Update of Scope

- added SMS content to the SMS table
- modified send time in SMS and calls to a datetime
- answered support tickets also show a response
- purchase time also includes time of purchase now
- fix email in employee table
- added a check constraint, support tickets cannot be closed without a response

Views and Grants

Customer -
call history - read
SMS history - read
phone plan, data - read
transaction - write, to purchase a new plan.

Sales -
view with top 10 plans and their income
ability to alter plan data

Administrator -
access to defaulter view
permission to activate or deactivate SIMs
access to tower-wise statistics

Employee -
access to their own support tickets - read
ability to answer them - write

Advanced Queries

1. To avail BadaFone's referral offer, the **referrer** must be **called by** the referree. They also want to ensure that both people are in different cities. This programme is only available to Mumbai customers. Make a function to find the city that a phone is currently in, and use it to find customers who may avail this offer.

```
CREATE FUNCTION location_of (s_t bigint)
    RETURNS varchar(50) READS SQL DATA
 RETURN (
   SELECT city
   FROM sim_card INNER JOIN tower
   ON sim_card.current_tower = tower.tower_ID
```

```
    WHERE sim_card.phone_number = s_t
    LIMIT 1
  );

SELECT callee, location_of(callee), location_of(caller) FROM call_table
WHERE location_of(callee) <> location_of(caller)
AND location_of(callee) = 'Mumbai'
```

2. BadaFone thinks that the chance of having a person using excess data depends on their payment method. They want you to verify this claim. Make a function to check whether or not a person is a defaulter, and use this to check the claim.

```
CREATE FUNCTION is_defaulter(phone_num bigint)
    RETURNS tinyint(1) READS SQL DATA
  RETURN (
    SELECT COUNT(*)
    FROM usage_data
      INNER JOIN plan_data p on usage_data.plan_ID = p.plan_ID
      INNER JOIN sim_card sc on usage_data.phone_number = sc.phone_number
    WHERE
      data_used >= data_limit
      AND sc.phone_number = phone_num
  );

SELECT payment_method, COUNT(*) as num_defaulters
FROM wallet
WHERE is_defaulter(phone_number)
GROUP BY payment_method
```

3. Seth from the R&D department has a hypothesis. Messages at 5~7 PM have a higher chance of being read by the receiver than messages sent between 5~7 AM. Sonia argues that the discrepancy is because less messages are sent in the morning. Make a table to evaluate these claims.

```
SELECT HOUR(send_time) AS hour, `read`, COUNT(*) AS quantity
FROM SMS
GROUP BY HOUR(send_time), `read` WITH ROLLUP
```

4. Robin has a hypothesis. Divide the day into four quarters. The first quarter has much less of an SMS count. He says that this pattern doesn't hold for people who are roaming, because they're less likely to be asleep in that time. Make a table to verify this claim.

```
SELECT HOUR(send_time) DIV 6 AS period,
 roaming,
 `read`,
 COUNT(*) as quantity
FROM SMS INNER JOIN sim_card sc on sms.receiver = sc.phone_number
GROUP BY HOUR(send_time) DIV 6, roaming, `read` WITH ROLLUP
```

5. Find the names of top 10 plans that have rolled in the most users, and this valuable information with the sales team.

```
CREATE ROLE sales;

CREATE VIEW top_plans AS
SELECT plan.name, COUNT(*)
FROM subscription INNER JOIN plan ON subscription.plan_ID = plan.plan_ID
GROUP BY subscription.plan_ID
ORDER BY COUNT(*) DESC
LIMIT 10;

GRANT SELECT ON top_plans TO sales;
```

6. Rank each plan by sales, partition them over validity, then perform windowing in each partition so as to make a cumulative distribution table

```
SELECT validity, plan.name,
COUNT(*) * plan.price AS individual_sale,
SUM(COUNT(*) * plan.price) over(
    PARTITION BY validity
    ORDER BY COUNT(*) * plan.price DESC
    ROWS UNBOUNDED PRECEDING
) AS cum_sale
FROM subscription
INNER JOIN plan
    ON subscription.plan_ID = plan.plan_ID
GROUP BY plan.plan_ID
```

7. Cut prices of all plans of the `super` category to 50% for a flash sale, and prices of all `maha` plans to 70%

```
UPDATE plan SET price =
    CASE
```

```
        WHEN plan_name like 'super%'
            THEN 0.5 * price
        WHEN plan_name like 'maha%'
            THEN 0.7 * price
        ELSE
            1 * price
    END
```

8. Badafone plans to migrate employees to its own e-mail domain. But doing it all together would be too hasty. Give up to 50 employees access to this new e-mail domain.

```
SET @PIVOT = (
    WITH rank_table AS (
        SELECT date_of_joining, RANK()
            OVER (ORDER BY date_of_joining) AS seniority
        FROM employee
    )
    SELECT DISTINCT date_of_joining
    FROM rank_table
    WHERE seniority = 50
);
UPDATE employee
    SET e_mail = REPLACE(e_mail, 'gmail.com', 'badafone.in')
    WHERE e_mail LIKE '%gmail.com'
    AND employee_ID IN(
        SELECT employee_ID
        WHERE date_of_joining < @PIVOT
    )
```

