**Janet Zhang - SQL**  
1) When a customer updates their contact phone number, what query should we run to save that update to the database?

To update the customer's contact phone number in the `tblSubscriptionInfo` table, I would use an SQL `UPDATE` query. Here's the query I can use:

UPDATE tblSubscriptionInfo

SET customer\_contact\_phone = ‘new\_phone\_number’

WHERE customer\_id = desired\_customer\_id;

Here's an example with actual values:

UPDATE tblSubscriptionInfo

SET customer\_contact\_phone = ‘65-12345678’

WHERE customer\_id = 123;

This query will find the customer with the ID ‘123’ and update their contact phone number to ’65-12345678’.

2) What could we do to speed up this query?

To improve the efficiency of this query, I will consider using prepared statements or bulk updates. (The efficiency gains will depend on the database system we’re using)

1. Prepared statements: Prepared statements can be used to execute the same SQL statement multiple times with different parameter values. They can be more efficient because the database prepares the statement once and then executes it with different parameters, reducing parsing overhead. While this might not make a significant difference for a single update, it can be beneficial if we’re updating multiple rows.

PREPARE update\_phone AS

UPDATE tblSubscriptionInfo

SET customer\_contact\_phone = $1

WHERE customer\_id = $2;

EXECUTE update\_phone (‘65-12345678’, 123);

1. Bulk updates: Perform a bulk update (or batch processing) for multiple customers at once. This can be more efficient than updating each row individually, as it minimizes the number of transactions and communication with the database server.

UPDATE tblSubscriptionInfo

SET customer\_contact\_phone = CASE

WHEN customer\_id = 123 THEN ’65-12345678’

WHEN customer\_id = 124 THEN ’65-23456789’

…

ELSE customer\_contact\_phone

END

WHERE customer\_id IN (123, 124 /\*, more customer IDs \*/);

Other strategies are:

1. Indexes: Ensure that the customer\_id column is indexed. Indexes can significantly speed up the retrieval and updating of rows by allowing the database to quickly locate the relevant rows. Without an index, the database would need to perform a full table scan to find the matching row.
2. Query: Avoid complex query constraints
3. Database:
   1. Regular database maintenance like optimizing indexes
   2. Implement a caching layer to store frequently accessed data
   3. Sharding (horizontal partitioning) across multiple servers of partitioning techniques
   4. Scaling: more servers

3) number of subscribers whose subscriptions will be ending in 2023;

SELECT COUNT(\*) AS subscriber\_count

FROM tblSubscriptionInfo

WHERE YEAR(subscription\_end\_date) = 2023;

4) number of subscribers who have subscribed for more than 3 months in 2022;

SELECT COUNT(\*) AS subscriber\_count

FROM tblSubscriptionInfo

WHERE YEAR(subscription\_start\_date) = 2022

AND DATEDIFF(MONTH, subscription\_start\_date, subscription\_end\_date) > 3;

5) subscribers who have subscribed for more than two products;

SELECT customer\_id, customer\_name

FROM tblSubscriptionInfo

GROUP BY customer\_id, customer\_name

HAVING COUNT(DISTINCT product\_id) > 2;

6) product with the most/2ndmost/3rdmost number of subscribers in 2022;

(LIMIT and OFFSET queries are used to fetch the top products based on subscriber counts)

Most number of subscribers:

SELECT product\_id, product\_name, COUNT(\*) AS subscriber\_count

FROM tblSubscriptionInfo

WHERE YEAR(subscription\_start\_date) = 2022

GROUP BY product\_id, product\_name

ORDER BY subscriber\_count DESC

LIMIT 1;

Second most number of subscribers:

SELECT product\_id, product\_name, COUNT(\*) AS subscriber\_count

FROM tblSubscriptionInfo

WHERE YEAR(subscription\_start\_date) = 2022

GROUP BY product\_id, product\_name

ORDER BY subscriber\_count DESC

LIMIT 1 OFFSET 1;

Third most number of subscribers:

SELECT product\_id, product\_name, COUNT(\*) AS subscriber\_count

FROM tblSubscriptionInfo

WHERE YEAR(subscription\_start\_date) = 2022

GROUP BY product\_id, product\_name

ORDER BY subscriber\_count DESC

LIMIT 1 OFFSET 2;

7) number of subscribers who have re-subscribed more than once for each product;

(Subquery: calculate the number of distinct subscriptions start dates for each product-customer combination. For num\_subscriptions is greater than 1, indicating that the customer has re-subscribed. The main query joins this result and groups the results to get the count of subscribers who have re-subscribed more than once for each product.)

SELECT product\_id, product\_name, COUNT(\*) AS re\_subscription\_count

FROM (

SELECT product\_id, customer\_id, COUNT(DISTINCT subscription\_start\_date) AS num\_subscriptions

FROM tblSubscriptionInfo

GROUP BY product\_id, customer\_id

HAVING num\_subscriptions > 1

) AS re\_subscriptions

JOIN tblSubscriptionInfo ON re\_subscriptions.product\_id = tblSubscriptionInfo.product\_id

AND re\_subscriptions.customer\_id = tblSubscriptionInfo.customer\_id

GROUP BY product\_id, product\_name;

8) subscribers who have re-subscribed to a higher version of the product in 2023 - for example Autocad 2022 to Autocad 2023.

SELECT DISTINCT s1.customer\_id, s1.customer\_name, s1.product\_name AS old\_version, s2.product\_name AS new\_version

FROM tblSubscriptionInfo s1

JOIN tblSubscriptionInfo s2

ON s1.customer\_id = s2.customer\_id

AND s1.product\_name = Autocad 2022'

AND s2.product\_name = Autocad 2023'

AND YEAR(s1.subscription\_start\_date) = 2023

AND YEAR(s2.subscription\_start\_date) = 2023

AND s2.subscription\_start\_date > s1.subscription\_start\_date;