL (75)* Assignment2 Stude...PGQI4S\DELL (52)* codingexample.sql_PQI4S\DELL (57)*
--4) pass 3 columns into the ROLLUP extension
select SalesYear,SalesQuarters,SalesMonth,sum(salesTotal) as SalesTotal from SalesList
group by rollup(SalesYear,SalesQuarters,SalesMonth);

Results Messages
SalesYear SalesQuarters SalesMonth SalesTotal
1 2019 Q1 March 60.00
2 2019 Q1 NULL 60.00
3 2019 Q2 March 30.00
4 2019 Q2 NULL 30.00
5 2019 Q3 July 160.00
6 2019 Q3 NULL 160.00
7 2019 Q4 November 300.00
8 2019 Q4 October 150.00
9 2019 Q4 NULL 450.00
10 2019 NULL NULL 700.00
11 2020 Q1 March 220.00
12 2020 Q1 NULL 220.00
13 2020 Q3 July 10.00
14 2020 Q3 NULL 10.00
15 2020 Q4 November 120.00
16 2020 Q4 NULL 120.00
17 2020 NULL NULL 350.00
18 NULL NULL NULL 1050.00
Query executed successfully.

```

GROUPING function

The GROUPING function is used to determine whether the columns in the GROUP BY list have been aggregated.



subtotals.sql - DESKTOP-HPGQ4S\SQLEXPRESS.subtotals (DESKTOP-HPGQ4S\DELL (59)) - Microsoft SQL Server Management Studio

```
-- GROUPING FUNCTIONS
select SalesYear,SalesQuarters,sum(salesTotal) as SalesTotal,
grouping(salesquarters) as SalesQuarterGrp,
grouping(salesyear) as SalesYearGrp
from SalesList
group by rollup(SalesYear,SalesQuarters);
```

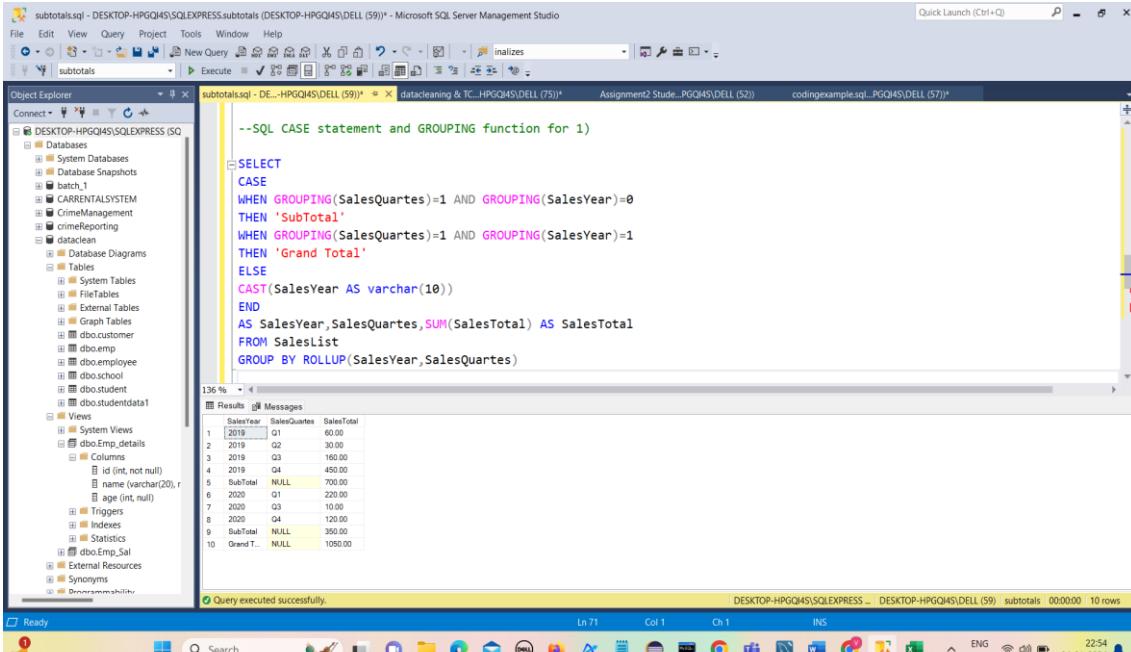
Results

SalesYear	SalesQuarters	SalesTotal	SalesQuarterGrp	SalesYearGrp
1	2019 Q1	60.00	0	0
2	2019 Q2	30.00	0	0
3	2019 Q3	160.00	0	0
4	2019 Q4	450.00	0	0
5	2019 NULL	700.00	1	0
6	2020 Q1	220.00	0	0
7	2020 Q3	10.00	0	0
8	2020 Q4	120.00	0	0
9	2020 NULL	350.00	1	0
10	NULL NULL	1050.00	1	1

Query executed successfully.

DESKTOP-HPGQ4S\SQLEXPRESS - DESKTOP-HPGQ4S\DELL (59) subtotals 00:00:00 10 rows

SQL CASE statement and GROUPING function together so that we will replace NULL values with more meaningful explanations.



subtotals.sql - DESKTOP-HPGQ4S\SQLEXPRESS.subtotals (DESKTOP-HPGQ4S\DELL (59)) - Microsoft SQL Server Management Studio

