Business Report Summary: Section A

Summarize one real-world business report that can be created from the attached Data Sets and Associated Dictionaries:

To optimize sales, it is important that we closely monitor the popularity of the films that we keep in inventory. Film popularity will likely shift and change as new films are released and consumer preferences change. We need to make sure that we are replacing films that are not being rented with new releases periodically to guarantee that our customers will have a fresh inventory of films to choose from, and to trim our inventory carrying cost to only those films that are generating sales. This report should be used to help make decisions on which films should be replaced at regular intervals.

Describe the data used for the report:

To generate this report, we will need to rely on the data housed within our dvdrental database. Specifically, the data required will come from the inventory, film, staff, and rental tables within the database. We will use these three tables to analyze the number of times that each film is rented over a defined period. This will enable our managers at each store to determine which films they need to look at replacing.

Identify two or more specific tables from the given dataset that will provide the data necessary for the detailed and the summary sections of the report:

The inventory, film, staff, and rental tables will be used to put the inventory reports together.

Identify the specific fields that will be included in the detailed and the summary sections of the report:

For the detailed portion of the inventory report, we will use the inventory_id and film_id fields from the inventory table. We will also use the first_name, last_name, and email fields from the staff table. From the film table we will use the title and film_id fields. Finally, we will use the rental_date and staff_id fields from the rental table.

For the summary section of the report, we will use the title, email, first_name & last_name columns. These two columns will be transformed/concatenated into a manager_name field as detailed in the next section. We will also be creating a new field titled rental_count that will be useful in the summary table.

Identify one field in the detailed section that will require a custom transformation and explain why it should be transformed:

We will need to transform the first_name and last_name fields within the detailed table into a new field title manager_name so that upper management can easily see the full names of the store managers at each location if they need to follow up with them directly.

Explain the different business uses of the detailed and the summary sections of the report:

The detailed section of the report will give management a report to further analyze when films are being rented which will allow them to drill into the date/time that specific films are being rented which could lead to driving more informed decisions about inventory management. However, that analysis is outside of the scope for this project.

The summary section of the report will give management an easy-to-read report on which films are being rented, and those that are not. This will create efficiencies when deciding which films need to be removed from inventory and replaced with other films. Management could also run marketing campaigns highlighting the more popular films and use this report to decide which films to include. The summary should be automatically emailed to our store managers at regular intervals as defined by the business stakeholders. The email automation is not within the scope of this project and will need to be revisited at a future date.

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

I recommend that this report be refreshed once per month at minimum. New films are constantly being released and this will help us to stay relevant in the marketplace and ensure that our customers are returning to our stores for film rentals.

Section B: Below statement creates empty detailed table.

```
create table detailed (
	rental_date date,
	inventory_id smallint,
	title varchar(250),
	film_id smallint,
	staff_id smallint,
	first_name varchar(45),
	last_name varchar(45),
	email varchar(50)
);
```

Section B: Below statement creates empty summary table.

```
create table summary (
	rental_count smallint,
	title varchar(250),
	manager_name varchar(100),
```

```
        email varchar(50)
);
```

Section C: Below statement extracts data from DB and populates detailed table.

```
insert into detailed (
        rental_date,
        inventory_id,
        title,
        film_id,
        staff_id,
        first_name,
        last_name,
        email
)
select
        r.rental_date, i.inventory_id, f.title, i.film_id,
        r.staff_id, s.first_name, s.last_name, s.email

from rental as r
inner join staff as s on s.staff_id = r.staff_id
inner join inventory as i on i.inventory_id = r.inventory_id
inner join film as f on f.film_id = i.film_id;
```

Section D: Below statement creates function to transform the staff name to one column.

```
create or replace function name_concat(first_name text, last_name text)
returns varchar(100)
language plpgsql
as
$$
declare manager_name varchar(100);
begin
        select concat_ws(' ',first_name, last_name) into manager_name;
        return manager_name;
end;
$$
```

Section E: Below statement creates trigger function on detailed table to update/populate the summary table as the detailed table is updated.

```
create or replace function refresh_summary()
returns trigger
language plpgsql
```

```
as $$
begin

insert into summary(
	select
		count(title),
		title,
		name_concat(first_name, last_name) as manager_name,
		email
	from detailed
	group by title, manager_name, email
);

return new;
end; $$
```

Section E: Below statement creates trigger on detailed table to update/populate the summary table as the detailed table is updated.

```
create trigger refresh_summary
	after insert
	on detailed
	for each statement
	execute procedure refresh_summary();
```

Below statement creates trigger function to remove duplicates from the summary table. It is triggered by any update to the summary table which means it will be triggered shortly after any update to the detailed table as well.

```
create or replace function remove_dups()
returns trigger
language plpgsql
as $$
```

```
begin

drop table if exists new_summary;
drop table if exists old_summary;

create table new_summary as
select rental_count, title, manager_name, email
from summary
group by title, rental_count, manager_name, email;

alter table summary
rename to old_summary;

alter table new_summary
rename to summary;


return new;
end; $$
```

Below statement creates trigger on summary table to remove duplicates.

```
create trigger remove_dups
        after insert
        on summary
        for each statement
        execute procedure remove_dups();
```

Section F: Below statement creates stored procedure to repopulate the detailed table, which will in turn trigger the summary table to be refreshed as well.

```
create or replace procedure refresh_tables()
language plpgsql
as $$
begin
```

```
	truncate detailed;
	truncate summary;

	insert into detailed (
		rental_date,
		inventory_id,
		title,
		film_id,
		staff_id,
		first_name,
		last_name,
		email
	)

	select
		r.rental_date, i.inventory_id, f.title, i.film_id,
		r.staff_id, s.first_name, s.last_name, s.email

	from rental as r
	inner join staff as s on s.staff_id = r.staff_id
	inner join inventory as i on i.inventory_id = r.inventory_id
	inner join film as f on f.film_id = i.film_id;


end; $$
```

Section F: Below statement calls the stored procedure.

```
Call refresh_tables();
```

Section F1: Below describes how the stored procedure can be automatically run on a schedule.

(https://severalnines.com/blog/overview-job-scheduling-tools-postgresql/, 2020) The stored procedure can be run using pgAgent scheduling tool for PosgreSQL. The agent can be set up to automatically check for procedures to run at regular intervals. If pgAgent is not installed within your pgAdmin application, you will need to do that first. After you have completed the configuration of the pgAgent extension, you will be able to access pgAgent jobs in your menu and scheduling the procedure can be accomplished there.

I recommend that this procedure be executed once per month. This will guarantee that our management can keep tabs on the films that are not being rented so that they know which to remove and replace with newer films as they come out.

Section G: Below is a link to the video demonstration

Section H: Record web sources used to acquire data or segments of third party-code to support the application if applicable.

This is not applicable to this project. No third-party data or code segments were utilized.

Section I: Acknowledge Sources:

Several9s. (2020)***.*** *An Overview of Job Scheduling Tools for PostrgeSQL.* ***https://severalnines.com/blog/overview-job-scheduling-tools-postgresql/***