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.

This report is about finding the least popular film/movie available to rent. The business question being asked is “What is the least popular movie/film?”. This will benefit the DVD business by being able to see what movies to replace with better movies their customers will like. The report will help to reduce costs and increase revenue.

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

The table below shows what fields are included in either the Summary or Detailed table as well as what database table they came from.

Field name	**Detailed or Summary Table**	**Database Table**
film_id	Detailed and Summary	Film
title	Detailed and Summary	Film
genre (name)	Detailed	Category
popularity (rental_date)	Detailed and Summary	Rental
description	Detailed	Film
release_year	Detailed	Film
avg_amount (amount)	Summary	Payment

In the table above, the field’s popularity, genre, and avg amount are all changed by their names, what they show, or both. Inside the parenthesis are the fields they get their information from.

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

The table below shows the name of the field, the datatype of that field, as well as a brief description of the field.

Field Name	**Datatype**	**description**
film_id	integer	Shows the film_id of a film.
title	varchar	Shows the title of the film.
genre	varchar	Shows the genre/or film category of the film.
popularity	integer	Shows how often a film has been rented. When inserted, it will be ordered by least rented to most rented.
description	text	Shows a brief description of the film.
release_year	integer	Shows the year the film released.

avg_amount	numeric	Shows the average amount spent on a film.

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.

For the detailed table, the Film, Rental, and Category tables will provide the information necessary.

For the summary table, the Rental, Payment, and Film table will provide the information necessary.

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*).

The one field that will require a custom transformation in the detailed table is the popularity field. This field is meant to use the rental date field from the rental table as its information. Since showing just the date of the rent is not needed to find the popularity of a film, a custom function called rent count is defined. This function helps to count the number of times a film was rented based on the date, taking in the film id as its parameter. When inserting the data, the data gets put in order from least popular to most popular.

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

A stakeholder would use the information based on the detailed table to know what film to no longer provide to customers to make space for new films.

A stakeholder would use the information based on the summary table to increase or decrease the cost of renting a film based on popularity.

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

According to fictionhorizon.com's "How Long Do Movies Stay in Theaters? (With Statistics)", movies can stay in the movie theater for anywhere from 2-4 weeks or even 8-10 weeks. The more popular, the longer a film will stay in the theater. Using this information, the detailed and summary tables should get refreshed every 2 weeks to track the least popular film.

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

```
CREATE OR REPLACE FUNCTION rent_count(movie_id integer)

RETURNS integer AS

$$

DECLARE rental_count integer;
```

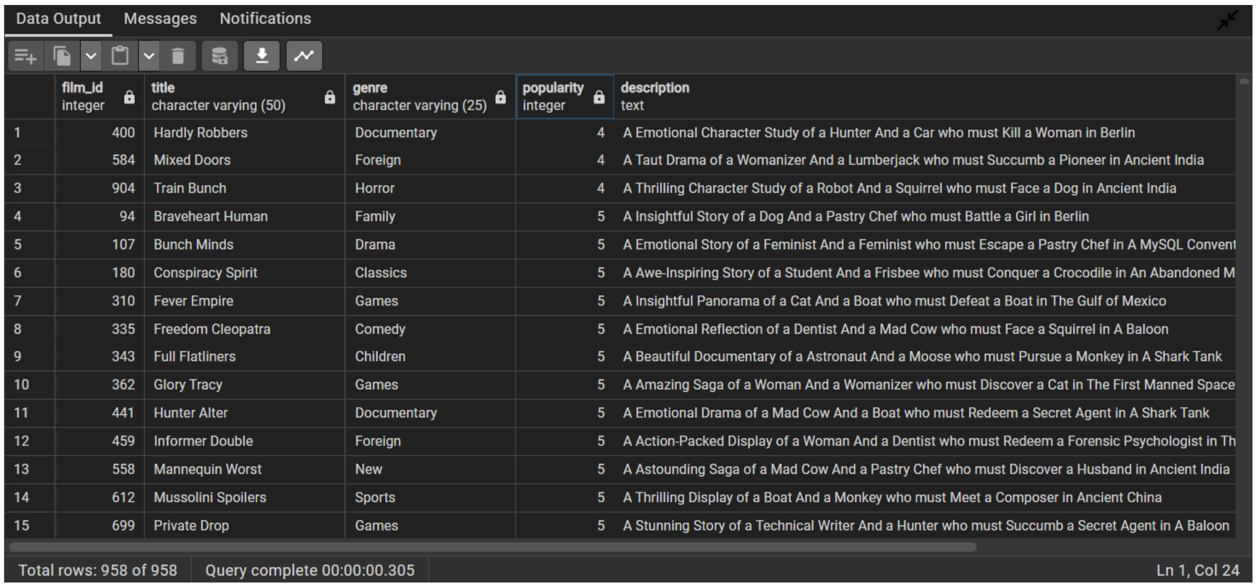
```
BEGIN
  SELECT COUNT(rental.rental_date) INTO rental_count
       FROM rental
       JOIN inventory ON rental.inventory_id = inventory.inventory_id
       JOIN film ON inventory.film_id = film.film_id
       WHERE film.film_id = movie_id;
RETURN rental_count;
END;
$$
LANGUAGE plpgsql;
```