```
--SQL CASE statement and GROUPING function for 1
SELECT
CASE
WHEN GROUPING(SalesQuarters)=1 AND GROUPING(SalesYear)=0
THEN 'SubTotal'
WHEN GROUPING(SalesQuarters)=1 AND GROUPING(SalesYear)=1
THEN 'Grand Total'
ELSE
CAST(SalesYear AS varchar(10))
END
AS SalesYear,SalesQuarters,SUM(SalesTotal) AS SalesTotal
FROM SalesList
GROUP BY ROLLUP(SalesYear,SalesQuarters);
```

Results

SalesYear	SalesQuarters	SalesTotal
1	2019 Q1	60.00
2	2019 Q2	30.00
3	2019 Q3	160.00
4	2019 Q4	450.00
5	2019 NULL	700.00
6	2020 Q1	220.00
7	2020 Q3	10.00
8	2020 Q4	120.00
9	SubTotal NULL	350.00
10	GrandT.. NULL	1050.00

Query executed successfully.

DESKTOP-HPGQ4S\SQLEXPRESS - DESKTOP-HPGQ4S\DELL (59) subtotals 00:00:00 10 rows

```
-->
--select SalesYear,SalesQuarters,SalesMonth,sum(salesTotal) as salesTotal,
grouping(salesmonth) as SalesMonthGrp,
grouping(salesquarters) as SalesQuartersGrp,
grouping(salesyear) as SalesYearGrp
from SalesList
group by rollup(salesYear,salesQuartes,salesMonth);
```

The screenshot shows the SQL Server Management Studio interface. The Object Explorer on the left lists the database structure, including tables like `dbo.customer`, `dbo.emp`, and `dbo.school`. The central pane displays the query above, which calculates subtotals for SalesYear, SalesQuarters, and SalesMonth using the `SUM` function and grouping functions. The results pane shows 18 rows of data, and the status bar at the bottom indicates the query was executed successfully.

Calculate subtotal in SQL query only for one column

use the ROW_NUMBER() and NEWID() function together to add unique number and this unique number is used for grouping

```
--Calculate subtotal in SQL query only for one column
--select salesMonth,salesTotal,ROW_NUMBER() OVER(order by newid()) as RowNumber
from SalesList
```

The screenshot shows the SQL Server Management Studio interface. The Object Explorer on the left lists the database structure. The central pane displays the query above, which uses the `ROW_NUMBER()` function over a partition defined by `newid()` to generate a unique row number for each `salesMonth` entry. The results pane shows 10 rows of data, and the status bar at the bottom indicates the query was executed successfully.

Use CTE (Common Table Expressions) to work on this result set temporarily to aggregate sales amount and adding the extra subtotals rows.

subtotals.sql - DESKTOP-HPGQ4S\SQLEXPRESS.subtotals (DESKTOP-HPGQ4S\DELL (59)) - Microsoft SQL Server Management Studio

