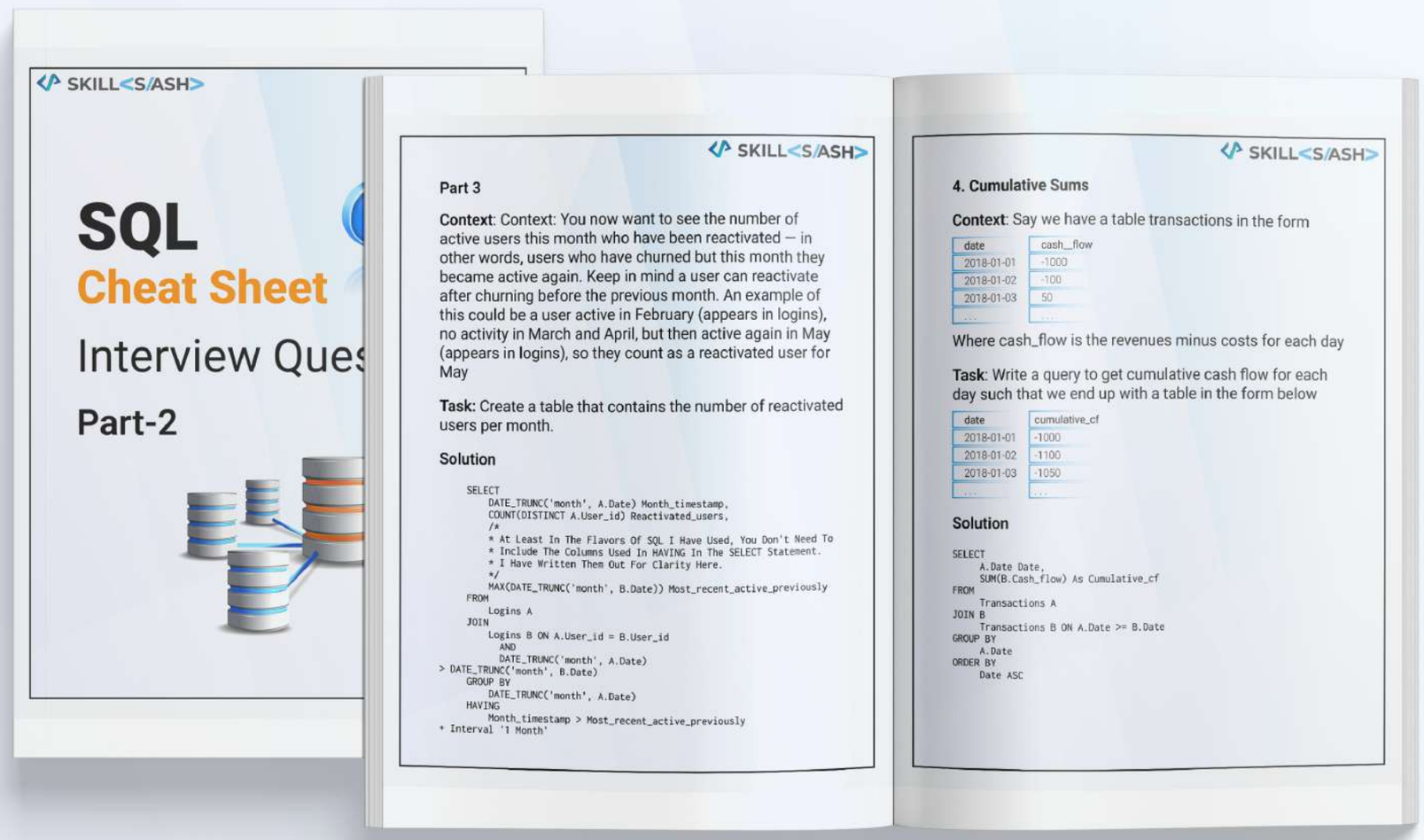
SKILL<S/ASH>

SQL Cheat Sheet

Interview Questions

Part-2



Part 3

Context: Context: You now want to see the number of active users this month who have been reactivated — in other words, users who have churned but this month they became active again. Keep in mind a user can reactivate after churning before the previous month. An example of this could be a user active in February (appears in logins), no activity in March and April, but then active again in May (appears in logins), so they count as a reactivated user for May

Task: Create a table that contains the number of reactivated users per month.

