

# RACHIT CHIKARA

## ASSESSMENT - DATA ANALYST, PACKT

**NOTE** - For this assessment, I have created the sample data in MS Excel and imported it into **Microsoft SQL server management Studio**.

### Q1 - Assumptions:

Since LF\_1, LF\_2 & LF\_3 are three stages that users go through before onboarding, So to calculate the duration in days that users take before reaching the current stage I am considering the difference that I have to calculate from LF\_1 to the current stage and not from the previous stage to the current stage.

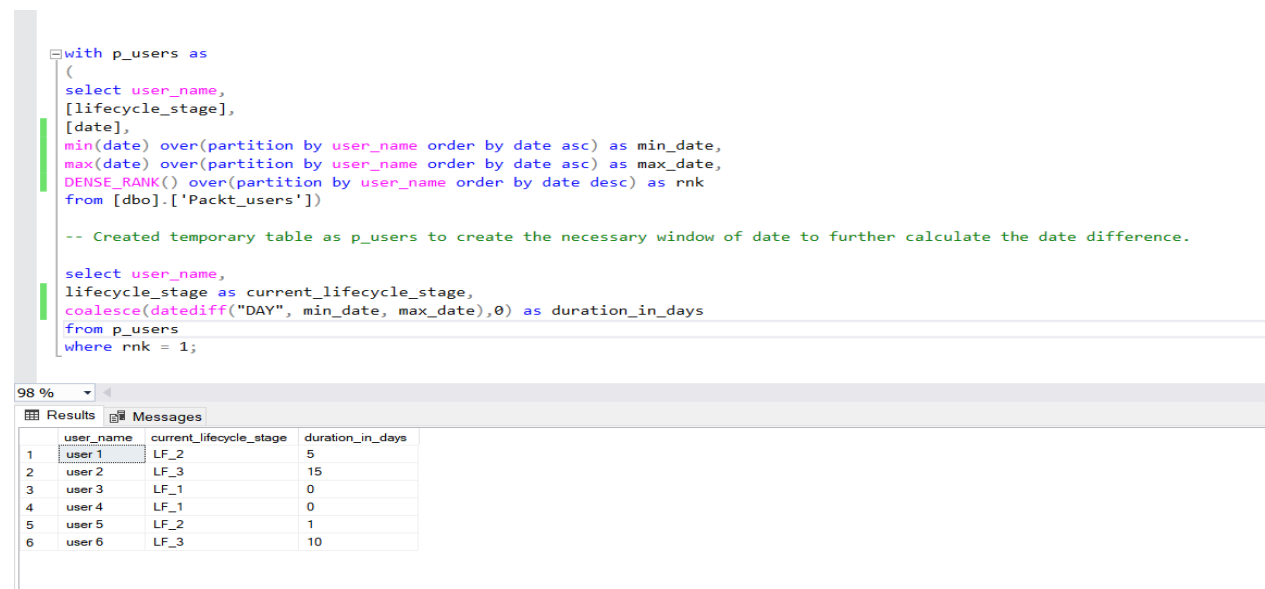
#### SQL Query 1:

```
with p_users as
(
select user_name,
[lifecycle_stage],
[date],
min(date) over(partition by user_name order by date asc) as min_date,
max(date) over(partition by user_name order by date asc) as max_date,
DENSE_RANK() over(partition by user_name order by date desc) as rnk
from [dbo].['Packt_users'])
```

-- Created temporary table as p\_users to create the necessary window of date to further calculate the date difference.

```
select user_name,
lifecycle_stage as current_lifecycle_stage,
coalesce(datediff("DAY", min_date, max_date),0) as duration_in_days
from p_users
where rnk = 1;
```

### 1st SQL QUERY & OUTPUT IMAGE:



The screenshot displays the SQL Server Enterprise Edition interface. The top pane shows a SQL query that defines a CTE named 'p\_users' and then selects user information along with the duration in days between their first and last lifecycle stage. The bottom pane shows the results of this query, which are displayed in a table with 6 rows and 3 columns: user\_name, current\_lifecycle\_stage, and duration\_in\_days.

```
with p_users as
(
select user_name,
[lifecycle_stage],
[date],
min(date) over(partition by user_name order by date asc) as min_date,
max(date) over(partition by user_name order by date asc) as max_date,
DENSE_RANK() over(partition by user_name order by date desc) as rnk
from [dbo].['Packt_users'])

-- Created temporary table as p_users to create the necessary window of date to further calculate the date difference.

select user_name,
lifecycle_stage as current_lifecycle_stage,
coalesce(datediff("DAY", min_date, max_date),0) as duration_in_days
from p_users
where rnk = 1;
```

	user_name	current_lifecycle_stage	duration_in_days
1	user 1	LF_2	5
2	user 2	LF_3	15
3	user 3	LF_1	0
4	user 4	LF_1	0
5	user 5	LF_2	1
6	user 6	LF_3	10

