**1. Write a SQL query that performs the following aggregations per payment_id:**

```
select payment_id,

(select count(payment_id) from PAYMENTS p1 where p1.user_profile_id = p.user_profile_id AND
p1.payment_submit_time < p.payment_submit_time) payment_cnt,

(select count(distinct target_currency_id) from PAYMENTS p1 where
p.user_profile_id=p1.user_profile_id) target_ccy_cnt,

(select count(distinct source_currency_id) from PAYMENTS p1 where
p.user_profile_id=p1.user_profile_id) source_ccy_cnt,

(select DATEDIFF( SYSDATE(), min(payment_submit_time) ) from PAYMENTS p1 where
p1.payment_id = p.payment_id group by p1.payment_id) oldest_payment_age,

IFNULL((select count(*) from PAYMENTS p1 where p1.target_currency_id = p.target_currency_id
and p1.payment_submit_time < p.payment_submit_time ),0) target_ccy_payment_cnt,

IFNULL((select count(*) from PAYMENTS p1 where p1.source_currency_id =
p.source_currency_id and p1.payment_submit_time < p.payment_submit_time ),0)
source_ccy_payments_cnt,

IFNULL((select count(payment_id) from PAYMENTS p1 where p1.source_currency_id =
p.source_currency_id and p1.target_currency_id=p.target_currency_id and p1.user_profile_id =
p.user_profile_id and p1.payment_submit_time < p.payment_submit_time ),0)
same_route_pmnts_cnt,

IFNULL((select count(payment_id) from PAYMENTS p1 where p1.user_profile_id =
p.user_profile_id and p1.source_currency_id = p.target_currency_id and p1.target_currency_id =
p.source_currency_id and p1.payment_submit_time < p.payment_submit_time  ),0)
backward_route_pmnts_cnt
from PAYMENTS p;
```

**2. With focus on query performance, please explain different approaches you would use to make this query faster than a simple self-join. Note that it is OK to provide a solution to part 1 that relies on a self-join, but what else can we do (if anything) to make it performant and be able to execute millions of aggregations in a timely manner?**

For a better query performance, we may think of using sub-queries for each aggregation in the “from” clause with payment_id as the primary key. Then simply, we join the payments table with each custom table and retrieve results.

In that case we run 8 sub-queries containing all the aggregations and 1 final query to the results.

Hence, in the case that if we have 1M records, the query that is written as above in Part 1, will have to run on each of records essentially equivalent to running 8M queries.
However, the computationally efficient concept of the theory presented in Part 2 will be equivalent to running 8 queries and not 8M queries.