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

```
-- Detailed table
CREATE TABLE detailed_table(
    film_id INTEGER,
    title VARCHAR(50),
    genre VARCHAR(25),
    popularity INTEGER,
    description TEXT,
    release_year INTEGER
);


-- Summary table
CREATE TABLE summary_table(
    film_id INTEGER,
    title VARCHAR(50),
    popularity INTEGER,
    avg_amount NUMERIC(5,2)
);

--verify table

SELECT * FROM detailed_table;

SELECT * FROM summary_table;


SELECT COUNT(*) FROM detailed_table;

SELECT COUNT(*) FROM summary_table;
```

<u>The photos below show that the variables are in the correct order, the datatypes are correct, the values are outputted the way they are supposed to, and the numbers of rows match. Note: The table omits popularity values of 0 because the database does not provide the information of whether a film is available to rent. If the film is not available to rent and it's included, it would not help the stakeholders to take the proper action.</u>

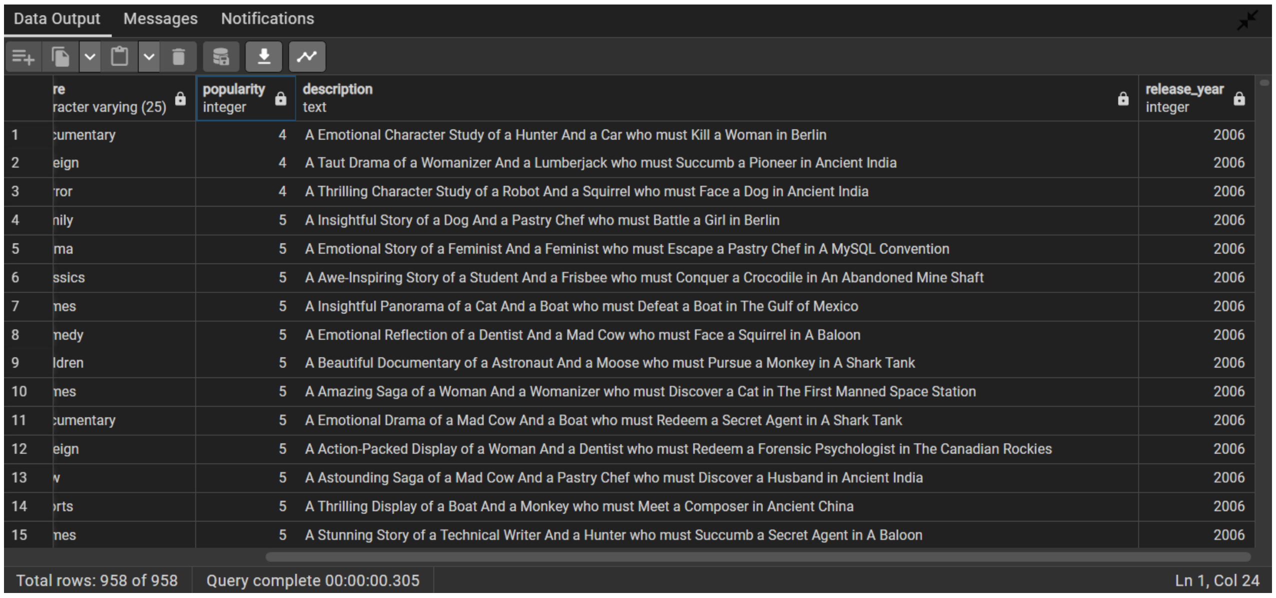
<u>This is the output after the "SELECT * FROM detailed table;".</u>



	film_id integer	title character varying (50)	genre character varying (25)	popularity integer	description text
1	400	Hardly Robbers	Documentary	4	A Emotional Character Study of a Hunter And a Car who must Kill a Woman in Berlin
2	584	Mixed Doors	Foreign	4	A Taut Drama of a Womanizer And a Lumberjack who must Succumb a Pioneer in Ancient India
3	904	Train Bunch	Horror	4	A Thrilling Character Study of a Robot And a Squirrel who must Face a Dog in Ancient India
4	94	Braveheart Human	Family	5	A Insightful Story of a Dog And a Pastry Chef who must Battle a Girl in Berlin
5	107	Bunch Minds	Drama	5	A Emotional Story of a Feminist And a Feminist who must Escape a Pastry Chef in A MySQL Convent
6	180	Conspiracy Spirit	Classics	5	A Awe-Inspiring Story of a Student And a Frisbee who must Conquer a Crocodile in An Abandoned M
7	310	Fever Empire	Games	5	A Insightful Panorama of a Cat And a Boat who must Defeat a Boat in The Gulf of Mexico
8	335	Freedom Cleopatra	Comedy	5	A Emotional Reflection of a Dentist And a Mad Cow who must Face a Squirrel in A Baloon
9	343	Full Flatliners	Children	5	A Beautiful Documentary of a Astronaut And a Moose who must Pursue a Monkey in A Shark Tank
10	362	Glory Tracy	Games	5	A Amazing Saga of a Woman And a Womanizer who must Discover a Cat in The First Manned Space
11	441	Hunter Alter	Documentary	5	A Emotional Drama of a Mad Cow And a Boat who must Redeem a Secret Agent in A Shark Tank
12	459	Informer Double	Foreign	5	A Action-Packed Display of a Woman And a Dentist who must Redeem a Forensic Psychologist in Th
13	558	Mannequin Worst	New	5	A Astounding Saga of a Mad Cow And a Pastry Chef who must Discover a Husband in Ancient India
14	612	Mussolini Spoilers	Sports	5	A Thrilling Display of a Boat And a Monkey who must Meet a Composer in Ancient China
15	699	Private Drop	Games	5	A Stunning Story of a Technical Writer And a Hunter who must Succumb a Secret Agent in A Baloon

```
Total rows: 958 of 958    Query complete 00:00:00.305                    Ln 1, Col 24
```