## Q2 - Assumptions:

To write a SQL query to find the days when the sales were higher than the previous day. I am assuming that the ID column in the given sample data is the order ID. Also, I am considering the order ID as a primary key for the given data. It should be unique. If there are any duplicate ID values, this means that there are duplicate records in the data. These **duplicate records** should be **removed** before starting any **analysis**. So I have done the required analysis only after considering that ID(4, 5 & 6) are the duplicate values.

## SQL Query 2:

```
With P_total_sales as
(select distinct * from [dbo].['Packt_sales']),
p_sales as
(select
[ID (int)],
[SaleDate (date)],
[SaleAmount (Float)],
sum([SaleAmount (Float)]) over(partition by [SaleDate (date)] order by
[SaleDate (date)]) as Daily_sales,
row_number() over (partition by [SaleDate (date)] order by [SaleAmount (Float)]
desc) as rnk
from P_total_sales),

p_sales_new as
(select [ID (int)],
[SaleDate (date)],
daily_sales,
coalesce(lag(daily_sales) over(order by [SaleDate (date)]),0) as Prev_day_sales
from p_sales
where rnk = 1)

select *
/*, case
when Prev_day_sales > Daily_sales then 'Lower sales than prev day sales'
when Prev_day_sales < Daily_sales then 'Higher sales than prev day sales'
else 'Equal sales than prev day sales' end as Daily_sales_status */
from p_sales_new
where Daily_sales > Prev_day_sales;

/* Filtering out only records where present-day sales are higher than the previous
day's sales using where clause */
```

## 2nd SQL QUERY & OUTPUT IMAGE:

The screenshot displays the SQL Server Enterprise Manager interface. On the left, the 'Object Explorer' pane shows the 'Packt Database' expanded, with 'Tables' listed. The main area shows a SQL query in the 'Query Editor' window. The query is as follows:

```
With P_total_sales as
(select distinct * from [dbo].['Packt_sales']),
p_sales as
(select
[ID (int)],
[SaleDate (date)],
[SaleAmount (Float)],
sum([SaleAmount (Float)]) over(partition by [SaleDate (date)] order by [SaleDate (date)]) as Daily_sales,
row_number() over (partition by [SaleDate (date)] order by [SaleAmount (Float)] desc) as rnk
from P_total_sales),

p_sales_new as
(select [ID (int)],
[SaleDate (date)],
daily_sales,
coalesce(lag(daily_sales) over(order by [SaleDate (date)]),0) as Prev_day_sales
from p_sales
where rnk = 1)

select *
/*, case
when Prev_day_sales > Daily_sales then 'Lower sales than prev day sales'
when Prev_day_sales < Daily_sales then 'Higher sales than prev day sales'
else 'Equal sales than prev day sales' end as Daily_sales_status */
from p_sales_new
where Daily_sales > Prev_day_sales;
```

Below the query, the 'Results' pane shows the output of the query. The results are displayed in a table with the following columns: ID (int), SaleDate (date), daily\_sales, and Prev\_day\_sales. The results are as follows:

ID (int)	SaleDate (date)	daily_sales	Prev_day_sales
1	2021-06-04 00:00:00.000	23000	0
2	2021-06-05 00:00:00.000	24000	23000
3	2021-06-07 00:00:00.000	25000	24000
4	2021-06-08 00:00:00.000	26000	25000

## Q3 - Assumptions:

To write a SQL query to delete the duplicate records and to display invalid emails.

I am considering these guidelines for an email address to be a valid email address:

1. It must have at least one @ symbol. The @ symbol separates the username from the domain name.
2. It must have a username. The username is part of the email address that comes before the @ symbol. It can be any combination of letters, numbers, and underscores.
3. It must have a domain name. The domain name is the part of the email address that comes after the @ symbol. It is usually the name of the email service provider, such as Gmail, Yahoo, or Outlook.
4. It must have a top-level domain. The top-level domain is the part of the domain name that comes after the period. It is usually a 2-letter code that indicates the country or region where the email service provider is located, such as .com.