9. The ATF has detected a terrorist dogwhistle, `lorem`. They want you to delete any messages with this word sent between any two people through BadaFone's services sent by anyone in four of India's biggest cities.

```
DELETE FROM sms
WHERE (sms_content LIKE '%lorem%')
  AND date > DATE_SUB(CURDATE(), INTERVAL 365 DAY)
  AND sender IN (
      SELECT phone_number
      FROM sim_card
      INNER JOIN tower ON current_tower = tower.tower_ID
      WHERE city IN ('Mumbai', 'Chennai', 'Kolkāta', 'Bangalore')
  )
```

10. Give 100 Rs. cashback to all the users who have used above 24 hours of talk time, total, either as a caller or as a callee.

```
CREATE FUNCTION hourdiff (start_time TIME, end_time TIME) RETURNS int
    RETURN HOUR(ADDTIME(TIMEDIFF(end_time, start_time), '24:00:00')) % 24;

UPDATE wallet
    SET balance = balance + 100
    WHERE phone_number IN
        (SELECT customer FROM
        (SELECT dtt.callee AS customer, time_as_caller + time_as_callee as
         FROM
            (SELECT caller,
            hourdiff(start_time, end_time) as time_as_caller
            FROM call_table
            GROUP BY caller) AS ctt
                INNER JOIN
            (SELECT callee,
            hourdiff(start_time, end_time) as time_as_callee
            FROM call_table
            GROUP BY callee) as dtt
                ON dtt.callee = ctt.caller) as ctc
        WHERE total_calling > 24)
```

Embedded SQL Queries

1. Fetch a user's data (Python, Pandas)

```
pandas.read_sql_query(
            "SELECT * FROM customer NATURAL JOIN usage_calling "
            "WHERE customer_ID = " + str(user_id), connection)
```

2. Fetch number of calls sent and received by each tower (Python, Pandas)

```
pandas.read_sql_query("""SELECT city, received, sent FROM
(SELECT callee_tower as tower, COUNT(*) AS received FROM call_table
GROUP BY callee_tower) as tr
JOIN
(SELECT caller_tower as tower, COUNT(*) AS sent FROM call_table
GROUP BY caller_tower) as ts on tr.tower = ts.tower
JOIN tower ON tower_ID = tr.tower""", connection)
```

3. Set new data for a calling plan (Python, MySQL connector, django)

```
cursor.execute(f"UPDATE plan "
               f"SET validity = {int(data_in['validity']):d}, "
               f"price = {int(data_in['price']):d} "
               f"WHERE `name` = '{data_in['name']:s}';")
```

4. Update the response to a support ticket, and close it if open

```
cursor.execute(f"UPDATE support_ticket SET ticket_response = '{data_in['res
WHERE ticket_ID = {data_in['ID']:d};")
```

There are of course, many more.

Triggers

1. Create a trigger that raises the number of calls by one for both caller and callee when a call is made

```
CREATE TRIGGER called AFTER INSERT ON call_table
FOR EACH ROW
    BEGIN
        UPDATE usage_calling SET calls = CASE
        WHEN phone_number = caller THEN calls + 1
        WHEN phone_number = callee THEN calls + 1
        ELSE calls
    END;
END
```

2. Create a trigger that updates the value of `roaming` cell when the person enters or leaves their home tower.

```
CREATE TRIGGER is_roaming BEFORE UPDATE ON sim_card
FOR EACH ROW
    BEGIN
        SET NEW.roaming =
        IF(NEW.current_tower = NEW.home_tower, 0, 1);
    END
```

3. Create a trigger that deducts money from a person's wallet balance when they purchase a plan.

```
CREATE TRIGGER purchase BEFORE INSERT ON transaction
FOR EACH ROW
    BEGIN
        UPDATE wallet
        INNER JOIN transaction t
            on wallet.phone_number = t.phone_number
        INNER JOIN plan p
            on t.plan_ID = p.plan_ID
        SET balance = balance - p.price
        WHERE wallet.phone_number = NEW.phone_number;
    END
```

4. Create a trigger to assign a support ticket to an employee, provided they don't have any open support tickets assigned to them to begin with.

```
CREATE TRIGGER assign_employee BEFORE INSERT ON support_ticket
FOR EACH ROW
    BEGIN
        SET NEW.employee_ID = (
            SELECT employee.employee_ID FROM employee
                WHERE NOT employee_ID IN (
                    SELECT employee_ID FROM support_ticket
                    WHERE closed = 0
                )
                ORDER BY RAND()
                LIMIT 1
            );
    END
```

Indices

- on aadhaar card in customer, in case someone wants to do lookup by aadhaar card
- in phone number on customer to do lookup on (a single) customer by phone number
- on tower by name to get tower ID from name (e.g. we want a query that finds all SIM cards in mumbai. for this, we would need the ID of mumbai, which is O(n) to seek for without an index. So we also store the reverse of ID→city name mapping)

- on plan table by the name of the plan