# A: Summarize one real-world business report.

What movie genre is most popular, and which brings in the most money?

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

Detailed – dvdrentals.transactions

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Description |
| Payment\_id | Integer | Primary Key |
| Date | Date | Date of transaction |
| Amount | Numeric (5,2) | Total amount of transaction |
| Name | Character varying (25) | Name of the genre |
| Is\_kidfriendly | Boolean | Whether or not the rating is 'G', 'PG', or 'PG-13' |

Summary – dvdrentals.genre\_agg

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Description |
| Genre\_name\_id | Character varying (25) | Primary Key – Name of Genre |
| Rental\_count | Bigint | Number of rentals |
| Amount | Numeric (5,2) | Sum of the transactions for the genre |
| Kidfriendly\_count | Bigint | Count of the rentals that were kid friendly |
| Not\_kidfriendly\_count | Bigint | Count of the rentals that were not fid friendly |
| Rental\_order | Bigint | Ordinal – The order of the genres by the count of rentals |
| Amount\_order | Bigint | Ordinal – The order of the genres by the sum of their transactions |

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

The main data used for the report is the category names and the sum amount of their transactions as well as a count of the transactions for each genre.

Please see A1 for the types of data fields.

## A3: Identify at least two specific tables from the given dataset that will provide the data necessary for the detailed table section and the summary table section of the report.

The data used for the report is from the tables ‘payment,’ ‘category,’ and ‘film.’ However, because some of these tables were 5 relational steps away, the data had to be bound through ‘rental,’ ‘inventory,’ and ‘film\_category.’

## 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.

The field that transformed was ‘dvdrentals.transactions.is\_kidfriendly’.

The column was transformed into a Boolean using a CASE statement to determine whether or not the value was ‘kid friendly,’ i.e. was rated ‘G,’ ‘PG,’ or ‘PG-13’. This is done for future analysis on which genre is dominated by kid movies or adult movies.

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

Detailed – ‘dvdrentals.transactions’

This would allow you to create a historical time-series analysis to determine which genres and ratings were popular during which epochs. Thus, allowing you to better target your advertisements during said epochs.

Summary – ‘dvdrentals.genre\_agg’

This report would be used to show what the currently highest genres are by rental count and rental amount. Ultimately, this would be aggregated by whatever epoch the business need, e.g. month, quarter, holiday sectionals. It would also be used to verify the efficacy of the actions taken by the analysis done on the detailed report. similar refresh rate. However, because of common business key performance indicators focusing of months, e.g. Month-To-Date and Previous-Month-To-Date reports, the minimum refresh rate would need to be Monthly.

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

These reports would need to be updated based on which way the business plans to use them. If the detailed table is used for time-series analysis and prediction models using something similar to the Seasonal Autoregressive Integrated Moving Average model, then having the data almost live would be preferable but not necessary. The slowest refresh time necessary would be monthly or whatever seasonality the SAIRMA model defines as the smallest seasons. The information for the detailed report would have a similar refresh rate. However, because of common business key performance indicators focusing of months, e.g. Month-To-Date and Previous-Month-To-Date reports, the minimum refresh rate would need to be Monthly.

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

The code provided below creates a function to determine if a movie is kid-friendly or not. This would be used to determine how profitable it is to rent out movies to kids in the future.

/\* Create\_function\_is\_kidfriendly.sql

 Author: Dylan Young

 Email: dylanyoung3244@gmail.com

 Purpose: This file creates a function call 'is\_kidfriendly'

 - that returns true if the rating is 'G', 'PG', or 'PG-13'

 .Notes

  Version:        1.0

  Author:         Dylan Young

  Creation Date:  6/12/2024

  Purpose/Change: Initial script development. Minimal viable product acheived.