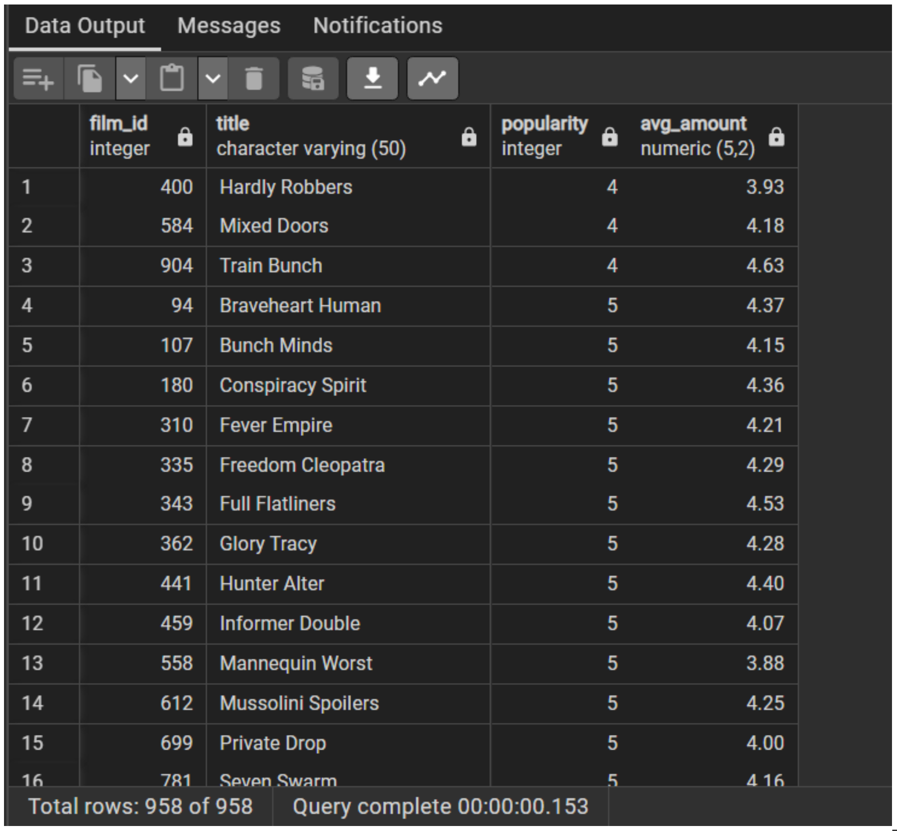
<u>This is the detailed Table continued.</u>


Data Output Messages Notifications

	re racter varying (25)	popularity integer	description text	release_year integer
1	umentary	4	A Emotional Character Study of a Hunter And a Car who must Kill a Woman in Berlin	2006
2	eign	4	A Taut Drama of a Womanizer And a Lumberjack who must Succumb a Pioneer in Ancient India	2006
3	ror	4	A Thrilling Character Study of a Robot And a Squirrel who must Face a Dog in Ancient India	2006
4	nily	5	A Insightful Story of a Dog And a Pastry Chef who must Battle a Girl in Berlin	2006
5	ma	5	A Emotional Story of a Feminist And a Feminist who must Escape a Pastry Chef in A MySQL Convention	2006
6	ssics	5	A Awe-Inspiring Story of a Student And a Frisbee who must Conquer a Crocodile in An Abandoned Mine Shaft	2006
7	nes	5	A Insightful Panorama of a Cat And a Boat who must Defeat a Boat in The Gulf of Mexico	2006
8	nedy	5	A Emotional Reflection of a Dentist And a Mad Cow who must Face a Squirrel in A Baloon	2006
9	ldren	5	A Beautiful Documentary of a Astronaut And a Moose who must Pursue a Monkey in A Shark Tank	2006
10	nes	5	A Amazing Saga of a Woman And a Womanizer who must Discover a Cat in The First Manned Space Station	2006
11	umentary	5	A Emotional Drama of a Mad Cow And a Boat who must Redeem a Secret Agent in A Shark Tank	2006
12	eign	5	A Action-Packed Display of a Woman And a Dentist who must Redeem a Forensic Psychologist in The Canadian Rockies	2006
13	v	5	A Astounding Saga of a Mad Cow And a Pastry Chef who must Discover a Husband in Ancient India	2006
14	rts	5	A Thrilling Display of a Boat And a Monkey who must Meet a Composer in Ancient China	2006
15	nes	5	A Stunning Story of a Technical Writer And a Hunter who must Succumb a Secret Agent in A Baloon	2006