```
-- use CTE to aggregate sales amount and add extra sub total rows
WITH CTE AS
(
    SELECT SalesMonth, SalesTotal ,
    ROW_NUMBER() OVER(ORDER BY NEWID())
    AS RowNumber FROM SalesList
)
SELECT RowNumber ,SalesMonth,SUM(SalesTotal) AS SalesTotal
FROM CTE
GROUP BY ROLLUP(SalesMonth, RowNumber)
```

Results

RowNumber	SalesMonth	SalesTotal
1	July	160.00
2	July	10.00
3	NULL	170.00
4	March	60.00
5	March	90.00
6	March	170.00
7	NULL	280.00
8	May	30.00
9	Mar	30.00
10	November	120.00
11	November	30.00
12	November	120.00
13	NULL	420.00
14	October	150.00
15	October	150.00
16	NULL	300.00

Query executed successfully.

GROUPING SET extension

It is a GROUP BY extension and it allows us to display multiple grouping set in one query.

Without using group set extension

subtotals.sql - DESKTOP-HPGQ4S\SQLEXPRESS.subtotals (DESKTOP-HPGQ4S\DELL (59)) - Microsoft SQL Server Management Studio

```
FROM CTE
GROUP BY ROLLUP(SalesMonth, RowNumber)

--without GROUPING SET extension

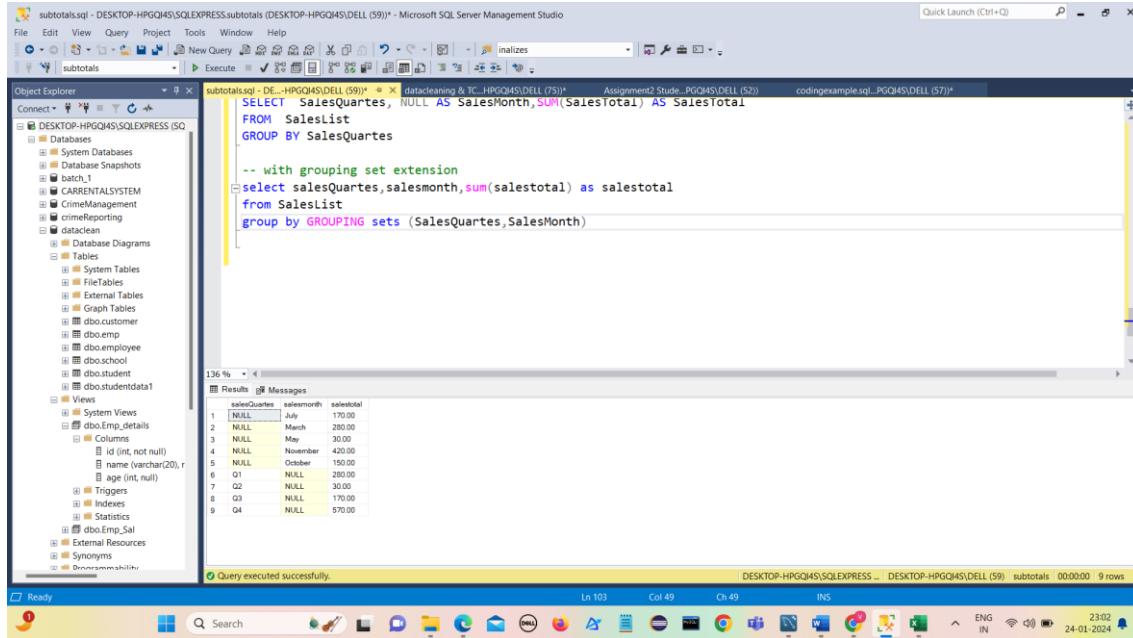
SELECT NULL AS SalesQuarter, SalesMonth,SUM(SalesTotal) AS SalesTotal
FROM Saleslist
GROUP BY SalesMonth
UNION ALL
SELECT SalesQuarters, NULL AS SalesMonth,SUM(SalesTotal) AS SalesTotal
FROM Saleslist
GROUP BY SalesQuarters
```

Results

SalesQuarter	SalesMonth	SalesTotal
1	July	170.00
2	March	280.00
3	May	30.00
4	November	420.00
5	October	150.00
6	Q1	280.00
7	Q2	30.00
8	Q3	170.00
9	Q4	570.00

Query executed successfully.

Using grouping set extension



The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer, there is a database named 'DESKTOP-HPGQ4S\DELL (59)'. A query window titled 'subtotals.sql' is open, displaying the following SQL code:

```
SELECT SalesQuarters, NULL AS SalesMonth, SUM(SalesTotal) AS SalesTotal
FROM SalesList
GROUP BY SalesQuarters

