1. Business problem: A report to determine which customers are the most frequent renters AND which customers have generated the highest revenue.

A1.

The detailed table will include the following: CustomerID(int), first\_name(VARCHAR), last\_name(VARCHAR), email(VARCHAR), total\_revenue(INT), rental\_count(INT), address(VARCHAR), city(VARCHAR), country(VARCHAR)

The summary table will include the following: customerid(INT), first\_name(VARCHAR), last\_name(VARCHAR), total\_revenue(INT), rental\_count(INT)

A2. Describe the types of data fields used for the report…. INT, DECIMAL, and VARCHAR

A3. Identify two tables from the dataset that will provide the data needed for the summary/detailed table section of this report…. For the summary AND detailed tables we will be utilizing the customer, payment, and rental tables from the dataset.

A4. Identify *at least* **one**field in the detailed table section that will require a custom transformation with a user-defined function and explain why it should be transformed (e.g., you might translate a field with a value of N to No and Y to Yes)….

(Rework this) While the amount field in the payment table typically stores numeric values representing payment amounts, displaying these values as formatted currency enhances readability and presentation in reports.

A5. Explain the different business uses of the detailed table section and the summary table section of the report….

The summary table will be able to provide us with key metrics such as total revenue generated and total number of rentals for each customer. And we can take this information and perform comparative analysis.

The detailed table will provide us with granular rental and payment details for each customer, transaction level insights and also operational insights which will help us understand the specific interactions customers have with the rental service.

A6. Explain how frequently your report should be refreshed to remain relevant to stakeholders….It depends what the stakeholders intend to do with the information. They could leverage this information to provide a customer reward system to most frequent renters or high revenue generating customers. Or use this information to gauge general customer patterns. In general though I believe this report should be ran weekly to monthly.

Code section:

I regularly used these queries whilst testing

DELETE FROM customer\_summary

SELECT \* FROM customer\_summary

DELETE FROM detailed\_table

SELECT \* FROM detailed\_table

DROP TABLE detailed\_table

DROP TABLE customer\_summary

B. This code will perform the transformation previously mentioned in part A4

-- Create or replace the function to format amount as currency

CREATE OR REPLACE FUNCTION format\_currency(amount numeric)

RETURNS text AS $$

BEGIN

RETURN '$' || TO\_CHAR(amount, 'FM999,999,999,999.99');

END;

$$ LANGUAGE plpgsql;

-- Create the detailed table

CREATE TABLE detailed\_table (

customer\_id INT,

payment\_id INT,

payment\_date DATE,

amount NUMERIC(10, 2),

rental\_id INT,

rental\_date DATE);

-- Create the summary table

CREATE TABLE customer\_summary (

customer\_id INT PRIMARY KEY,

total\_revenue DECIMAL(10, 2) DEFAULT 0.00,

total\_rentals INT DEFAULT 0

);

-- Insert data into detailed\_table from source tables

INSERT INTO detailed\_table (customer\_id, payment\_id, payment\_date, amount, rental\_id, rental\_date)

SELECT

c.customer\_id,

p.payment\_id,

p.payment\_date,

p.amount AS payment\_amount,

r.rental\_id,

r.rental\_date

FROM

customer c

JOIN

payment p ON c.customer\_id = p.customer\_id

JOIN

rental r ON p.rental\_id = r.rental\_id

ORDER BY

c.customer\_id, p.payment\_date, r.rental\_date;

-- insert data into summary\_table from source tables

INSERT INTO customer\_summary (customer\_id, total\_revenue, total\_rentals)

SELECT

p.customer\_id,

SUM(p.amount) AS total\_revenue,

COUNT(r.rental\_id) AS total\_rentals

FROM

payment p

JOIN

rental r ON p.rental\_id = r.rental\_id

GROUP BY

p.customer\_id;

-- Create or replace the trigger function

CREATE OR REPLACE FUNCTION update\_customer\_summary()

RETURNS TRIGGER AS $$

BEGIN

-- Update total revenue and total rentals for the customer in the summary table

UPDATE customer\_summary

SET

total\_revenue = total\_revenue + NEW.amount,

total\_rentals = total\_rentals + 1

WHERE customer\_id = NEW.customer\_id;

-- If customer does not exist in summary table, insert new row

IF NOT FOUND THEN

INSERT INTO customer\_summary (customer\_id, total\_revenue, total\_rentals)

VALUES (NEW.customer\_id, NEW.amount, 1);

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

-- Create the trigger on the detailed table

CREATE TRIGGER update\_customer\_summary\_trigger

AFTER INSERT ON detailed\_table

FOR EACH ROW

EXECUTE FUNCTION update\_customer\_summary();

-- Create or replace the stored procedure

CREATE OR REPLACE PROCEDURE refresh\_detailed\_and\_summary\_tables()

AS $$

BEGIN

-- Clear the contents of the detailed table

TRUNCATE TABLE detailed\_table;

-- Clear the contents of the summary table

TRUNCATE TABLE customer\_summary;

-- Insert raw data into the detailed table

INSERT INTO detailed\_table (customer\_id, payment\_id, payment\_date, amount, rental\_id, rental\_date)

SELECT

c.customer\_id,

p.payment\_id,

p.payment\_date,

p.amount,

r.rental\_id,

r.rental\_date

FROM

customer c

JOIN

payment p ON c.customer\_id = p.customer\_id

JOIN

rental r ON p.rental\_id = r.rental\_id

ORDER BY

c.customer\_id, p.payment\_date, r.rental\_date;

-- Update the summary table based on the newly inserted data

INSERT INTO customer\_summary (customer\_id, total\_revenue, total\_rentals)

SELECT

customer\_id,

SUM(amount) AS total\_revenue,

COUNT(rental\_id) AS total\_rentals

FROM

detailed\_table

GROUP BY

customer\_id

ON CONFLICT (customer\_id)

DO UPDATE SET

total\_revenue = EXCLUDED.total\_revenue,

total\_rentals = EXCLUDED.total\_rentals;

END;

$$ LANGUAGE plpgsql;

CALL refresh\_detailed\_and\_summary\_tables();

SELECT \* FROM detailed\_table

SELECT \* FROM customer\_summary

Finish..

F1.

I would use the PGAGENT tool and have it run either at the end of the day on the last day of the business “week” or end of day end of the month.

G. (Insert video link here)

H. Considering Triggers and procedures were something I do not recall learning in my previous SQL classes I utilized the Postgresql documentation, and ChatGPT.

**NOTE: CHATGPT WAS NOT USED TO WRITE THE CODE FOR THIS PROJECT, IT WAS UTILIZED FOR COMPREHENSION AND ERROR CHECKING.**