Assignment 3

Task 1:

Write a query to find the name and age of the oldest passenger who survived.

```
1   /* Write a query to find the name and age of the oldest passenger who survived */
2
3   select * from titanic;
4
5   select concat(first_name,' ',last_name) AS Passenger_name,age,alive
6   from titanic
7   where age=(select max(age) from titanic where alive='yes');
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Passenger_name	age	alive
Harvey Mikkilineni	58	yes

Task 2:

Create a view to display passenger survival status, class, age, and fare.

```
1   select * from Titanic;
2
3   CREATE VIEW passg_survive_status AS
4   select survived,class,age,fare
5   from titanic;
6
7   select * from passg_survival_status;
```

Result Grid | Filter Rows: | Export: | Wrap Cell

survived	class	age	fare
1	First	58	61770
0	First	54	63090
1	First	38	50806
1	First	35	26969
1	Second	55	43386
0	Second	35	20068
1	Second	34	46632
1	Second	14	29047
0	Third	39	59197
0	Third	35	34048
0	Third	31	66662
1	Third	29	38628

Task 3:

Create a stored procedure to retrieve passengers based on a given age range.

```
1  select * from titanic;
2  DELIMITER //
3  CREATE PROCEDURE passg_agee(in min_age int, in max_age int)
4  BEGIN
5  SELECT CONCAT(first_name,' ',last_name) AS passenger_name,age
6  FROM titanic
7  WHERE age BETWEEN min_age AND max_age;
8  END //
9  DELIMITER ;
10 CALL passg_agee(10,40);
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

passenger_name	age
Ross Kochhar	38
Bruce Popp	35
Jack Greene	35
Peter Hall	34
Adam Vollman	14
Julia Patel	39
David Raphaely	35
Sartha Sully	31
William Smith	29
Nancy Khoo	27
Sigal Kaufling	27
Alex Urman	26

Task 4:

Write a query to categorize passengers based on the fare they paid: 'Low', 'Medium', or 'High'.

```
1  select * from titanic;
2
3  select CONCAT(first_name,' ',last_name) AS passenger_name,age,
4  CASE
5  WHEN fare < 30000 then 'Low'
6  WHEN fare BETWEEN 30000 AND 50000 then 'Medium'
7  ELSE 'High'
8  END AS passg_detail
9  from titanic;
10
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

passenger_name	age	passg_detail
Harvey Mikkilineni	58	High
John Baida	54	High
Ross Kochhar	38	High
Bruce Popp	35	Low
James Zlotkey	55	Medium
Jack Greene	35	Low
Peter Hall	34	Medium
Adam Vollman	14	Low
Julia Patel	39	High
David Raphaely	35	Medium
Sartha Sully	31	High
William Smith	29	Medium

Task 5:

Show each passenger's fare and the fare of the next passenger

```
1 •  select * from titanic;
2
3    select concat(first_name,' ',last_name) as FULL_NAME,fare,
4    LEAD(fare) Over(order by fare desc) as next_passg_fare
5    from titanic;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

FULL_NAME	fare	next_passg_fare
Sartha Sully	66662	63090
John Baida	63090	61770
Harvey Mikkilineni	61770	61211
Irene Davies	61211	59197
Julia Patel	59197	56356
Daniel Weiss	56356	54071
Alex Urman	54071	51428
Mathew Himuro	51428	50806
Ross Kochhar	50806	46632
Peter Hall	46632	46206
Nancy Khoo	46206	43386
James Zlotkey	43386	38628

Task 6:

Show the age of each passenger and the age of the previous passenger

```
1 •  select * from titanic;
2
3    select concat(first_name,' ',last_name) AS Full_Name, age,
4    LAG(age) OVER(
5    order by age desc) as prev_passg_age
6    from titanic;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Full_Name	age	prev_passg_age
Harvey Mikkilineni	58	NULL
James Zlotkey	55	58
John Baida	54	55
Julia Patel	39	54
Ross Kochhar	38	39
Bruce Popp	35	38
Jack Greene	35	35
David Raphaely	35	35
Peter Hall	34	35
Sartha Sully	31	34
William Smith	29	31
Nancy Khoo	27	29

Task 7:

Write a query to rank passengers based on their fare, displaying rank for each passenger.

```
1 •   select * from titanic;
2
3     select concat(first_name,' ',last_name) as Full_Name,fare,
4     RANK() OVER(
5     order by fare DESC) AS Non_continuos_rnk
6     from titanic;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Full_Name	fare	Non_continuos_rnk
Sartha Sully	66662	1
John Baida	63090	2
Harvey Mikkilineni	61770	3
Irene Davies	61211	4
Julia Patel	59197	5
Daniel Weiss	56356	6
Alex Urman	54071	7
Mathew Himuro	51428	8
Ross Kochhar	50806	9
Peter Hall	46632	10
Nancy Khoo	46206	11
James Zlotkey	43386	12

Task 8:

Write a query to rank passengers based on their fare, ensuring no gaps in rank

```
1 •   select * from titanic;
2
3     select concat(first_name,' ',last_name) as Full_Name, fare,
4     dense_rank() over(
5     order by fare asc) as continous_rnk
6     from titanic;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Full_Name	fare	continous_rnk
Jack Greene	20068	1
Sigal Kaufling	23488	2
Steven King	24000	3
Bruce Popp	26969	4
Jason Bernstein	27978	5
Adam Vollman	29047	6
Kevin Rogers	30897	7
David Raphaely	34048	8
William Smith	38628	9
James Zlotkey	43386	10
Nancy Khoo	46206	11
Peter Hall	46632	12

Task 9:

Assign row numbers to passengers based on the order of their fares

```
1 •   select * from titanic;
2
3     select concat(first_name,' ',last_name)as Full_Name,fare,
4     row_number() over(
5     order by fare desc) as row_no
6     from titanic;
7
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Full_Name	fare	row_no
Sartha Sully	66662	1
John Baida	63090	2
Harvey Mikkilineni	61770	3
Irene Davies	61211	4
Julia Patel	59197	5
Daniel Weiss	56356	6
Alex Urman	54071	7
Mathew Himuro	51428	8
Ross Kochhar	50806	9
Peter Hall	46632	10
Nancy Khoo	46206	11
James Zlotkey	43386	12

Task 10:

Use a CTE to calculate the average fare and find passengers who paid more than the average.

```
1 •   select * from titanic;
2
3     WITH Avg_Fare AS
4     (select round(avg(fare),2) as avg_fare from titanic)
5     select CONCAT(t.first_name,' ',t.last_name) AS Full_Name,
6     t.fare,Avg_Fare.avg_fare from titanic t JOIN Avg_Fare ON t.fare > avg_fare;
7
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Full_Name	fare	avg_fare
Harvey Mikkilineni	61770	43616.10
John Baida	63090	43616.10
Ross Kochhar	50806	43616.10
Peter Hall	46632	43616.10
Julia Patel	59197	43616.10
Sartha Sully	66662	43616.10
Nancy Khoo	46206	43616.10
Alex Urman	54071	43616.10
Irene Davies	61211	43616.10
Mathew Himuro	51428	43616.10
Daniel Weiss	56356	43616.10