73990 66172

CSN1 — CSN1 TASK 1_ DATA ANALYSIS.docx

Assignment

Class

University

Document Details

Submission ID

trn:oid:::1:2947766605

Submission Date

Jun 14, 2024, 3:25 PM UTC

Download Date

Jun 14, 2024, 3:26 PM UTC

File Name

AgAD_RMAAkgPYFM

File Size

18.5 KB

26 Pages

2,317 Words

15,288 Characters

How much of this submission has been generated by AI?

41%

of qualifying text in this submission has been determined to be generated by AI.

Caution: Percentage may not indicate academic misconduct. Review required.

It is essential to understand the limitations of AI detection before making decisions about a student's work. We encourage you to learn more about Turnitin's AI detection capabilities before using the tool.

Frequently Asked Questions

What does the percentage mean?
The percentage shown in the AI writing detection indicator and in the AI writing report is the amount of qualifying text within the submission that Turnitin's AI writing detection model determines was generated by AI.

Our testing has found that there is a higher incidence of false positives when the percentage is less than 20. In order to reduce the likelihood of misinterpretation, the AI indicator will display an asterisk for percentages less than 20 to call attention to the fact that the score is less reliable.

However, the final decision on whether any misconduct has occurred rests with the reviewer/instructor. They should use the percentage as a means to start a formative conversation with their student and/or use it to examine the submitted assignment in greater detail according to their school's policies.

How does Turnitin's indicator address false positives?
Our model only processes qualifying text in the form of long-form writing. Long-form writing means individual sentences contained in paragraphs that make up a longer piece of written work, such as an essay, a dissertation, or an article, etc. Qualifying text that has been determined to be AI-generated will be highlighted blue on the submission text.

Non-qualifying text, such as bullet points, annotated bibliographies, etc., will not be processed and can create disparity between the submission highlights and the percentage shown.

What does 'qualifying text' mean?
Sometimes false positives (incorrectly flagging human-written text as AI-generated), can include lists without a lot of structural variation, text that literally repeats itself, or text that has been paraphrased without developing new ideas. If our indicator shows a higher amount of AI writing in such text, we advise you to take that into consideration when looking at the percentage indicated.

In a longer document with a mix of authentic writing and AI generated text, it can be difficult to exactly determine where the AI writing begins and original writing ends, but our model should give you a reliable guide to start conversations with the submitting student.

Student's Name

Name of School or Institution

Course Number and Name

Instructor

Date

a. 1. Fields to include in the detailed table.

Monthly DVD Rental Revenue and customer activity report

below is the list of the specific fields to include in the detailed table and summary table

Detailed Table

- Rental ID
- Rental Date
- Return Date
- Customer ID
- Customer Name
- Film Title
- Category
- Rental Duration
- Rental Fee

Summary table

- Total Rentals
- Total Revenue
- Average Rental Duration
- Most Popular Film
- Most Popular Category
- Active Customers (number of customers who rented at least one DVD)

2. Below is a list of the type of data fields used for this report

- **Integer**: Rental ID, Customer ID, Total Rentals, Active Customers
- **Date/Time**: Rental Date, Return Date

- **Text/String**: Customer Name, Film Title, Category, Most Popular Film, Most Popular Category
- **Numeric**: Rental Duration (calculated in days), Rental Fee, Total Revenue, Average Rental Duration

3. **Specific tables from the given dataset to provide the data necessary for detailed table section and summary table.**

- **rental**: This table provides details on each DVD rental, including Rental ID, Rental Date, Return Date, and Customer ID.
- **customer**: This table provides customer details, including Customer ID and Customer Name.
- **film**: This table provides film details, including Film Title and Category.

4. **Customer Transformation with a User-Defined Function**

Field: Rental Duration

Transformation Explanation: The Rental Duration will be calculated by subtracting the Rental Date from the Return Date. This requires a custom function to ensure the duration is accurately represented in days, especially handling cases where the return date might be null (indicating a DVD not yet returned).

5. Business Uses of the Detailed Table and Summary Table

Detailed Table:

Use: The detailed table provides granular data on each rental transaction. It is useful for identifying trends in rental behaviors, monitoring individual customer activities, and managing inventory by tracking the popularity of specific films and categories.

Summary Table:

Use: The summary table offers high-level insights into the overall performance of the rental business. It helps stakeholders understand the financial health of the business by showing total revenue, average rental duration, and customer engagement metrics. This information is crucial for making strategic decisions, such as marketing campaigns for popular categories or films, and identifying areas for improvement.