  Tested:         Sucessful in both FBI-Van & WGU's VDI instance name DESKTOP-BT4131D

\*/

CREATE OR REPLACE FUNCTION is\_kidfriendly(rating varchar(5))

    RETURNS boolean

    AS $$

    BEGIN

    CASE rating

        WHEN 'G' THEN RETURN TRUE;

        WHEN 'PG' THEN RETURN TRUE;

        WHEN 'PG-13' THEN RETURN TRUE;

        ELSE RETURN FALSE;

    END CASE;

    END

    $$ LANGUAGE plpgsql; -- Says the function is implemented in the plpgsql language. Sometimes it was being sassy with different envronments.

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

## Detailed table

/\* Create\_Transactions.sql

.Author: Dylan Young

.Email: dylanyoung3244@gmail.com

.Purpose: Creating a detailed table to show each transaction

 - by de-normalizing data to provide an easier view.

 - Includes he date, price, genre, and whether or not the dvd

 - was kid friendly.

.Tables: Output table contains data de-normalized from the following tables

 - dvdrental.payment

 - dvdrental.category

 - dvdrental.film

                        Table "dvdrental.transactions"

    Column          |         Type          | Description

    ----------------+-----------------------+-----------------------------------------

    payment\_id      | integer               | Primary Key

    date            | date                  | Date of the transaction

    amount          | numeric (5,2)         | Total amount of the transaction

    name            | character varying (25)| Name of the genre of the DVD

    is\_kidfriendly  | boolean               | Whether or not the rating is 'G', 'PG', or 'PG-13'

.Notes

  Version:        1.0

  Author:         Dylan Young

  Creation Date:  9/26/2022

  Purpose/Change: Initial script development. Minimal viable product acheived.

  Tested:         Sucessful in both HCLS-PostgreSQL01 & WGU's VDI instance name DESKTOP-BT4131D

.Notes

  Version:        1.1

  Author:         Dylan Young

  Creation Date:  6/17/2024

  Purpose/Change: Minimal changes due to retaking the classe.

  Tested:         Sucessful in both FBI-Van & WGU's VDI instance name DESKTOP-BT4131D

.Notes

  Version:        1.2

  Author:         Dylan Young

  Creation Date:  6/21/2024

  Purpose/Change: Switched to using the 'is\_kidfriendly' function

  Tested:         Sucessful in both FBI-Van & WGU's VDI instance name DESKTOP-BT4131D

\*/

DROP TABLE IF EXISTS transactions;

CREATE TABLE transactions (

    payment\_id integer,

    date DATE,

    amount numeric(5,2),

    name varchar(25),

    is\_kidfriendly boolean

);

INSERT INTO transactions

SELECT payment.payment\_id,

    DATE(payment.payment\_date),

    payment.amount,

    category.name,

    --This is the transformation field

    is\_kidfriendly(film.rating) as is\_kidfriendly

FROM payment

/\*

\* The 'category' table is 5 steps away from the 'payment' table

\* payment >> rental >> inventory >> film >> film\_category >> category

\*/

LEFT Join rental ON rental.rental\_id = payment.rental\_id

LEFT Join inventory ON inventory.inventory\_id = rental.inventory\_id

LEFT Join film ON film.film\_id = inventory.film\_id

LEFT Join film\_category ON film\_category.film\_id = film.film\_id

LEFT Join category ON category.category\_id = film\_category.category\_id

## Summary Table

/\* Create\_GenreAg.sql

.Author: Dylan Young

 Email: dylanyoung3244@gmail.com

.Purpose: Creating a table with the the following information for each genre

 - Genre name, rental count, amount of profit the genre generated,

 - how many were and were not kid frienly, as well as their order by

 - rental count and amount of profit generated.

.Tables: Output table contains data originating from the following tables

 - dvdrental.transactions

                        Table "dvdrental.genre\_agg"

    Column                  |         Type          | Description

    ------------------------+-----------------------+----------------------------------------------------------------

    genre\_name              | character varying (25)| Primary Key - Name of the Genre

    rental\_count            | bigint                | Number of rentals

    amount                  | bigint                | Sum of the transaction for the genre

    kidfriendly\_count       | bigint                | Count of rentals that were kid friendly

    not\_kidfriendly\_count   | bigint                | Count of rentals that were not kid friendly

    rental\_order            | bigint                | Ordinal - The order of the genres by the count of rentals

    amount\_order            | bigint                | Ordinal - The order of the genres by the sum of the transaction amounts

.Notes

  Version:        1.0

  Author:         Dylan Young

  Creation Date:  9/26/2022

  Purpose/Change: Initial script development. Minimal viable product acheived.