Solution

```
SELECT
    DATE_TRUNC('month', A.Date) Month_timestamp,
    COUNT(DISTINCT A.User_id) Reactivated_users,
    /*
    * At Least In The Flavors Of SQL I Have Used, You Don't Need To
    * Include The Columns Used In HAVING In The SELECT Statement.
    * I Have Written Them Out For Clarity Here.
    */
    MAX(DATE_TRUNC('month', B.Date)) Most_recent_active_previously
FROM
    Logins A
JOIN
    Logins B ON A.User_id = B.User_id
      AND
      DATE_TRUNC('month', A.Date)
> DATE_TRUNC('month', B.Date)
GROUP BY
    DATE_TRUNC('month', A.Date)
HAVING
    Month_timestamp > Most_recent_active_previously
+ Interval '1 Month'
```

4. Cumulative Sums

Context: Say we have a table transactions in the form

date	cash__flow
2018-01-01	-1000
2018-01-02	-100
2018-01-03	50
. . .	. . .

Where cash_flow is the revenues minus costs for each day

Task: Write a query to get cumulative cash flow for each day such that we end up with a table in the form below

date	cumulative_cf
2018-01-01	-1000
2018-01-02	-1100
2018-01-03	-1050
. . .	. . .

Solution

```
SELECT
      A.Date Date,
      SUM(B.Cash_flow) As Cumulative_cf
FROM
      Transactions A
JOIN B
      Transactions B ON A.Date >= B.Date
GROUP BY
      A.Date
ORDER BY
      Date ASC
```

Alternate Solution Using A Window Function (More Efficient!):

```
SELECT
      Date,
      SUM(Cash_flow) OVER (ORDER BY Date ASC) As Cumulative_cf
FROM
      Transactions
ORDER BY
      Date ASC
```

5. Rolling Averages

Note: There are different ways to compute rolling/ moving averages. Here we'll use a preceding average which means that the metric for the 7th day of the month would be the average of the preceding 6 days and that day itself.

Context: Say we have table signups in the form

date	sign_ups
2018-01-01	10
2018-01-02	20
2018-01-03	50
. . .	. . .
2018-10-01	35

Task: Write a query to get 7-day rolling (preceding) average of daily sign ups

Solution

```
SELECT
    A.Date,
    AVG(B.Sign_ups) Average_sign_ups
FROM
    Signups A
JOIN
    Signups B ON A.Date <= B.Date + Interval '6 Days' AND
A.Date >= B.Date
GROUP BY
    A.Date
```

6. Multiple Join Conditions

Context: Say we have a table emails that includes emails sent to and from

Task: Write a query to get the response time per email (id) sent to zach@g.com. Do not include ids that did not receive a response from zach@g.com. Assume each email thread has a unique subject. Keep in mind a thread may have multiple responses back-and-forth between zach@g.com and another email address.

Solution

```
SELECT
      A.Id,
      MIN(B.Timestamp) - A.Timestamp As Time_to_respond
FROM
      Emails A
JOIN
      Emails B
          ON
              B.Subject = A.Subject
          AND
              A.To = B.From
          AND
              A.From = B.To
          AND
              A.Timestamp < B.Timestamp
WHERE
      A.To = 'zach@G.Com'
GROUP BY
      A.Id
```