Report Refresh Frequency

Refresh Frequency: Monthly

Explanation: Given that this report is intended to provide monthly insights into rental revenue and customer activity, it should be refreshed at the beginning of each month. This frequency ensures that the data is up-to-date and relevant for stakeholders, allowing them to make timely and informed decisions based on the most recent month's performance.

b. Original code for the custom function that calculates the rental duration which was identified in part A4

```
CREATE OR REPLACE FUNCTION calculate_rental_duration(rental_date DATE, return_date DATE)
RETURNS INTEGER AS $$
BEGIN
  IF return_date IS NULL THEN
    RETURN CURRENT_DATE - rental_date;
  ELSE
    RETURN return_date - rental_date;
```

```
    END IF;
END;
$$ LANGUAGE plpgsql;
```

c. **Below is the SQL code to create a detailed and summary table to hold the necessary information**

Detailed table

```
CREATE TABLE detailed_rental_report (
    rental_id INTEGER PRIMARY KEY,
    rental_date DATE NOT NULL,
    return_date DATE,
    customer_id INTEGER NOT NULL,
    customer_name VARCHAR(255) NOT NULL,
    film_title VARCHAR(255) NOT NULL,
    category VARCHAR(255) NOT NULL,
    rental_duration INTEGER,
    rental_fee NUMERIC(5,2) NOT NULL
);
```

SQL code to create summary table

```
CREATE TABLE summary_rental_report (
    report_month DATE PRIMARY KEY,
    total_rentals INTEGER NOT NULL,
    total_revenue NUMERIC(10,2) NOT NULL,
```

```
    average_rental_duration NUMERIC(5,2) NOT NULL,
    most_popular_film VARCHAR(255) NOT NULL,
    most_popular_category VARCHAR(255) NOT NULL,
    active_customers INTEGER NOT NULL
);
```

SQL code to insert Data into detailed table

```
INSERT INTO detailed_rental_report (rental_id, rental_date, return_date, customer_id,
customer_name, film_title, category, rental_duration, rental_fee)
SELECT
    r.rental_id,
    r.rental_date,
    r.return_date,
    c.customer_id,
    c.first_name || ' ' || c.last_name AS customer_name,
    f.title AS film_title,
    f.category,
    calculate_rental_duration(r.rental_date, r.return_date) AS rental_duration,
    r.rental_rate AS rental_fee
FROM
    rental r
JOIN
    customer c ON r.customer_id = c.customer_id
```

```
JOIN
    film f ON r.film_id = f.film_id;
```

SQL code to insert data into summary table

```
INSERT INTO summary_rental_report (report_month, total_rentals, total_revenue, average_rental_duration, most_popular_film, most_popular_category, active_customers)
SELECT
    DATE_TRUNC('month', r.rental_date) AS report_month,
    COUNT(r.rental_id) AS total_rentals,
    SUM(r.rental_rate) AS total_revenue,
    AVG(calculate_rental_duration(r.rental_date, r.return_date)) AS average_rental_duration,
    (
        SELECT f.title
        FROM rental r2
        JOIN film f ON r2.film_id = f.film_id
        GROUP BY f.title
        ORDER BY COUNT(r2.rental_id) DESC
        LIMIT 1
    ) AS most_popular_film,
    (
        SELECT f.category
        FROM rental r2
```

```
        JOIN film f ON r2.film_id = f.film_id
        GROUP BY f.category
        ORDER BY COUNT(r2.rental_id) DESC
        LIMIT 1
    ) AS most_popular_category,
    COUNT(DISTINCT r.customer_id) AS active_customers
FROM
    rental r
GROUP BY
    report_month;
```

d. Below is the SQL query to extract the raw data needed for the detailed section of the report from the source database.

```
SELECT
    r.rental_id,
    r.rental_date,
    r.return_date,
    c.customer_id,
    c.first_name || ' ' || c.last_name AS customer_name,
    f.title AS film_title,
    cat.name AS category,
    calculate_rental_duration(r.rental_date, r.return_date) AS rental_duration,
    r.rental_rate AS rental_fee
FROM
```

```
    rental r
JOIN
    customer c ON r.customer_id = c.customer_id
JOIN
    inventory i ON r.inventory_id = i.inventory_id
JOIN
    film f ON i.film_id = f.film_id
JOIN
    film_category fc ON f.film_id = fc.film_id
JOIN
    category cat ON fc.category_id = cat.category_id;
```

