DDL

```
create table sim_card
(
 phone_number       bigint                    not null,
 activated          tinyint(1)    default 0   not null,
 date_of_activation date                      not null,
 home_tower         int                       null,
 current_tower      int                       null,
 roaming            tinyint(1)    default 0   not null,

 primary key(phone_number),

 check (7000000000 <= phone_number and phone_number < 10000000000),

 constraint sim_card_tower_tower_ID_fk
    foreign key (home_tower) references tower (tower_ID)
      on update cascade on delete set null,

 constraint sim_card_tower_tower_ID_fk_2
    foreign key (current_tower) references tower (tower_ID)
      on update cascade on delete set null
);
```

```
create table call_table
(
 caller        bigint        not null,
 callee        bigint        not null,
 start_time    time          not null,
 end_time      time          not null,
 caller_tower  int           not null,
 callee_tower  int           not null,

 primary key (caller, start_time),

 check (end_time > start_time,

 constraint call_table_sim_card_phone_number_fk
        foreign key (caller) references sim_card (phone_number),

 constraint call_table_sim_card_phone_number_fk_2
        foreign key (callee) references sim_card (phone_number),
```

```
 constraint call_table_tower_tower_ID_fk
        foreign key (caller_tower) references tower (tower_ID)
            on update cascade on delete cascade,

 constraint call_table_tower_tower_ID_fk_2
        foreign key (callee_tower) references tower (tower_ID)
            on update cascade on delete cascade
);
```

Basic SELECT

1. Select all phone numbers that have dialled at 4 AM

```
SELECT caller
FROM call_table
WHERE HOUR(start_time) = 4
```

2. Find the number of phone numbers in the same tower region as the phone number 78191502196

```
SELECT COUNT(phone_number)
FROM SIM_Card
WHERE current_tower = (
    SELECT current_tower
        FROM SIM_Card
    WHERE phone_number = 78191502196
)
```

3. Select all calling plans that cost under 500 rupees and last for more than 31 days

```
SELECT *
FROM plan_calling
WHERE price < 500 AND validity > 31
```

JOIN

1. Select all customers with an open support ticket asking for calling help

```
SELECT customer.username
FROM customer
    NATURAL JOIN support_ticket
WHERE category = 'Calling Help' and closed = 0
```

2. Find all customers with names starting with J who purchased their SIM Card in Mumbai

```
SELECT username
FROM (Customer
 INNER JOIN SIM_Card AS T ON Customer.phone_number = T.phone_number)
 INNER JOIN Tower ON Tower.tower_ID = T.home_tower
WHERE username LIKE 'J%'
```

GROUP BY

1. Find the number of plans of the "Maha" series in each validity category, except the fortnightly one.

```
SELECT COUNT(*)
FROM plan
WHERE name like 'maha%'
GROUP BY validity
HAVING validity > 15
```

2. Select the average local minutes used under each yearly calling plan

```
SELECT plan.name, AVG(l_minutes)
FROM usage_calling
INNER JOIN plan ON usage_calling.plan_ID = plan.plan_ID
WHERE validity = 365
GROUP BY plan.plan_ID, plan.name
HAVING plan.name LIKE 'super%'
```

Table Modification

1. 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
```

2. Delete all information in the call usage table, then delete the table itself

```
DELETE FROM call_usage
DROP TABLE call_usage
```

3. Add column content to the SMS table, and remove the column sent_time

```
ALTER TABLE SMS
  ADD content varchar(512)
  DROP sent_time
```

4. Insert an employee's information into the proper table

```
INSERT INTO MY_TABLE(id, first_name, last_name, birth)
VALUES (10001, 'Georgi', 'Facello', 'M');
```

Creating Views

1. Make a view that stores all calling plans that have more than 25 international minutes, and offer SMS facility

```
CREATE VIEW international_special AS
SELECT name
FROM plan_calling
WHERE i_minutes > 25 AND SMS > 0
```

2. Select all users who have used up their data plan. Put this into a view called `defaulters`. Also store the amount of extra data that has been used by the defaulter.