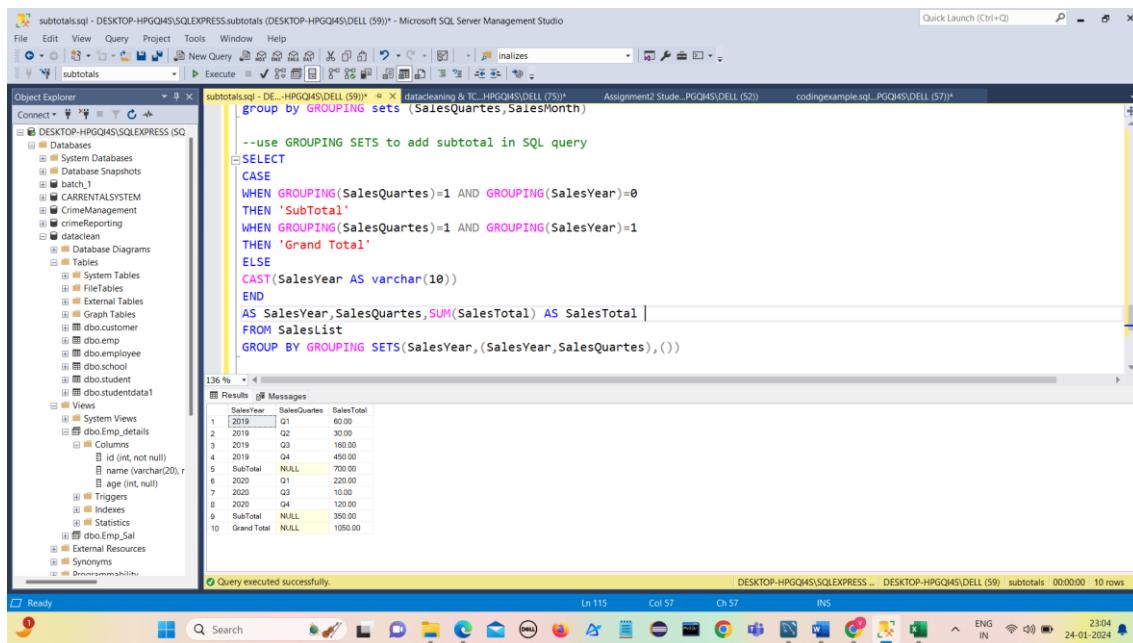
-- with grouping set extension
SELECT salesQuarters, salesmonth, SUM(salestotal) AS salestotal
FROM SalesList
GROUP BY GROUPING SETS (SalesQuarters, SalesMonth)
```

The results pane shows the output of the query:

salesQuarters	salesmonth	salestotal
1	NULL	170.00
2	NULL	200.00
3	NULL	30.00
4	NULL	420.00
5	NULL	150.00
6	Q1	280.00
7	Q2	30.00
8	Q3	170.00
9	Q4	570.00

At the bottom of the results pane, it says 'Query executed successfully.'

use GROUPING SETS to add subtotal in SQL query



The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer, there is a database named 'DESKTOP-HPGQ4S\DELL (59)'. A query window titled 'subtotals.sql' is open, displaying the following SQL code:

```
group by GROUPING SETS (SalesQuarters, SalesMonth)

--use GROUPING SETS to add subtotal in SQL query
CASE
WHEN GROUPING(SalesQuarters)=1 AND GROUPING(SalesYear)=0
THEN 'SubTotal'
WHEN GROUPING(SalesQuarters)=1 AND GROUPING(SalesYear)=1
THEN 'Grand Total'
ELSE
CAST(SalesYear AS varchar(10))
END
AS SalesYear, SalesQuarters, SUM(SalesTotal) AS SalesTotal |
FROM SalesList
GROUP BY GROUPING SETS(SalesYear, (SalesYear, SalesQuarters), ())
```

The results pane shows the output of the query:

SalesYear	SalesQuarters	SalesTotal
1	2019 Q1	60.00
2	2019 Q2	30.00
3	2019 Q3	120.00
4	2019 Q4	450.00
5	SubTotal	700.00
6	2020 Q1	220.00
7	2020 Q3	10.00
8	2020 Q4	120.00
9	SubTotal	350.00
10	Grand Total	1050.00

At the bottom of the results pane, it says 'Query executed successfully.'