e. Below is the SQL code to create a trigger on the detailed table that will continually update the summary table as data is added to the detailed table.

```
CREATE OR REPLACE FUNCTION update_summary_rental_report()
RETURNS TRIGGER AS $$
BEGIN
    -- Update total rentals and total revenue
    UPDATE summary_rental_report
    SET
        total_rentals = total_rentals + 1,
        total_revenue = total_revenue + NEW.rental_fee,
        average_rental_duration = (
```

```
        (average_rental_duration * (total_rentals - 1) + NEW.rental_duration) /
total_rentals
      ),
      active_customers = (
        SELECT COUNT(DISTINCT customer_id)
        FROM detailed_rental_report
        WHERE DATE_TRUNC('month', rental_date) = DATE_TRUNC('month',
NEW.rental_date)
      )
  WHERE
    report_month = DATE_TRUNC('month', NEW.rental_date);

  -- Insert new month record if not exists
  IF NOT FOUND THEN
    INSERT INTO summary_rental_report (
      report_month, total_rentals, total_revenue, average_rental_duration,
most_popular_film, most_popular_category, active_customers
    )
    VALUES (
      DATE_TRUNC('month', NEW.rental_date), 1, NEW.rental_fee,
NEW.rental_duration, '', '', 1
    );
  END IF;
```

```
-- Update most popular film and category
UPDATE summary_rental_report
SET
    most_popular_film = (
        SELECT f.title
        FROM detailed_rental_report dr
        JOIN film f ON dr.film_title = f.title
        WHERE DATE_TRUNC('month', dr.rental_date) = report_month
        GROUP BY f.title
        ORDER BY COUNT(dr.rental_id) DESC
        LIMIT 1
    ),
    most_popular_category = (
        SELECT c.name
        FROM detailed_rental_report dr
        JOIN film f ON dr.film_title = f.title
        JOIN film_category fc ON f.film_id = fc.film_id
        JOIN category c ON fc.category_id = c.category_id
        WHERE DATE_TRUNC('month', dr.rental_date) = report_month
        GROUP BY c.name
        ORDER BY COUNT(dr.rental_id) DESC
        LIMIT 1
```

```
    )
  WHERE
    report_month = DATE_TRUNC('month', NEW.rental_date);

  RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Below is the trigger that will call the above function whenever a new row is inserted into the detailed table

```
CREATE TRIGGER update_summary_trigger
AFTER INSERT ON detailed_rental_report
FOR EACH ROW
EXECUTE FUNCTION update_summary_rental_report();
```

f. Below is the SQL code for a stored procedure that refreshes the data in both the detailed and summary table.

```
CREATE OR REPLACE PROCEDURE refresh_rental_report()
LANGUAGE plpgsql
AS $$
BEGIN
  -- Clear contents of the detailed table
```

```
    TRUNCATE TABLE detailed_rental_report;

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

    -- Insert new data into the detailed table
    INSERT INTO detailed_rental_report (rental_id, rental_date, return_date, customer_id,
customer_name, film_title, category, rental_duration, rental_fee)
    SELECT
        r.rental_id,
        r.rental_date,
        r.return_date,
        c.customer_id,
        c.first_name || ' ' || c.last_name AS customer_name,
        f.title AS film_title,
        cat.name AS category,
        calculate_rental_duration(r.rental_date, r.return_date) AS rental_duration,
        r.rental_rate AS rental_fee
    FROM
        rental r
    JOIN
        customer c ON r.customer_id = c.customer_id
    JOIN
```

```
    inventory i ON r.inventory_id = i.inventory_id
  JOIN
    film f ON i.film_id = f.film_id
  JOIN
    film_category fc ON f.film_id = fc.film_id
  JOIN
    category cat ON fc.category_id = cat.category_id;


  -- Insert new data into the summary table
  INSERT INTO summary_rental_report (report_month, total_rentals, total_revenue, average_rental_duration, most_popular_film, most_popular_category, active_customers)
  SELECT
    DATE_TRUNC('month', r.rental_date) AS report_month,
    COUNT(r.rental_id) AS total_rentals,
    SUM(r.rental_rate) AS total_revenue,
    AVG(calculate_rental_duration(r.rental_date, r.return_date)) AS average_rental_duration,
    (
      SELECT f.title
      FROM rental r2
      JOIN film f ON r2.film_id = f.film_id
      GROUP BY f.title
      ORDER BY COUNT(r2.rental_id) DESC
```