Total rows: 958 of 958 Query complete 00:00:00.305 Ln 1, Col 24

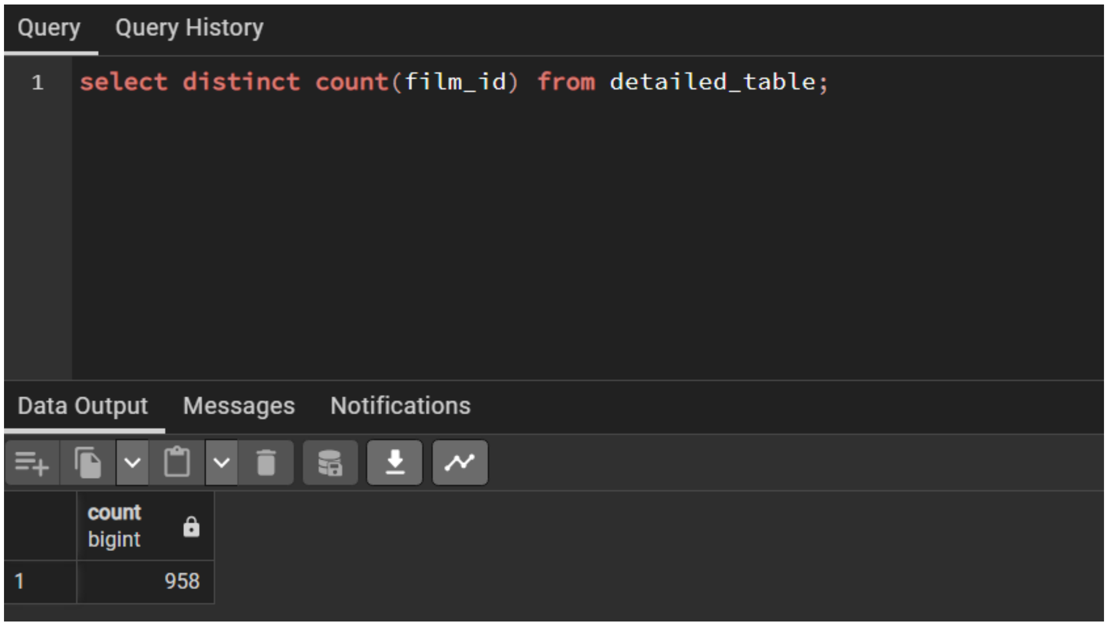
<u>This is the output after the "SELECT * FROM summary_table;".</u>


Data Output Messages Notifications

	film_id integer	title character varying (50)	popularity integer	avg_amount numeric (5,2)
1	400	Hardly Robbers	4	3.93
2	584	Mixed Doors	4	4.18
3	904	Train Bunch	4	4.63
4	94	Braveheart Human	5	4.37
5	107	Bunch Minds	5	4.15
6	180	Conspiracy Spirit	5	4.36
7	310	Fever Empire	5	4.21
8	335	Freedom Cleopatra	5	4.29
9	343	Full Flatliners	5	4.53
10	362	Glory Tracy	5	4.28
11	441	Hunter Alter	5	4.40
12	459	Informer Double	5	4.07
13	558	Mannequin Worst	5	3.88
14	612	Mussolini Spoilers	5	4.25
15	699	Private Drop	5	4.00
16	781	Seven Swarm	5	4.16

Total rows: 958 of 958 Query complete 00:00:00.153

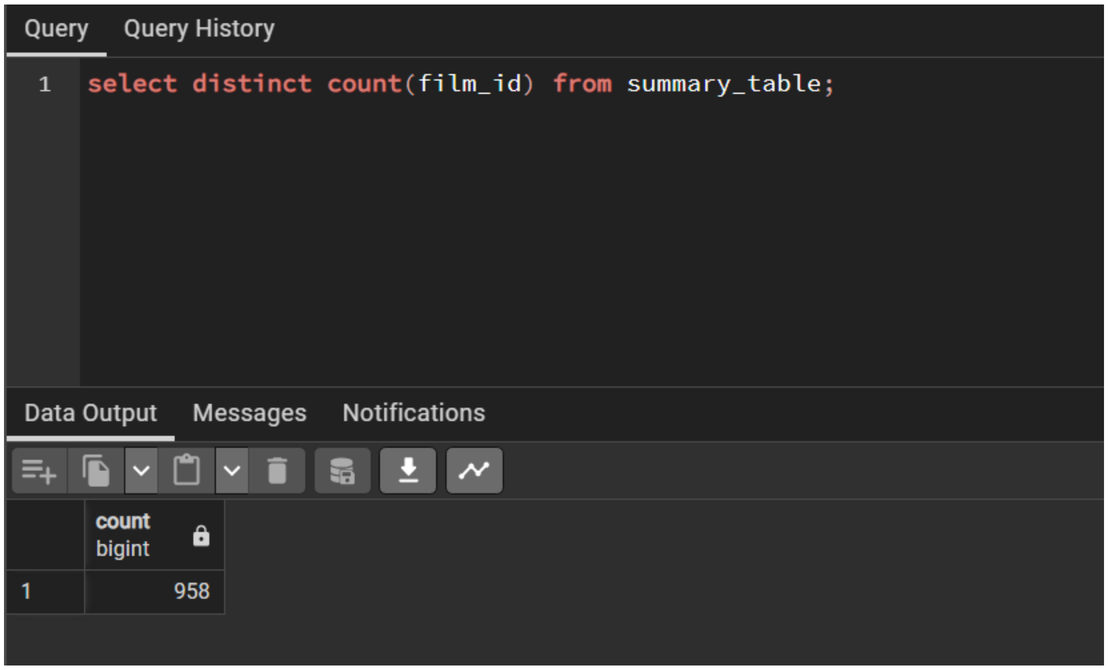
This is the output after the “SELECT COUNT(*) FROM detailed_table;”. This shows the numbers of rows.



```
select distinct count(film_id) from detailed_table;
```

```
  count
  bigint
1    958
```

This is the output after the “SELECT COUNT(*) FROM summary_table;”. This also shows the numbers of rows.



```
select distinct count(film_id) from summary_table;
```

```
  count
  bigint
1    958
```

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.