  Tested:         Sucessful in both HCLS-PostgreSQL01 & VDI DESKTOP-BT4131D

.Notes

  Version:        1.1

  Author:         Dylan Young

  Creation Date:  10/13/2022

  Purpose/Change: Fixed issues mentioned in feedback.

  Tested:         Sucessful in both HCLS-PostgreSQL01 & VDI DESKTOP-BT4131D

.Notes

  Version:        1.2

  Author:         Dylan Young

  Creation Date:  6/18/2024

  Purpose/Change: Updated for new version of coursework

  Tested:         Sucessful in both FBI-Van & VDI DESKTOP-BT4131D

\*/

DROP TABLE IF EXISTS genre\_agg;

CREATE TABLE genre\_agg (

    genre\_name varchar(25),

    rental\_count bigint,

    amount bigint,

    kidfriendly\_count bigint,

    not\_kidfriendly\_count bigint,

    rental\_order bigint,

    amount\_order bigint

);

INSERT INTO genre\_agg

SELECT transactions.name

        AS genre\_name,

    COUNT(transactions.payment\_id)

        AS rental\_count,

    SUM(transactions.amount)

        AS amount,

    COUNT(\*) FILTER (WHERE transactions.is\_kidfriendly = TRUE)

        AS kidfriendly\_count,

    COUNT(\*) FILTER (WHERE transactions.is\_kidfriendly = FALSE)

        AS not\_kidfriendly\_count,

    RANK() OVER (ORDER BY COUNT(transactions.payment\_id) DESC)

        AS rental\_order,

    RANK() OVER (ORDER BY SUM(transactions.amount) DESC)

        AS amount\_order

FROM transactions

GROUP BY transactions.name

# 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.

The code listed in part C for the detailed table includes the SQL query that extracts the raw data.

Please see Part C.

# 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.

## Update Summary Table When Data is Added to the Detailed Table

/\* Trigger\_GenreAgg.sql

.Author: Dylan Young

 Email: dylanyoung3244@gmail.com

.Purpose: Trigger for creating a table with the the following information for each genre

 - Genre name, rental count, amount of profit the genre generated,

 - how many were and were not kid frienly, as well as their order by

 - rental count and amount of profit generated.

.Tables: Output table contains data originating from the following tables

 - dvdrental.transactions

                        Table "dvdrental.genre\_agg"

    Column                  |         Type          | Description

    ------------------------+-----------------------+----------------------------------------------------------------

    genre\_name              | character varying (25)| Primary Key - Name of the Genre

    rental\_count            | bigint                | Number of rentals

    amount                  | bigint                | Sum of the transaction for the genre

    kidfriendly\_count       | bigint                | Count of rentals that were kid friendly

    not\_kidfriendly\_count   | bigint                | Count of rentals that were not kid friendly

    rental\_order            | bigint                | Ordinal - The order of the genres by the count of rentals

    amount\_order            | bigint                | Ordinal - The order of the genres by the sum of the transaction amounts

.Notes

  Version:        1.0

  Author:         Dylan Young

  Creation Date:  9/26/2022

  Purpose/Change: Initial script development. Minimal viable product acheived.

  Tested:         Sucessful in both HCLS-PostgreSQL01 & VDI DESKTOP-BT4131D

.Notes

  Version:        1.1

  Author:         Dylan Young

  Creation Date:  10/13/2022

  Purpose/Change: Fixed issues mentioned in feedback.