```
        LIMIT 1
    ) AS most_popular_film,
    (
        SELECT c.name
        FROM rental r2
        JOIN film f ON r2.film_id = f.film_id
        JOIN film_category fc ON f.film_id = fc.film_id
        JOIN category c ON fc.category_id = c.category_id
        GROUP BY c.name
        ORDER BY COUNT(r2.rental_id) DESC
        LIMIT 1
    ) AS most_popular_category,
    COUNT(DISTINCT r.customer_id) AS active_customers
FROM
    rental r
GROUP BY
    report_month;
END;
$$;
```

Job Scheduling Tool: pg_cron

A relevant job scheduling tool that can be used to automate the execution of this stored procedure is ***pg_cron***. ***pg_cron*** is a PostgreSQL extension for running periodic jobs. It allows you to schedule PostgreSQL commands directly from the database.

Script

Below is the transcript for video recording demonstrating the functionality of the code used for the analysis.

Monthly DVD Rental Revenue and Customer Activity Report

Hello, and welcome to this demonstration on generating a Monthly DVD Rental Revenue and Customer Activity Report using PostgreSQL. Today, we'll walk through the process of setting up the necessary tables, creating functions and triggers, and running a stored procedure to refresh our report data. Let's get started!

In this presentation we will discuss the following:

- Introduction to the DVD Rental Database
- Creating the Detailed and Summary Tables
- Custom Function for Rental Duration
- Inserting Data into the Tables
- Creating and Using a Trigger
- Refreshing the Report Data
- Automating the Process with pg_cron

Introduction to the DVD Rental Database

Our DVD Rental Database contains tables for rentals, customers, films, and categories. We will use this data to generate detailed and summary reports.

Now let us create a detailed and summary tables

```
CREATE TABLE detailed_rental_report (
    rental_id INTEGER PRIMARY KEY,
    rental_date DATE NOT NULL,
    return_date DATE,
    customer_id INTEGER NOT NULL,
    customer_name VARCHAR(255) NOT NULL,
    film_title VARCHAR(255) NOT NULL,
    category VARCHAR(255) NOT NULL,
    rental_duration INTEGER,
    rental_fee NUMERIC(5,2) NOT NULL
);

CREATE TABLE summary_rental_report (
    report_month DATE PRIMARY KEY,
    total_rentals INTEGER NOT NULL,
    total_revenue NUMERIC(10,2) NOT NULL,
    average_rental_duration NUMERIC(5,2) NOT NULL,
    most_popular_film VARCHAR(255) NOT NULL,
    most_popular_category VARCHAR(255) NOT NULL,
    active_customers INTEGER NOT NULL
);
```

Next we need a function to calculate the rental duration. This function will handle cases where the return date might be null.

```
CREATE OR REPLACE FUNCTION calculate_rental_duration(rental_date DATE, return_date DATE)
RETURNS INTEGER AS $$
BEGIN
    IF return_date IS NULL THEN
        RETURN CURRENT_DATE - rental_date;
    ELSE
        RETURN return_date - rental_date;
    END IF;
END;
$$ LANGUAGE plpgsql;
```

Now let us insert data into our detailed table using a query that joins multiple tables and uses our custom function.

```
INSERT INTO detailed_rental_report (rental_id, rental_date, return_date, customer_id, customer_name, film_title, category, rental_duration, rental_fee)
SELECT
    r.rental_id,
    r.rental_date,
    r.return_date,
    c.customer_id,
```

```
    c.first_name || ' ' || c.last_name AS customer_name,
    f.title AS film_title,
    cat.name AS category,
    calculate_rental_duration(r.rental_date, r.return_date) AS rental_duration,
    r.rental_rate AS rental_fee
FROM
    rental r
JOIN
    customer c ON r.customer_id = c.customer_id
JOIN
    inventory i ON r.inventory_id = i.inventory_id
JOIN
    film f ON i.film_id = f.film_id
JOIN
    film_category fc ON f.film_id = fc.film_id
JOIN
    category cat ON fc.category_id = cat.category_id;
```

To keep our summary table updated, we create a trigger that updates it whenever a new row is inserted into the detailed table.