Data has been verified in part C.

```
INSERT INTO detailed_table (film_id, title, genre, popularity, description, release_year)

SELECT DISTINCT film.film_id, title, category.name AS genre,

        rent_count(film.film_id) AS popularity, description, release_year

FROM film

JOIN film_category ON film.film_id = film_category.film_id

JOIN inventory ON film.film_id = inventory.film_id

JOIN category ON film_category.category_id = category.category_id

ORDER BY popularity ASC;
```

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.

```
-- Trigger Function

CREATE OR REPLACE FUNCTION my_trigger_function()

RETURNS TRIGGER

LANGUAGE plpgsql

AS $$

BEGIN

  IF NOT EXISTS (

    SELECT *

    FROM summary_table

    WHERE film_id = NEW.film_id

  ) THEN

    INSERT INTO summary_table(film_id, title, popularity, avg_amount)

    SELECT DISTINCT film.film_id, title, rent_count(film.film_id) AS popularity,

      AVG(payment.amount) AS avg_amount

    FROM film

    JOIN inventory ON film.film_id = inventory.film_id

    JOIN rental ON inventory.inventory_id = rental.inventory_id

    JOIN payment ON rental.customer_id = payment.customer_id
    GROUP BY film.film_id, title
    ORDER BY popularity ASC;
  END IF;
  RETURN NEW;
END;
$$;


-- Actual Trigger
CREATE TRIGGER my_trigger
AFTER INSERT
ON detailed_table
FOR EACH ROW
EXECUTE PROCEDURE my_trigger_function();
```

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.

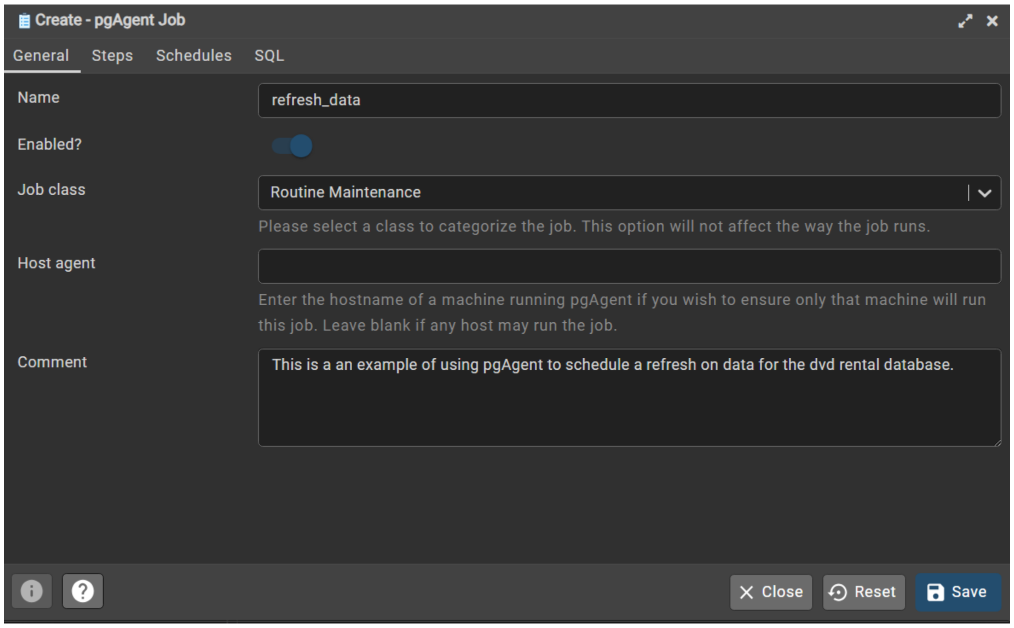
```
-- CREATE PROCEDURE
CREATE PROCEDURE refresh_data()
LANGUAGE plpgsql
AS $$
BEGIN
      DELETE FROM detailed_table;
      DELETE FROM summary_table;


      INSERT INTO detailed_table(film_id, title, genre, popularity, description, release_year)
      SELECT DISTINCT film.film_id, title, category.name AS genre,
                                                   rent_count(film.film_id) AS popularity, description,
release_year
```

```
    FROM film

    JOIN film_category ON film.film_id = film_category.film_id

    JOIN inventory ON film.film_id = inventory.film_id

    JOIN category ON film_category.category_id = category.category_id

    ORDER BY popularity ASC;


END;

$$;
```

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

A relevant job scheduling tool to automate the stored procedure would be the PgAgent job scheduler. According to fictionhorizon.com's "How Long Do Movies Stay in Theaters? (With Statistics)", the average movie stays in the theater for 2-4 weeks with some movies even staying as long as 8-10 weeks. With that in mind, refreshing the detailed and summary tables every 2 weeks at most would help to update the data to keep track of the least and most popular movies.

The photos below show the scheduler set up in pgAdmin 4.