```
CREATE VIEW defaulters AS
 SELECT sim_card.phone_number,
        Data_Usage.data_used - Data_Plan.data_limit AS extra_usage

 FROM usage_data as Data_Usage, plan_data as Data_Plan, SIM_Card
 WHERE Data_Usage.phone_number = SIM_Card.phone_number
        AND Data_Usage.plan_ID = Data_Plan.plan_ID
        AND Data_Usage.data_used >= Data_Plan.data_limit;
```

3. Find the names of top 10 plans that have rolled in the most users, and make this a view called `top_plans`.

```
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;
```

4. Find the total amount of money rolled in by Badafone Inc. from these top plans after deducting 18% GST, and make this into a view called `revenue`.

```
CREATE VIEW revenue AS
SELECT SUM(val) FROM (
 SELECT COUNT(*) * plan.price * 0.82 AS val
   FROM subscription
     INNER JOIN plan ON subscription.plan_ID = plan.plan_ID
   GROUP BY subscription.plan_ID
```

```
    ORDER BY COUNT(*) DESC
    LIMIT 10
) as spv;
```

Authorisation

1. Create role administrator, and allow them to see the `defaulters` view.

```
CREATE ROLE administrator;
GRANT SELECT ON defaulters TO administrator;
```

2. Create role sales, and allow them to see the `top_plans` view

```
CREATE ROLE sales;
GRANT SELECT ON top_plans TO sales;
```

3. Create role accounts, and allow them to see the `revenue` view

```
CREATE ROLE accounts;
GRANT SELECT ON revenue TO accounts;
```

4. Create role plan_managers, and allow them access to see and update the `plan` table. Also grant the sales table access to these tables, so they can adjust the prices as and when needed. Additionally, let them create views for their convenience. Then remove all privileges on this table from all other users. Assign Robin as a plan manager for his great insights so far.

```
GRANT CREATE VIEW, SELECT, UPDATE on TABLE plan to plan_managers;
GRANT CREATE VIEW, SELECT, UPDATE on TABLE plan to sales;
REVOKE all on TABLE plan to PUBLIC;
GRANT plan_manager TO Robin;
```

Functions

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_tabl
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_numb
   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
```

Rollup

1. 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
```

2. 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
```

Ranking and Windowing

1. Rank each plan by the revenue earned through it

```
SELECT plan.plan_ID,
 COUNT(*) * plan.price as revenue,
 RANK() over (ORDER BY COUNT(*) * plan.price DESC) AS 'rank'
```

```
 FROM subscription
 INNER JOIN plan ON subscription.plan_ID = plan.plan_ID
 GROUP BY plan.plan_ID
```

2. Do the same, but for the sale of the plan. If multiple plans have the same rank, the next plan should have rank one less than it, not the default setting.

```
SELECT plan.plan_ID,
COUNT(*) as sale,
DENSE_RANK() over (ORDER BY COUNT(*)) AS 'rank'
FROM subscription
INNER JOIN plan
    ON subscription.plan_ID = plan.plan_ID
GROUP BY plan.plan_ID
```

3. Do the same as (1), but ranking plans in each validity range separately

```
SELECT plan.plan_ID, plan.validity,
       COUNT(*) * plan.price as revenue,
RANK() over (PARTITION BY plan.validity
ORDER BY COUNT(*) * plan.price DESC) AS sale_rank
FROM subscription
INNER JOIN plan
    ON subscription.plan_ID = plan.plan_ID
GROUP BY plan.plan_ID
```

4. Divide the ranking in (1) into 5 ntiles

```
SELECT plan.plan_ID,
COUNT(*) * plan.price as revenue,
ntile(5) over (ORDER BY COUNT(*) * plan.price DESC) AS pentile
FROM subscription
INNER JOIN plan
    ON subscription.plan_ID = plan.plan_ID
GROUP BY plan.plan_ID
```

5. Do the same as (4), but perform windowing in each partition, and make a cumulative distribution table

```
```

```sql
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
```