```
CREATE OR REPLACE FUNCTION update_summary_rental_report()
RETURNS TRIGGER AS $$
BEGIN
```

```
    -- Update total rentals and total revenue
    UPDATE summary_rental_report
    SET
        total_rentals = total_rentals + 1,
        total_revenue = total_revenue + NEW.rental_fee,
        average_rental_duration = (
            (average_rental_duration * (total_rentals - 1) + NEW.rental_duration) / total_rentals
        ),
        active_customers = (
            SELECT COUNT(DISTINCT customer_id)
            FROM detailed_rental_report
            WHERE DATE_TRUNC('month', rental_date) = DATE_TRUNC('month', NEW.rental_date)
        )
    WHERE
        report_month = DATE_TRUNC('month', NEW.rental_date);

    -- Insert new month record if not exists
    IF NOT FOUND THEN
        INSERT INTO summary_rental_report (
            report_month, total_rentals, total_revenue, average_rental_duration, most_popular_film, most_popular_category, active_customers
        )
```

```
        VALUES (
            DATE_TRUNC('month', NEW.rental_date), 1, NEW.rental_fee, NEW.rental_duration, '',
'', 1
        );
    END IF;

    -- Update most popular film and category
    UPDATE summary_rental_report
    SET
        most_popular_film = (
            SELECT f.title
            FROM detailed_rental_report dr
            JOIN film f ON dr.film_title = f.title
            WHERE DATE_TRUNC('month', dr.rental_date) = report_month
            GROUP BY f.title
            ORDER BY COUNT(dr.rental_id) DESC
            LIMIT 1
        ),
        most_popular_category = (
            SELECT c.name
            FROM detailed_rental_report dr
            JOIN film f ON dr.film_title = f.title
            JOIN film_category fc ON f.film_id = fc.film_id
```

```
        JOIN category c ON fc.category_id = c.category_id
        WHERE DATE_TRUNC('month', dr.rental_date) = report_month
        GROUP BY c.name
        ORDER BY COUNT(dr.rental_id) DESC
        LIMIT 1
    )
  WHERE
    report_month = DATE_TRUNC('month', NEW.rental_date);

  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER update_summary_trigger
AFTER INSERT ON detailed_rental_report
FOR EACH ROW
EXECUTE FUNCTION update_summary_rental_report();
```

To refresh our report data, we'll use a stored procedure that clears both tables and re-populates them.

```
CREATE OR REPLACE PROCEDURE refresh_rental_report()
LANGUAGE plpgsql
AS $$
```

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

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

    -- Insert new data into the detailed table
    INSERT INTO detailed_rental_report (rental_id, rental_date, return_date, customer_id, customer_name, film_title, category, rental_duration, rental_fee)
    SELECT
        r.rental_id,
        r.rental_date,
        r.return_date,
        c.customer_id,
        c.first_name || ' ' || c.last_name AS customer_name,
        f.title AS film_title,
        cat.name AS category,
        calculate_rental_duration(r.rental_date, r.return_date) AS rental_duration,
        r.rental_rate AS rental_fee
    FROM
        rental r
    JOIN
```

```
    customer c ON r.customer_id = c.customer_id
JOIN
    inventory i ON r.inventory_id = i.inventory_id
JOIN
    film f ON i.film_id = f.film_id
JOIN
    film_category fc ON f.film_id = fc.film_id
JOIN
    category cat ON fc.category_id = cat.category_id;

-- Insert new data into the summary table
INSERT INTO summary_rental_report (report_month, total_rentals, total_revenue,
average_rental_duration, most_popular_film, most_popular_category, active_customers)
SELECT
    DATE_TRUNC('month', r.rental_date) AS report_month,
    COUNT(r.rental_id) AS total_rentals,
    SUM(r.rental_rate) AS total_revenue,
    AVG(calculate_rental_duration(r.rental_date, r.return_date)) AS average_rental_duration,
    (
        SELECT f.title
        FROM rental r2
        JOIN film f ON r2.film_id = f.film_id
        GROUP BY f.title
```

```
        ORDER BY COUNT(r2.rental_id) DESC
        LIMIT 1
    ) AS most_popular_film,
    (
        SELECT c.name
        FROM rental r2
        JOIN film f ON r2.film_id = f.film_id
        JOIN film_category fc ON f.film_id = fc.film_id
        JOIN category c ON fc.category_id = c.category_id
        GROUP BY c.name
        ORDER BY COUNT(r2.rental_id) DESC
        LIMIT 1
    ) AS most_popular_category,
    COUNT(DISTINCT r.customer_id) AS active_customers
FROM
    rental r
GROUP BY
    report_month;
END;
$$;
```

Finally, we can automate this process using *pg_cron*, a PostgreSQL extension for scheduling jobs.

```
-- Load the pg_cron extension
CREATE EXTENSION pg_cron;

-- Schedule the stored procedure to run at midnight on the first day of every month
SELECT cron.schedule('0 0 1 * *', $$CALL refresh_rental_report();$$);
```

Now with that we have successfully set up a process to generate and refresh our monthly DVD rental revenue and customer activity report. Thank you for watching this demonstration.