Ashton Wray
D326 – Advanced Data Management Performance Assessment

A. Summarize one real-world written business report that can be created from the DVD Dataset from the "Labs on Demand Assessment Environment and DVD Database" attachment.

- Based on the DVD rental ER diagram, I propose a business report focusing on **Monthly DVD Rental Revenue**. This report will utilize data from various tables in the DVD Dataset, including rental, payment, and film tables. Its analysis will provide management with a comprehensive view of the company's monthly revenue trends, helping to inform decisions on inventory management, pricing strategies, and seasonal marketing campaigns.

A1. Identify the specific fields that will be included in the detailed table and the summary table of the report.

Detailed Table:

- `***rental_id***` (from `*rental*`): Unique identifier for each rental transaction.
- `***rental_date***` (from `*rental*`): The rental date to determine the revenue month.
- `***return_date***` (from `*rental*`): When the DVD was returned.
- `***amount***` (from `*payment*`): Revenue from each rental.
- `***customer_id***` (from `*payment*`): To link to customer details.
- `***first_name***`, `***last_name***` (from `*customer*`): Customer name for personalized data tracking.

Summary Table:

- `***rental_date***` (from `*rental*`): Used to extract the month.
- `***total_revenue***`: Sum of `*amount*` from `*payment*` for the month.
- `***total_transactions***`**:** Count of transactions per month.

A2. Describe the types of data fields used for the report.

The Monthly DVD Rental Revenue Report will utilize various SQL data types to store and manipulate the required information. Here's a description of the types of data fields used:

- **INTEGER:** `*rental_id*`, `*customer_id*`
- **TIMESTAMP/DATE:** `*rental_date*`, `*return_date*`
- **VARCHAR:** `*first_name*`, `*last_name*`
- **DECIMAL/NUMERIC:** `*amount*` (for monetary values)

These data types ensure that each information is stored efficiently and can be manipulated accurately in our SQL queries and functions.

A3. Identify at least two specific tables from the given dataset that will provide the data necessary for the report's detailed and summary table sections.

Detailed Table:

- **rental:** Provides `*rental_id*`, `*rental_date*`, and `*return_date*`.
- **payment:** Contains `*payment_id*`, `*customer_id*`, and `*amount*`.

Summary Table:

- **rental**: For aggregation based on `*rental_date*`.
- **payment:** Sum up the `*amount*` to get the `*total_revenue*` and count the payments to get the `*total_transactions*`.

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).

`*rental_date*` in the Summary Table will require transformation to extract just the month and year for summarization. A user-defined function (UDF) can format this date from YYYY-MM-DD to YYYY-MM, simplifying grouping the data by month for revenue calculations.

A5. Explain the different business uses of the report's detailed and summary table sections.

Detailed Table:

- Helps in auditing specific transactions.

- Provides detailed insights into customer rental behavior.
- Useful for detailed financial tracking and individual customer service management.

Summary Table:

- Offers a quick overview of monthly performance.
- Useful for high-level business decisions and trends analysis.
- Simplifies reporting to stakeholders who need to summarize financial data rather than detailed transaction records.

A6. Explain how frequently your report should be refreshed to remain relevant to stakeholders.

Frequency: The report should be refreshed **monthly**. This frequency allows for up-to-date analysis of revenue trends and rental patterns, ensuring timely decision-making for stakeholders.

B. Provide original code for function(s) in text format that performs the transformation(s) you identified in part A4.

```
CREATE OR REPLACE FUNCTION extract_month_year(rental_date TIMESTAMP)
RETURNS VARCHAR AS $$
BEGIN
    RETURN TO_CHAR(rental_date, 'YYYY-MM');
END;
$$ LANGUAGE plpgsql;
```

C. Provide original SQL code in a text format that creates the detailed and summary tables to hold your report table sections.

```
-- Create Detailed Table --
CREATE TABLE detailed_rental_report (
    rental_id INT,
    rental_date DATE,
    return_date DATE,
    customer_id INT,
    customer_name VARCHAR(255),
    amount DECIMAL(10, 2)
);
```

```sql
-- Create Summary Table --
CREATE TABLE summary_rental_report (
    rental_month VARCHAR(7),  -- Format: YYYY-MM
    total_revenue DECIMAL(10, 2),
    total_transactions INT
);
```

