Modifications

- 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 some check constraints:

Check Constraints

1. Support tickets cannot be closed without a response
2.

Views and Grants

Customer -
info table: phone number / username / current tower / balance
call history view - read
SMS history view - read
money view - ability to alter balance column, but only if it increases

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

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.

```sql
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.

```sql
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.

```sql
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

Advanced Queries

1. As a part of BadaFone's reference programme, rewards are given to anyone who refers someone to join BadaFone. To avail this offer, the referrer must be called by the referree. To prevent people from gaming the system, they also want to ensure that both people are in different cities. For the first phase, 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 have availed 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's R&D department is once again having a strange idea. They think that the chance of having a person using excess data depends on their payment method. Hector suggests promoting plans with larger data plans to people who use UPI, because they're more likely to exceed their limits. He wants you to verify

his claim. Make a function to check whether or not a person is a defaulter, and use this to check his 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, on the other hand, argues that the discrepancy is only because less messages are sent in the morning, because most people are asleep. The manager asks you to make a table so that they can pass a verdict on Seth's argument.

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

4. Robin looked at this data and got another idea. He thinks you should divide the day into four quarters. If you do so you will notice that the first quarter has much less of an SMS count. He also says that this pattern doesn't hold for people who are roaming. Because they're travelling, they're equally likely to send messages in the morning as they are in the evening. Seth thinks this is nonsense, and asks you to make a table to prove Robin wrong.

```
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. 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
```

6. 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
```