Name: refresh_data

Enabled?

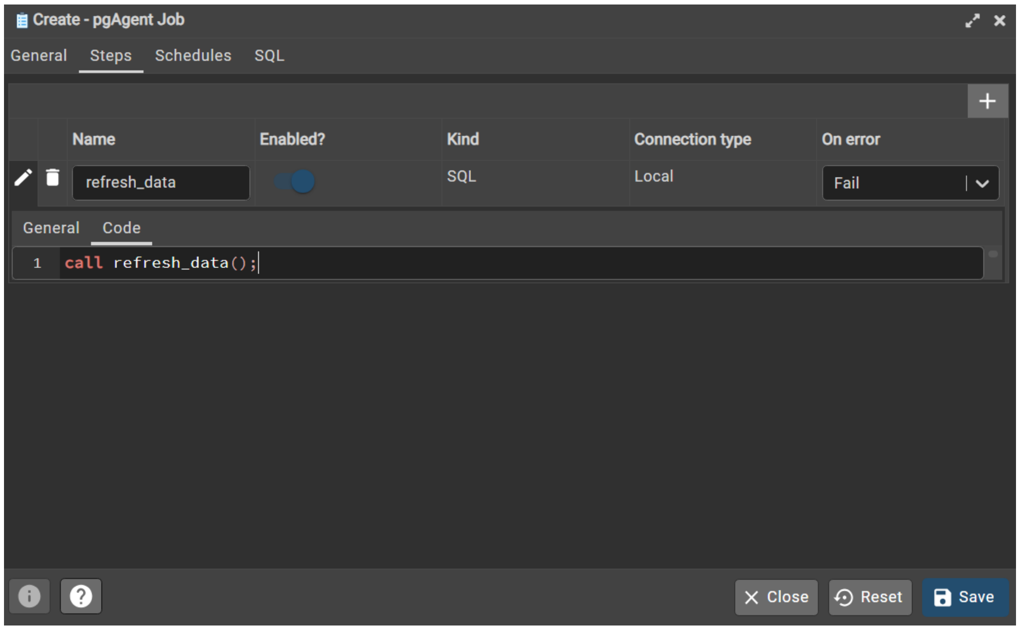
Job class: Routine Maintenance

Please select a class to categorize the job. This option will not affect the way the job runs.

Host agent:

Enter the hostname of a machine running pgAgent if you wish to ensure only that machine will run this job. Leave blank if any host may run the job.

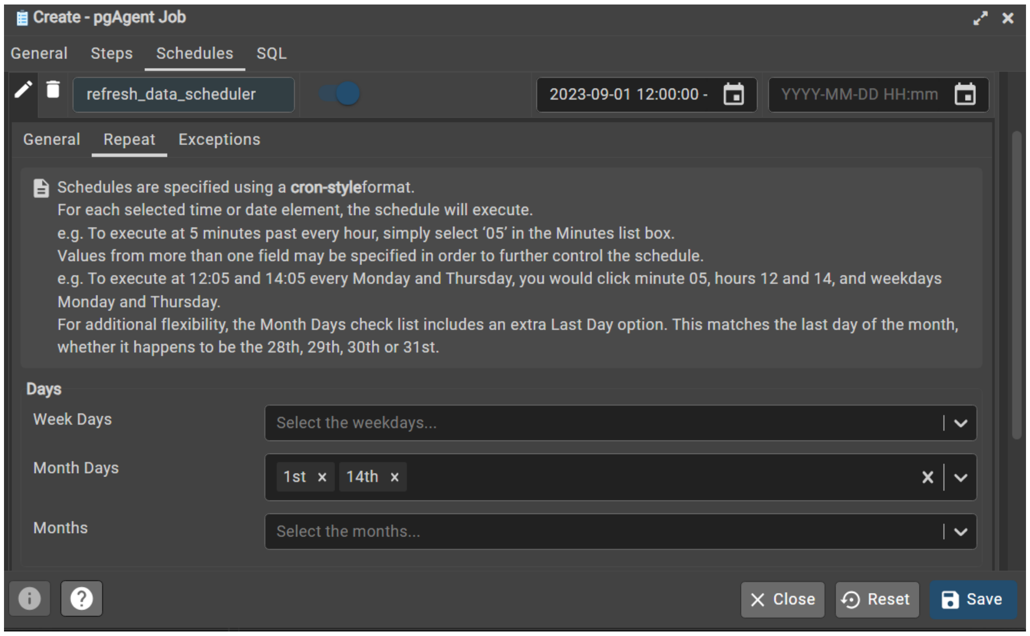
Comment: This is a an example of using pgAgent to schedule a refresh on data for the dvd rental database.