5. It must be at least 3 characters long. The shortest possible email address is 3 characters long, such as "a@b.c".

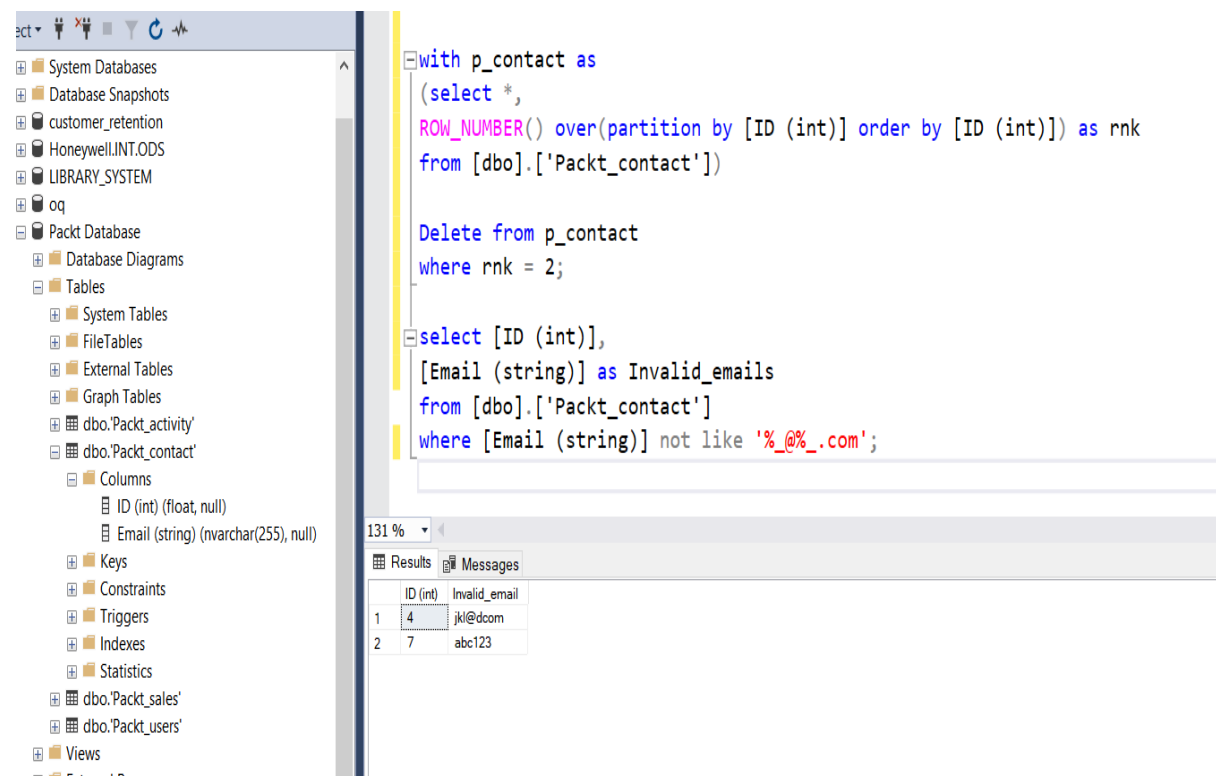
### SQL Query 3:

```
with p_contact as
(select *,
ROW_NUMBER() over(partition by [ID (int)] order by [ID (int)]) as rnk
from [dbo].['Packt_contact'])
```

```
Delete from p_contact
where rnk = 2;
```

```
select [ID (int)],
[Email (string)] as Invalid_emails
from [dbo].['Packt_contact']
where [Email (string)] not like '%_@%_.com';
```

### 3rd SQL QUERY & OUTPUT IMAGE:



The screenshot displays the SQL Server Enterprise Manager interface. On the left, the 'Packt Database' is expanded, showing the 'Packt\_contact' table. The main pane shows the following SQL query:

```
with p_contact as
(select *,
ROW_NUMBER() over(partition by [ID (int)] order by [ID (int)]) as rnk
from [dbo].['Packt_contact'])

Delete from p_contact
where rnk = 2;

select [ID (int)],
[Email (string)] as Invalid_emails
from [dbo].['Packt_contact']
where [Email (string)] not like '%_@%_.com';
```

Below the query, the 'Results' tab shows the output of the final SELECT statement:

	ID (int)	Invalid_email
1	4	jk@dscom
2	7	abc123

#### **Q4 - Assumptions:**

For the given sample data of the activity of users & to write a SQL query to find the users who got a “credit” for their streak completion. Where, a continuous activity of four weeks is considered a streak completion (i.e., a user must be active at least once a week for 4 consecutive weeks Also, I am considering that in case a user is active for more than once in a week will only be considered as the one streak.