  Tested:         Sucessful in both HCLS-PostgreSQL01 & VDI DESKTOP-BT4131D

.Notes

  Version:        1.2

  Author:         Dylan Young

  Creation Date:  6/18/2024

  Purpose/Change: Updated for new version of coursework

  Tested:         Sucessful in both FBI-Van & VDI DESKTOP-BT4131D

 \*/

CREATE OR REPLACE FUNCTION Update\_GenreAgg()

    RETURNS trigger

    AS $$

    --LANGUAGE plpgsql VOLATILE

    BEGIN

    DROP TABLE IF EXISTS genre\_agg;

    INSERT INTO public.genre\_agg

    SELECT transactions.name

            AS genre\_name,

        COUNT(transactions.payment\_id)

            AS rental\_count,

        SUM(transactions.amount)

            AS amount,

        COUNT(\*) FILTER (WHERE transactions.is\_kidfriendly = TRUE)

            AS kidfriendly\_count,

        COUNT(\*) FILTER (WHERE transactions.is\_kidfriendly = FALSE)

            AS not\_kidfriendly\_count,

        RANK() OVER (ORDER BY COUNT(transactions.payment\_id) DESC)

            AS rental\_order,

        RANK() OVER (ORDER BY SUM(transactions.amount) DESC)

            AS amount\_order

    FROM transactions

    GROUP BY transactions.name

    --RETURN NEW

;

END;

$$

LANGUAGE plpgsql VOLATILE -- Says the function is implemented in the plpgsql language; VOLATILE says the function has side effects.

COST 100; -- Estimated execution cost of the function.

CREATE Trigger GenreAgg\_update

    AFTER INSERT

    ON transactions

    FOR EACH ROW

    EXECUTE PROCEDURE Update\_GenreAgg();

## Update Detailed Table When Data is Added to Payment Table

/\* Trigger\_Transactions.sql

 Author: Dylan Young

 Email: dylanyoung3244@gmail.com

.Purpose: Trigger for creating a detailed table to show each transaction

 - by de-normalizing data to provide an easier view.

 - Includes he date, price, genre, and whether or not the dvd

 - was kid friendly.

.Tables: Output table contains data de-normalized from the following tables

 - dvdrental.payment

 - dvdrental.category

 - dvdrental.film

                        Table "dvdrental.transactions"

    Column          |         Type          | Description

    ----------------+-----------------------+-----------------------------------------

    payment\_id      | integer               | Primary Key

    date            | date                  | Date of the transaction

    amount          | numeric (5,2)         | Total amount of the transaction

    name            | character varying (25)| Name of the genre of the DVD

    is\_kidfriendly  | boolean               | Whether or not the rating is 'G', 'PG', or 'PG-13'

.Notes

  Version:        1.0

  Author:         Dylan Young

  Creation Date:  9/26/2022

  Purpose/Change: Initial script development. Minimal viable product acheived.

  Tested:         Sucessful in both HCLS-PostgreSQL01 & WGU's VDI instance name DESKTOP-BT4131D

.Notes

  Version:        1.1

  Author:         Dylan Young

  Creation Date:  6/17/2024

  Purpose/Change: Minimal changes due to retaking the classe.

  Tested:         Sucessful in both FBI-Van & WGU's VDI instance name DESKTOP-BT4131D

.Notes

  Version:        1.2

  Author:         Dylan Young

  Creation Date:  6/21/2024

  Purpose/Change: Switched to using the 'is\_kidfriendly' function

  Tested:         Sucessful in both FBI-Van & WGU's VDI instance name DESKTOP-BT4131D

 \*/

CREATE OR REPLACE FUNCTION Update\_Transactions()

    RETURNS trigger

    AS $$

    --LANGUAGE plpgsql VOLATILE

    BEGIN

    DROP TABLE IF EXISTS transactions;

    CREATE TABLE transactions (

        payment\_id integer,

        date DATE,

        amount numeric(5,2),

        name varchar(25),

        is\_kidfriendly boolean

    );

