QUESTION 1

**Explanation**

To solve the task of identifying customers who have at least one funded savings plan and one funded **investment plan**, I followed a two-step query process using a **Common Table Expression (CTE)** and a final aggregation.   
Step 1:

* Created the **plan\_type** CTE to classify plan types and associate each transaction with its plan type either savings or investment.
* Filtered only plans that have associated **confirmed transactions (confirmed\_amount > 0)** so that will be considered in the next step, ensuring we only process **funded** plans.

Step 2: A final aggregation and selection of targeted customers from **plan\_type** abovejoining with **users\_customuser** table;

* To retrieve customer names using CONCAT() function.
* COUNT(DISTINCT...) to ensure accurate counting of unique plans by type.
* SUMof **confirmed\_amount** to calculate Total inflows, and converted from kobo to naira by dividing by 100. Then, used the ROUND()function in 2 decimal places to match the expected output.
* **HAVING clause** to filter only customers who have at least one savings plan and one investment plan.

**Challenges and Resolution**

* Identifying the correct plan type (savings vs. investment). I used **CASE WHEN** in the CTE to create clearly separable columns savings and investment, based on **is\_regular\_savings** and **is\_a\_fund**.
* Combining customer names correctly. The **name** field in the **users\_customuser** table is empty (all NULL values) but has values for **first\_name** and **last\_name**. I used CONCAT(first\_name, ' ', last\_name) to match the expected name format.
* Filtering for meaningful activity only. I added a **WHERE** **confirmed\_amount > 0** condition to exclude any non-funding or system-generated transactions.
* Currency conversion. I ensured that **confirmed\_amount** in kobo are converted to naira using division by 100

QUESTION 2

**Explanation**

To segment customers by how frequently they transact, I used a two-step CTE-based query:

Step 1: Calculated Customer-Level Transaction Metrics

* Measure transaction volume and the active duration per customer.
* Added **+1** in **TIMESTAMPDIFF** to avoid division by zero when all transactions fall within a single month.
* Grouping by **owner\_id** to isolate each customer.

Step 2: Classified Customers by Frequency Level

* Calculated average monthly frequency by dividing total transactions by months transacted.
* Used CASE logic to categorize customers into High, Medium, or Low frequency buckets.

Step 3: The Final Aggregation

* Counted COUNT() how many customers fall into each frequency category.
* Calculated the AVG() average transaction frequency per group, and rounded into 1 decimal place using the ROUND()function to match the expected output.

**Challenges and Resolution**

* Avoiding divide-by-zero errors when calculating average transactions per month. I added **+1** to **TIMESTAMPDIFF** in case all transactions occurred in a single month.

QUESTION 3

Explanation

The goal of this task was to flag accounts (either savings or investment plans) that have had **no inflow transaction in the last 365 days**. The following steps explain the structure of the query and the logic used:

Step-by-Step Breakdown

* Selectedplan\_id, owner\_id, and the last transaction date for each plan.
* Classified plan type using CASE logic based on flags is\_regular\_savings and is\_a\_fund.
* Calculated inactivity in days using **DATEDIFF(CURRENT\_DATE, last\_transaction\_date)**.
* Performed an inner join between **plans\_plan** and transactions **savings\_savingsaccount** to link deposits to plan types.
* Filtered for actual inflow transactions using confirmed\_amount > 0.
* Grouped by plan to calculate the most recent transaction.
* Filtered to only include plans with no deposits in the last 365 days (HAVING inactivity\_days > 365).
* Finally, sort by most recent transaction to prioritize recent inactivity.

**Challenges and Resolution**

* Avoiding inclusion of never-funded plans. I filtered on **confirmed\_amount > 0** to ensure only funded plans are considered (i.e., those that were once active).
* Avoiding duplicate transactions skewing results. I used **MAX(transaction\_date)** to get the last activity per plan, regardless of the number of transactions.

QUESTION 4

**Explanation**

To estimate **Customer Lifetime Value (CLV)**, the following 3 steps logic explain the structure of the query and the logic CTE used for clarity and modularity:

Step 1:

* Used CONCAT() function to retrieve customer names to match expected output **name**.
* Calculate how long each customer has been active in months using TIMESTAMPDIFF(MONTH, created\_on, CURRENT\_DATE).
* Used TIMESTAMPDIFF(MONTH, created\_on, CURRENT\_DATE) to get monthly tenure.

Step 2:

* Count all confirmed transactions per customer using COUNT(\*).
* Calculate **avg\_profit\_per\_transaction** using **ROUND(AVG(confirmed\_amount) \* 0.001, 2)**; Profit per transaction = 0.1% of value, hence the \* 0.001 multiplier. Then, used the ROUND()function in 2 decimal places to match the expected output.
* With **confirmed\_amount > 0**, only actual inflows are counted.

Step 3: Final Key Field Selection

* Selected key fields, and use the CLV formular to calculate **estimated\_clv.**
* Ensured no divide-by-zero using **WHERE tenure\_months > 0**.
* Formatted output using **ROUND(..., 2)** to align with expected currency formatting.
* Used **ORDER BY estimated\_clv DESC** to sort from highest to low value customers to align with the expected output.

**Challenges and Resolution**

* Avoiding Zero tenure leading to division by zero. I added a filter **WHERE tenure\_months > 0** to prevent errors.