The data used for this report will help answer the question about the number of movies starring the same actor in the database. That will help stakeholders understand who the most popular actors are, as of right now in the database.

A1. The data used for this report will be extracted from the dvdrental database and will include the actors first and last name in one column named **actor_name** in a variable character data type, the actor id number under **actor_id** column in integer data type, the film id number under **film_id** column in an integer data type and the film name under **title** column in a variable character data type.

A2. To answer this question, I will be using data in the **film** table, the **film_actor** table and the **actor** table.

A3. the columns to be used in the detailed table are **film_id**, **title**, **actor_id**, **first_name**, **last_name**. The columns to be used in the summary table are **film_count** and **actor_name** in ascending order according to count of films.

A4. The **actor_name** column in the detailed table will require transformation of data from the concatenating the **first_name** and **last_name** columns from the **actor** table into one column to make it easier to identify the actors and increase readability.

A5. The detailed table serves the purpose of listing the title of every film along with the names of the actors that will help stakeholders identify the details pertaining to the number of movies starring every actor and recognize the popularity of those actors according to the number of movies.
The summary table will provide a quick look at the data where they can see the full actor's name next to the number of movies starring that same actor in the database. This will provide stakeholders with quick information about the number of films starring each actor and help them to understand their popularity.

The stakeholders can then conduct surveys to validate the popularity of those actors and if customers are still interested in movies starring those actors. With this information profits can be increased as customer satisfaction rates will be higher. Customers will be able to find movies in our database starring the most popular actors that they are interested in watching. The data can also help gain more customers, by adding more films with more actors that have not been included in this database yet.

A6. The report will be refreshed yearly to review the number of the new movies added to the database and to detect any changes in the number of movies starring every actor to assess popularity of each actor.

B. code for creating detailed table:

```
DROP TABLE IF EXISTS detailed;
create table detailed (
        film_id int,
        title varchar(100),
        actor_id int,
        actor_name varchar(100)
);
```

----

code for creating summary table :

```
DROP TABLE IF EXISTS summary;
create table summary (
        film_count int,
        actor_name varchar(100)
);
```

C. the code for inserting and transforming data into detailed table:

```
insert into detailed (film_id, title, actor_id, actor_name)
select f.film_id, f.title,fa.actor_id,
        (select concat (a.first_name,' ', a.last_name) as actor_name)
from film f
join film_actor fa
on f.film_id = fa.film_id
join actor a
on fa.actor_id=a.actor_id;
```

D. transformation of data from A4

Is performed through concatenating first_name and last_name coulmns from the actor table into one column into detailed table

The code is written in a subquery in the select statement

```
(Select concat (a.first_name,'  ', a.last_name) as actor_name)
```

Full code:

```
insert into detailed (film_id, title, actor_id, actor_name)
select f.film_id, f.title,fa.actor_id,
        (select concat (a.first_name,' ', a.last_name) as actor_name)
from film f
join film_actor fa
on f.film_id = fa.film_id
join actor a
on fa.actor_id=a.actor_id;
```

E. create a trigger on the detailed table

```
create or replace function function_detailed()
returns trigger
language plpgsql
as $$
begin
delete from summary;
insert into summary (film_count, actor_name)
select count(film_id) as film_count, actor_name
from detailed d
group by actor_name
order by film_count desc;
return new;
end;
$$;

Create or replace trigger trigger_detailed
        after insert
        on detailed
        for each statement
                execute procedure function_detailed();
```

F. Create a stored procedure that can be used to refresh the data in *both* your detailed and summary tables.

```
create or replace procedure refresh_reports()
language plpgsql
as $$
begin
delete from detailed;
delete from summary;

insert into detailed (film_id, title, actor_id, actor_name)
select f.film_id, f.title,fa.actor_id,
        (select concat (a.first_name,' ', a.last_name) as actor_name)
from film f
join film_actor fa
on f.film_id = fa.film_id
join actor a
on fa.actor_id=a.actor_id;

insert into summary (film_count, actor_name)
select count(film_id) as film_count, actor_name
from detailed d
```

```
group by actor_name
order by film_count desc;

return;
end;
$$;
```

The stored procedure should be executed annually on the last day of December to ensure data accuracy and freshness. It can be executed using the function:
call refresh_reports()

F1. The stored procedure can be run on schedule by using an external tool. I first downloaded the application stack builder included in the installation of pgAdmin4. In the stack builder there are lists for extensions including the pgAgent which is a tool that can be used to schedule stored procedures. From within the pgAgent tool I set the procedure to run at the end of each year. This will ensure data accuracy and freshness since new films are released annually and this will help assess the popularity of the actors according to the new number of films where they appeared in.