    INSERT INTO transactions

    SELECT payment.payment\_id,

        DATE(payment.payment\_date),

        payment.amount,

        category.name,

        --This is the transformation feild

        is\_kidfriendly(film.rating) as is\_kidfriendly

    FROM payment

    /\*

    \* The 'category' table is 5 steps away from the 'payment' table

    \* payment >> rental >> inventory >> film >> film\_category >> category

    \*/

    LEFT Join rental ON rental.rental\_id = payment.rental\_id

    LEFT Join inventory ON inventory.inventory\_id = rental.inventory\_id

    LEFT Join film ON film.film\_id = inventory.film\_id

    LEFT Join film\_category ON film\_category.film\_id = film.film\_id

    LEFT Join category ON category.category\_id = film\_category.category\_id

;

END;

$$

LANGUAGE plpgsql VOLATILE -- Says the function is implemented in the plpgsql language; VOLATILE says the function has side effects.

COST 100; -- Estimated execution cost of the function.

CREATE Trigger transactions\_update

    AFTER INSERT

    ON payment

    FOR EACH ROW

    EXECUTE PROCEDURE Update\_Transactions();

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

Procedure to refresh the detailed table and subsequently trigger the refresh of the summary table.

/\* Procedure\_Refresh.sql

 Author: Dylan Young

 Email: dylanyoung3244@gmail.com

 Purpose: This file creates a stored procedure call 'refresh\_genre'

 - that deletes both dvdrental.transactions and dvdrental.genre\_agg

 - and repopulates the data in them.

 Schedule: Due to how the triggers function this procedure can be ran

 - monthly or whenever the data's accuracy or age is in question

 .Notes

  Version:        1.0

  Author:         Dylan Young

  Creation Date:  9/26/2022

  Purpose/Change: Initial script development. Minimal viable product acheived.

  Tested:         Sucessful in both HCLS-PostgreSQL01 & WGU's VDI instance name DESKTOP-BT4131D

 .Notes

  Version:        1.1

  Author:         Dylan Young

  Creation Date:  6/18/2024

  Purpose/Change: Updated script for new version of the course

  Tested:         Sucessful in both FBI-Van & WGU's VDI instance name DESKTOP-BT4131D

.Notes

  Version:        1.2

  Author:         Dylan Young

  Creation Date:  6/21/2024

  Purpose/Change: Switched to using the 'is\_kidfriendly' function

  Tested:         Sucessful in both FBI-Van & WGU's VDI instance name DESKTOP-BT4131D

.Notes

  Version:        1.3

  Author:         Dylan Young

  Creation Date:  6/25/2024

  Purpose/Change: Removed possible redundant code for populating the summary table

  Tested:         Sucessful in both FBI-Van & WGU's VDI instance name DESKTOP-BT4131D

\*/

CREATE PROCEDURE refresh\_genre()

    LANGUAGE plpgsql

    AS $$

    BEGIN

    -- Start the detailed table recreation

    TRUNCATE transactions;

    INSERT INTO transactions

    SELECT payment.payment\_id,

        DATE(payment.payment\_date),

        payment.amount,

        category.name,

        --This is the transformation feild

        is\_kidfriendly(film.rating) as is\_kidfriendly

    FROM payment

    /\*

    \* The 'category' table is 5 steps away from the 'payment' table

    \* payment >> rental >> inventory >> film >> film\_category >> category

    \*/

    LEFT Join rental ON rental.rental\_id = payment.rental\_id

    LEFT Join inventory ON inventory.inventory\_id = rental.inventory\_id

    LEFT Join film ON film.film\_id = inventory.film\_id

    LEFT Join film\_category ON film\_category.film\_id = film.film\_id

    LEFT Join category ON category.category\_id = film\_category.category\_id;

END $$;

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

Inside of pgAdmin, there is a tool called pgAgent. This runs as either a daemon or a service and can be installed using the Application Stack Builder. From there, you can create a new job to run the stored procedure used to refresh the data in the detailed and summary tables in order as well as set a schedule of your choosing.