Window Function Practice Problems

1. Get The ID With The Highest Value

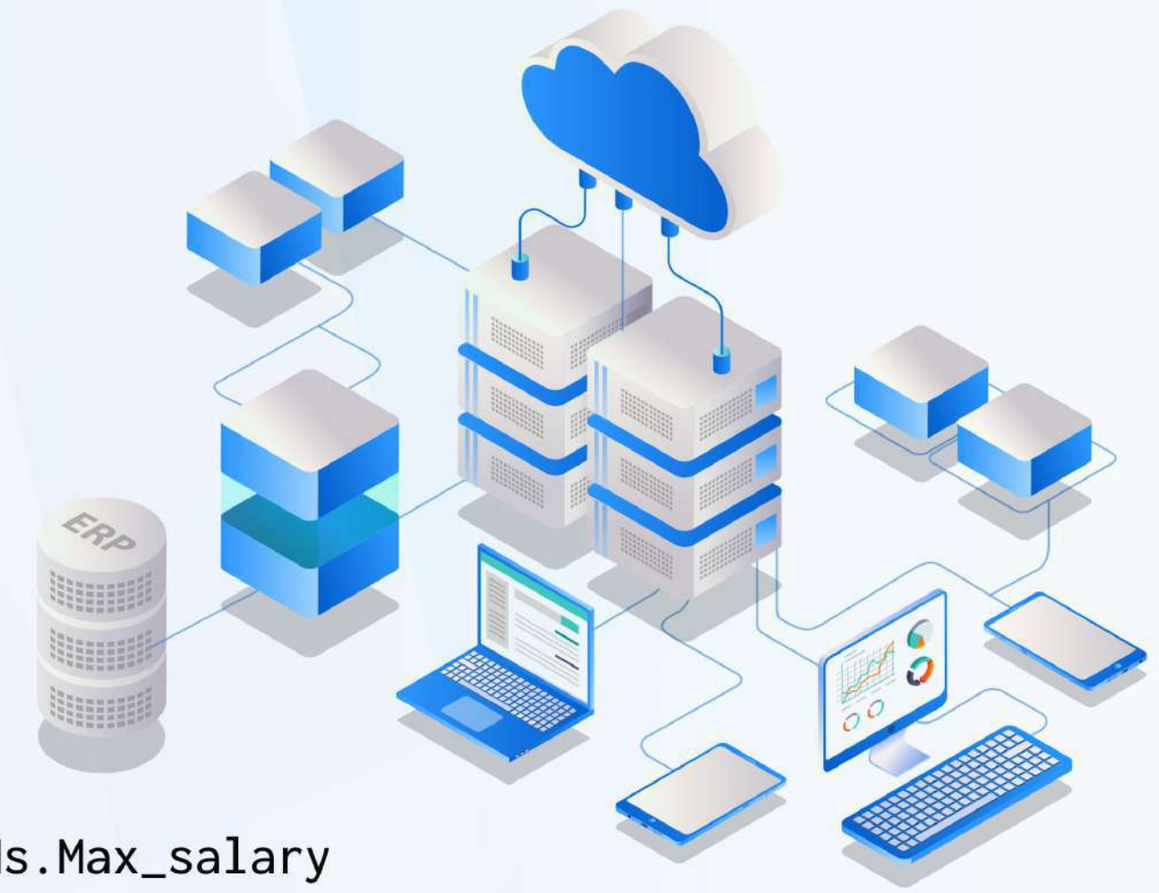
Context: Say we have a table salaries with data on employee salary and department in the following format

depname	empno	salary
develop	11	5200
develop	7	4200
develop	9	4500
develop	8	6000
develop	10	5200
personnel	5	3500
personnel	2	3900
sales	3	4800
sales	1	5000
sales	4	4800

Task: Write a query to get the empno with the highest salary. Make sure your solution can handle ties!

Solution

```
WITH Max_salary AS (
     SELECT
          MAX(Salary) Max_salary
     FROM
          Salaries
     )
SELECT
     S.Empno
FROM
     Salaries S
JOIN
     Max_salary Ms ON S.Salary = Ms.Max_salary
```



Alternate Solution Using RANK():