D. Provide an original SQL query in a text format that will extract the raw data needed for the detailed section of your report from the source database.

```sql
INSERT INTO detailed_rental_report (rental_id, rental_date, return_date,
customer_id, customer_name, amount)
SELECT
    r.rental_id,
    r.rental_date,
    r.return_date,
    c.customer_id,
    CONCAT(c.first_name, ' ', c.last_name) AS customer_name,
    p.amount
FROM
    rental r
JOIN
    payment p ON r.rental_id = p.rental_id
JOIN
    customer c ON r.customer_id = c.customer_id
ORDER BY
    r.rental_date;
```

E. Provide original SQL code in a text format that creates a trigger on the detailed table of the report that will continually update the summary table as data is added to the detailed table.

```sql
CREATE OR REPLACE FUNCTION update_summary_report()
RETURNS TRIGGER AS $$
```

```
BEGIN
    -- Check if the rental month already exists in the summary table --
    IF EXISTS (SELECT 1 FROM summary_rental_report WHERE rental_month =
TO_CHAR(NEW.rental_date, 'YYYY-MM')) THEN
        -- If it exists, update the total revenue and total transactions for that
month
        UPDATE summary_rental_report
        SET total_revenue = total_revenue + NEW.amount,
            total_transactions = total_transactions + 1
        WHERE rental_month = TO_CHAR(NEW.rental_date, 'YYYY-MM');
    ELSE
        -- If it doesn't exist, insert a new record for that month --
        INSERT INTO summary_rental_report (rental_month, total_revenue,
total_transactions)
        VALUES (TO_CHAR(NEW.rental_date, 'YYYY-MM'), NEW.amount, 1);
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Create Trigger --
CREATE TRIGGER after_insert_detailed_report
AFTER INSERT ON detailed_rental_report
FOR EACH ROW
EXECUTE FUNCTION update_summary_report();
```

F. Provide an original stored procedure in a text format that can be used to refresh the data in *both* the detailed and summary tables. The stored procedure should clear the contents of the detailed and summary tables and perform the raw data extraction from part D.

```
CREATE OR REPLACE PROCEDURE refresh_rental_report()
LANGUAGE plpgsql
AS $$
BEGIN
    -- Temporarily disable the trigger --
    ALTER TABLE detailed_rental_report DISABLE TRIGGER
after_insert_detailed_report;
```

```
    -- Clear the contents of the detailed table --
    TRUNCATE TABLE detailed_rental_report;

    -- Clear the contents of the summary table --
    TRUNCATE TABLE summary_rental_report;

    -- Insert fresh data into the detailed table --
    INSERT INTO detailed_rental_report (rental_id, rental_date, return_date,
customer_id, customer_name, amount)
    SELECT
        r.rental_id,
        r.rental_date,
        r.return_date,
        c.customer_id,
        CONCAT(c.first_name, ' ', c.last_name) AS customer_name,
        p.amount
    FROM
        rental r
    JOIN
        payment p ON r.rental_id = p.rental_id
    JOIN
        customer c ON r.customer_id = c.customer_id;

    -- Insert fresh data into the summary table by aggregating from the detailed
table
    INSERT INTO summary_rental_report (rental_month, total_revenue,
total_transactions)
    SELECT
        TO_CHAR(d.rental_date, 'YYYY-MM') AS rental_month,
        SUM(d.amount) AS total_revenue,
        COUNT(d.rental_id) AS total_transactions
    FROM
        detailed_rental_report d
    GROUP BY
        rental_month
    ORDER BY
        rental_month;

    -- Re-enable the trigger --
    ALTER TABLE detailed_rental_report ENABLE TRIGGER
after_insert_detailed_report;

    RAISE NOTICE 'Rental report data refreshed successfully.';
END;
```

```
$$;
```

F1. Identify a relevant job scheduling tool that can be used to automate the stored procedure.

A relevant job scheduling tool that can automate the stored procedure is pgAgent. pgAgent is a job scheduling agent for PostgreSQL databases designed to run complex schedules and manage recurring tasks.

We will use pgAgent to run our `*refresh_rental_data()*` stored procedure automatically monthly. This ensures that our rental report data is consistently updated without manual intervention. Using pgAgent, we can ensure that our rental report data is refreshed regularly and automatically, maintaining the accuracy and relevance of our business intelligence.

G. Provide a Panopto video recording that includes the presenter and a vocalized demonstration of the code's functionality used for the analysis.

https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=a48ba85b-2b0a-4084-92d9-b1f300eb8747

H. Acknowledge all utilized sources, including any sources of third-party code, using in-text citations and references. If no sources are used, clearly declare that no sources were used to support your submission.

No external sources were used to support this submission. All code and analysis presented in this task were developed independently based on the requirements provided and my existing knowledge of SQL and database management. The examples and data used are fictional and created solely to demonstrate the required functionality for this assessment.