Name	Enabled?	Kind	Connection type	On error
refresh_data	●	SQL	Local	Fail

General | Code

```
call refresh_data();
```

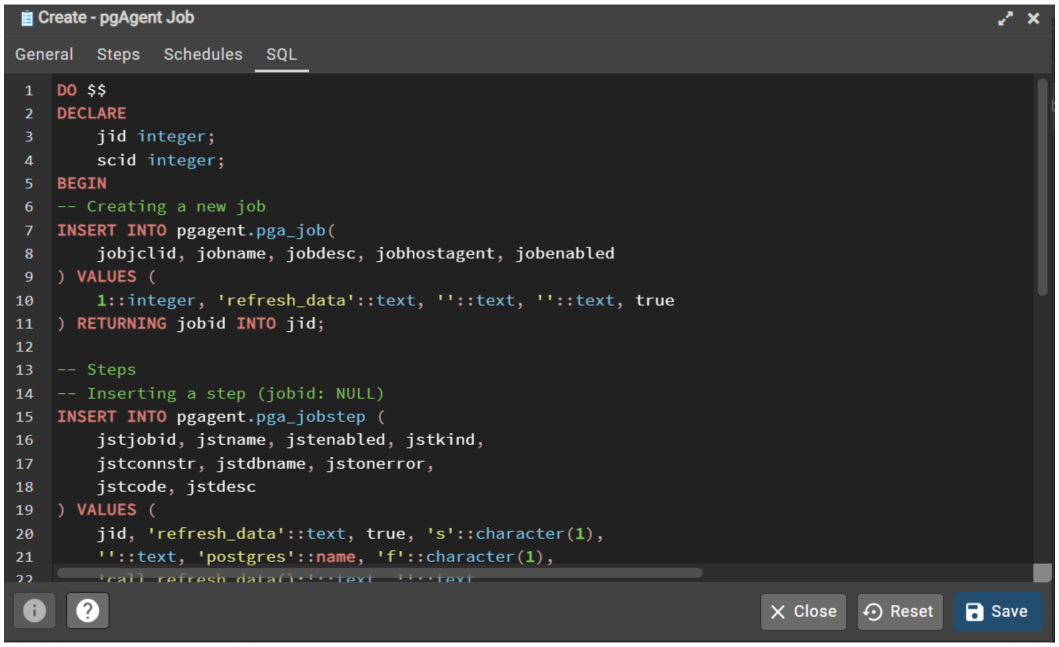


refresh_data_scheduler ● 2023-09-01 12:00:00 - YYYY-MM-DD HH:mm

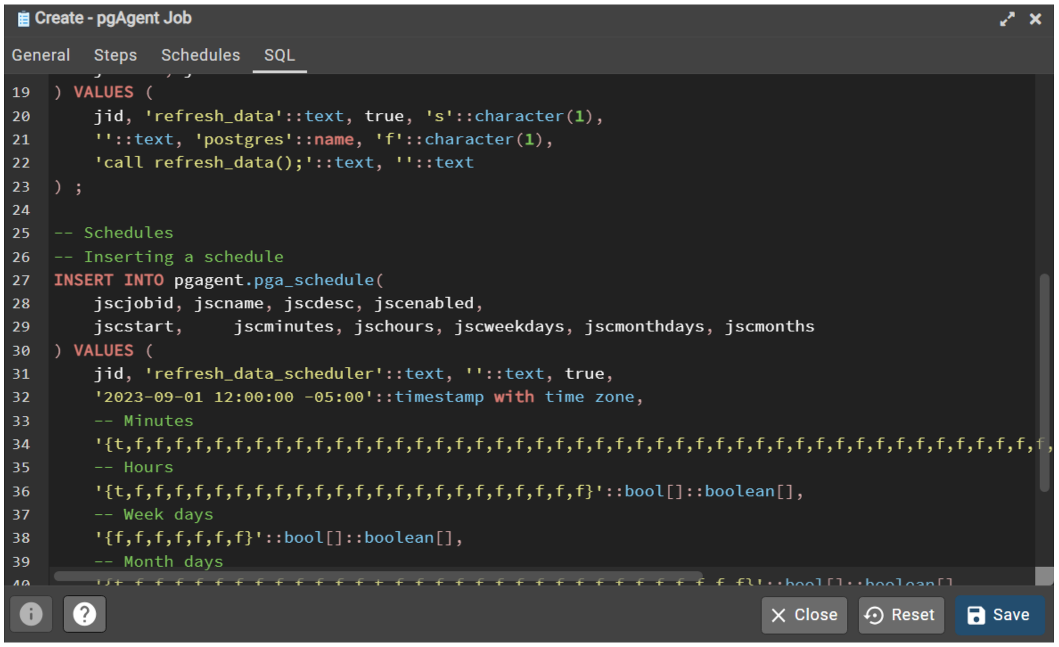
Schedules are specified using a **cron-style**format.
For each selected time or date element, the schedule will execute.
e.g. To execute at 5 minutes past every hour, simply select '05' in the Minutes list box.
Values from more than one field may be specified in order to further control the schedule.
e.g. To execute at 12:05 and 14:05 every Monday and Thursday, you would click minute 05, hours 12 and 14, and weekdays Monday and Thursday.
For additional flexibility, the Month Days check list includes an extra Last Day option. This matches the last day of the month, whether it happens to be the 28th, 29th, 30th or 31st.