#### **SQL Query 4:**

**With p\_activity as**

```
(select *,  
lag([activity_week]) over(partition by user_id order by [activity_week]) as  
prev_activity_week ,  
dense_rank() over(partition by user_id order by activity_week) as rnk  
from [dbo].['Packt_activity']),
```

**P\_activity\_1 as**

```
(select *,  
coalesce(DATEDIFF("DAY", prev_activity_week, activity_week),0) as  
day_count_between_act_week  
from p_activity),
```

**packt\_user\_activity as**

```
(select *,  
case  
when max(day_count_between_act_week) over (partition by user_id order by  
[activity_week] ) <= 7  
then dense_rank() over(partition by user_id order by activity_week)  
else 1  
end as streak_number  
from P_activity_1  
where rnk < 5)
```

```
select  
user_id,  
activity_date,  
activity_week,  
streak_number,  
case  
when streak_number % 4 = 0 then 1  
else 0 end as credited  
from packt_user_activity;
```

## 4th SQL QUERY & OUTPUT IMAGE:

The screenshot displays the SQL Server Enterprise Manager interface. On the left, the 'Server Explorer' pane shows the 'Packt Database' expanded, with 'Tables' and 'dbo.Packt\_activity' selected. The main pane shows a SQL query in the 'Query Editor' window. The query is a complex T-SQL statement that uses window functions to calculate streaks of activity for different users. The query is as follows:

```
With p_activity as
(select *,
lag([activity_week]) over(partition by user_id order by [activity_week]) as prev_activity_week ,
dense_rank() over(partition by user_id order by activity_week) as rnk
from [dbo].[Packt_activity]),

P_activity_1 as
(select *,
coalesce(DATEDIFF("DAY", prev_activity_week, activity_week),0) as day_count_between_act_week
from p_activity),

packt_user_activity as
(select *,
case
when max(day_count_between_act_week) over (partition by user_id order by [activity_week]) <= 7
then dense_rank() over(partition by user_id order by activity_week)
else 1
end as streak_number
from P_activity_1
where rnk < 5)

select
user_id,
activity_date,
activity_week,
streak_number,
case
when streak_number % 4 = 0 then 1
else 0 end as credited
from packt_user_activity;
```

Below the query editor, the 'Results' pane shows the output of the query. The results are displayed in a table with the following columns: user\_id, activity\_date, activity\_week, streak\_number, and credited. The table contains 16 rows of data, showing activity for users U1, U2, U3, and U4. The 'credited' column indicates whether the user's activity streak is credited (1) or not (0).

	user_id	activity_date	activity_week	streak_number	credited
1	U1	2022-12-27 00:00:00.000	2023-01-01 00:00:00.000	1	0
2	U1	2022-12-29 00:00:00.000	2023-01-01 00:00:00.000	1	0
3	U1	2023-01-08 00:00:00.000	2023-01-08 00:00:00.000	2	0
4	U1	2023-01-09 00:00:00.000	2023-01-15 00:00:00.000	3	0
5	U1	2023-01-19 00:00:00.000	2023-01-22 00:00:00.000	4	1
6	U2	2023-01-05 00:00:00.000	2023-01-08 00:00:00.000	1	0
7	U2	2023-01-10 00:00:00.000	2023-01-15 00:00:00.000	2	0
8	U2	2023-01-22 00:00:00.000	2023-01-22 00:00:00.000	3	0
9	U2	2023-01-21 00:00:00.000	2023-01-22 00:00:00.000	3	0
10	U3	2023-02-07 00:00:00.000	2023-02-12 00:00:00.000	1	0
11	U3	2023-02-19 00:00:00.000	2023-02-19 00:00:00.000	2	0
12	U3	2023-03-01 00:00:00.000	2023-03-05 00:00:00.000	1	0
13	U3	2023-03-05 00:00:00.000	2023-03-05 00:00:00.000	1	0
14	U3	2023-03-16 00:00:00.000	2023-03-19 00:00:00.000	1	0
15	U4	2023-01-04 00:00:00.000	2023-01-08 00:00:00.000	1	0
16	U4	2023-01-20 00:00:00.000	2023-01-22 00:00:00.000	1	0

At the bottom of the interface, a status bar indicates 'Query executed successfully.'