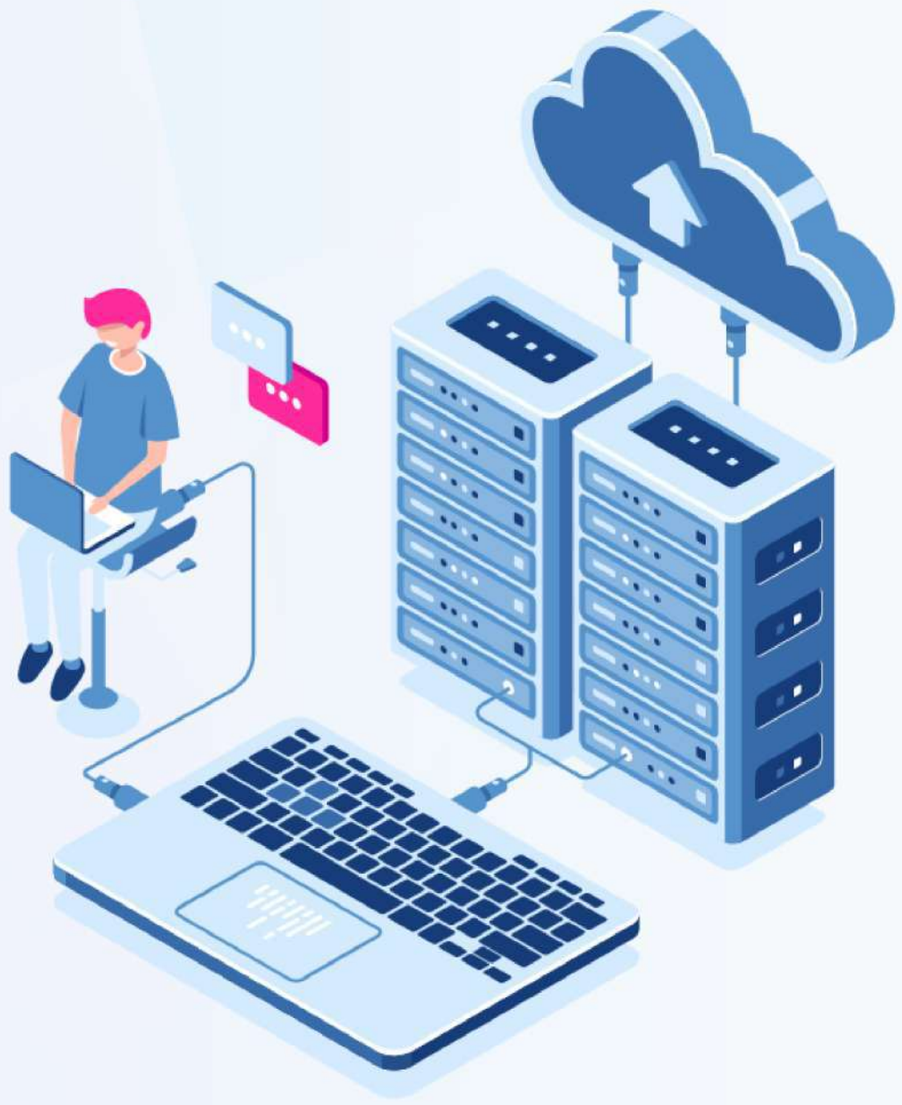
```
WITH Sal_rank AS
   (SELECT
     Empno,
     RANK() OVER(ORDER BY Salary DESC) Rnk
   FROM
     Salaries)
SELECT
   Empno
FROM
   Sal_rank
WHERE
   Rnk = 1;
```

2. Average And Rank With A Window Function (Multi-Part)

Part-1

Note: Say we have a table salaries in the format:

depname	empno	salary
develop	11	5200
develop	7	4200
develop	9	4500
develop	8	6000
develop	10	5200
personnel	5	3500
personnel	2	3900
sales	3	4800
sales	1	5000
sales	4	4800



Task: Write a query that returns the same table, but with a new column that has average salary per depname. We would expect a table in the form

depname	empno	salary	avg_salary
develop	11	5200	5020
develop	7	4200	5020
develop	9	4500	5020
develop	8	6000	5020
develop	10	5200	5020
personnel	5	3500	3700
personnel	2	3900	3700
sales	3	4800	4867
sales	1	5000	4867
sales	4	4800	4867

Solution

```
SELECT
      *,
      /*
      * AVG() Is A Postgres Command, But Other SQL Flavors
Like BigQuery Use
      * AVERAGE()
      */
      ROUND(AVG(Salary),0) OVER (PARTITION BY Depname) Avg_salary
FROM
      Salaries
```

SKILL<S/ASH>

Got value?
Save it.

Want to see more content like this? **Follow us**