Days

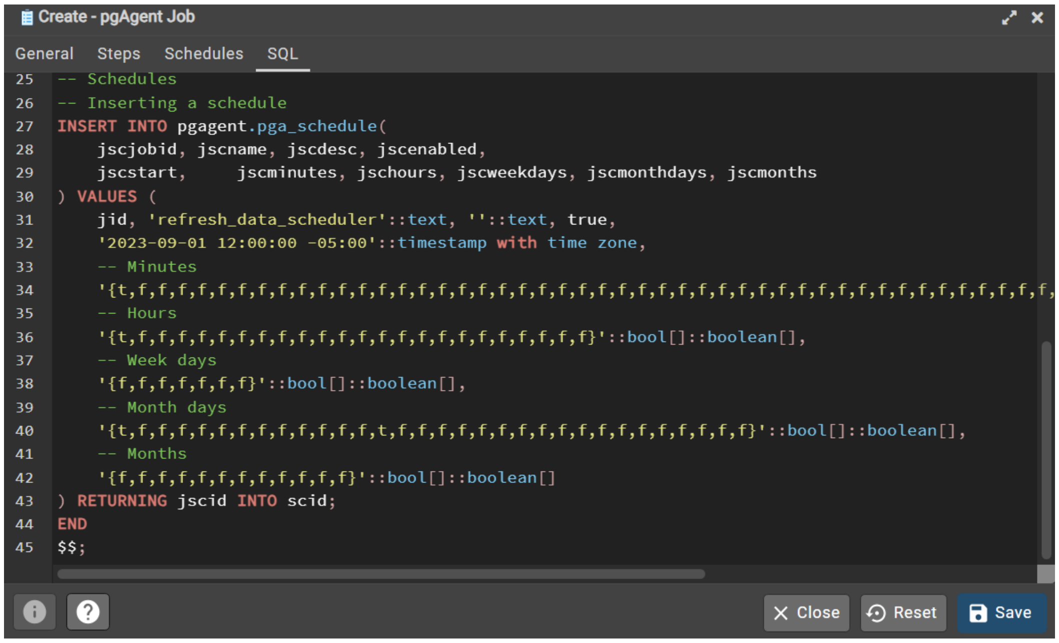
Week Days	Select the weekdays...
Month Days	1st × 14th ×
Months	Select the months...



```
DO $$
DECLARE
    jid integer;
    scid integer;
BEGIN
-- Creating a new job
INSERT INTO pgagent.pga_job(
    jobjclid, jobname, jobdesc, jobhostagent, jobenabled
) VALUES (
    1::integer, 'refresh_data'::text, ''::text, ''::text, true
) RETURNING jobid INTO jid;

-- Steps
-- Inserting a step (jobid: NULL)
INSERT INTO pgagent.pga_jobstep (
    jstjobid, jstname, jstenabled, jstkind,
    jstconnstr, jstdbname, jstonerror,
    jstcode, jstdesc
) VALUES (
    jid, 'refresh_data'::text, true, 's'::character(1),
    ''::text, 'postgres'::name, 'f'::character(1),
    'call refresh_data();'::text, ''::text
```



```
) VALUES (
    jid, 'refresh_data'::text, true, 's'::character(1),
    ''::text, 'postgres'::name, 'f'::character(1),
    'call refresh_data();'::text, ''::text
) ;

-- Schedules
-- Inserting a schedule
INSERT INTO pgagent.pga_schedule(
    jscjobid, jscname, jscdesc, jscenabled,
    jscstart,     jscminutes, jschours, jscweekdays, jscmonthdays, jscmonths
) VALUES (
    jid, 'refresh_data_scheduler'::text, ''::text, true,
    '2023-09-01 12:00:00 -05:00'::timestamp with time zone,
    -- Minutes
    '{t,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,
    -- Hours
    '{t,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f}'::bool[]::boolean[],
    -- Week days
    '{f,f,f,f,f,f,f}'::bool[]::boolean[],
    -- Month days
    '{t,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f}'::bool[]::boolean[]
```



```
-- Schedules
-- Inserting a schedule
INSERT INTO pgagent.pga_schedule(
    jscjobid, jscname, jscdesc, jscenabled,
    jscstart,     jscminutes, jschours, jscweekdays, jscmonthdays, jscmonths
) VALUES (
    jid, 'refresh_data_scheduler'::text, ''::text, true,
    '2023-09-01 12:00:00 -05:00'::timestamp with time zone,
    -- Minutes
    '{t,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,
    -- Hours
    '{t,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f}'::bool[]::boolean[],
    -- Week days
    '{f,f,f,f,f,f,f}'::bool[]::boolean[],
    -- Month days
    '{t,f,f,f,f,f,f,f,f,f,f,f,f,t,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f}'::bool[]::boolean[],
    -- Months
    '{f,f,f,f,f,f,f,f,f,f,f,f}'::bool[]::boolean[]
) RETURNING jscid INTO scid;
END
$$;
```

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

For the video, I will be going in this order: Create detailed and summary tables, create transformation function, create the trigger function, create the trigger, insert into the detailed table/extract data from the database, and then create my stored procedure. This is because after creating the tables, I will need the custom function included in my trigger and insert statement. My insert statement will activate my trigger, so it must be used after that.

Panopto presentation: https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=d2cadb0b-b86c-41ee-acb2-b06a00c94a56

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.

Luka Glavas. (2022). *How Long Do Movies Stay in Theaters? (With Statistics).* https://fictionhorizon.com/how-long-do-movies-stay-in-theaters/

PostgreSQLTutorial.com. (2022). *PostgreSQL Sample Database*. https://www.postgresqltutorial.com/postgresql-getting-started/postgresql-sample-database/

W3schools.com.(2023). *SQL Tutorial*. https://www.w3schools.com/sql/default.asp

Hugo Dias. (2020). *An Overview of Job Scheduling Tools for PostgreSQL*. https://severalnines.com/blog/overview-job-scheduling-tools-postgresql/