010743036
Brandon R Pinkston
3/2/2023

# D191 PA: Data Analysis

Section A – Part 1:

*** () denotes the table the field is derived from. ***

*** [] denotes the data type of the field. ***

Detailed table fields:

| Staff_id(staff) [INT] | First_name(staff) [VCHAR] | Last_name(staff) [VCHAR] | Amount(Payment) [NUMERIC] | Payment_date(payment) [TIMESTAMP] |
|---|---|---|---|---|

Summary table fields:

| Staff_id(staff) [INT] | Full_name(staff) [VCHAR] | Sales_sum(payment) [NUMERIC] | Sales_count (payment) [INT] | Payment_month (payment) [INT] | Payment_year (payment) [INT] |
|---|---|---|---|---|---|

Section A – Part 2:

The data types used for this report are very diverse considering the small scope of the reporting task. I will be using VARCHAR(VCHAR), INT, NUMERIC, and TIMESTAMP. A breakdown of the data types can be found below.

VARCHAR – A character data type that is used to store a variable length of characters. A positive integer enclosed in brackets must accompany this data type to denote the maximum length of the character string.

INT – A numeric data type that can store a maximum of 4 bytes. Holding values between 2147483648 to +2147483647.

NUMERIC – A numeric data type with user determined precision that can store a variable number of bytes. Holding values up to 131072 digits before the decimal point and up to 16383 digits after the decimal point.

TIMESTAMP – A date/time data type that can store up to 8 bytes. This data type is used to get both the date and time. This variation of TIMESTAMP does not include the time zone.

I will include the data types of the field in the visualizations in Part 1 of Section A so it is easier to associate which field is what type.

010743036
Brandon R Pinkston
3/2/2023

Section A – Part 3:

I will be using the “Payment” and “Staff” tables from the “dvdrental” database to pull all the raw data for the detailed table, and this same information will be used when transforming and loading the new data into the summary table.

Section A – Part 4:

Transformations 1: I will be taking the first and last name columns from the detailed table and concatenating them into a single full name for each staff member.

Transformation 2: I will extract the date and year from the “payment_date” and give them each their own column so I can group queries using them both or singularly.

Section A – Part 5:

The summary table serves the purpose of analyzing staff members’ performance in each month of each year. The scenario that I thought of for this data set was as follows: The DVD store in question pays out employee bonuses in two forms. The first form being a flat yearly bonus for working “X” number of hours in the store, and the second is a scaling monthly bonus dependent on the if they met “Y” sales in that month and had “Z” amount of DVD rentals. The summary table can then be used to have an automatically updated record of both the sum of DVD sales and the total count of DVD sales each employee makes every month. In the instance that these records are found to be incorrect in any way, the stored procedure will allow for a quick and efficient deletion, recalculation, transformation, and load of the data back into the database.

The detailed table section shows all of the information that is needed to come to a conclusion about the same questions that are being asked of the summary table but it would require much more effort by the user to come to these conclusions as they would manually have to count the total sum and count that each staff member had while simultaneously making sure that these figures only pertained to a single month of a single year. Transforming and aggregating the data like I did in the summary table makes things 10x easier to understand quickly with a high degree of accuracy.

The detailed table could primarily be used as a log of all of the orders that were executed by each of the two staff members on file. This is nice if there were any discrepancies between the data that has been stored and the actual amount of money that was physically there for the shop. This table can also be joined to the rental table to see a comprehensive list of which staff members sold the item, the amount paid, the customer id of the buyer, and the exact date and time the

transaction occurred. This can be handy in any number of scenarios, for instance a customer returning a DVD past their 30-day rental time limit or claiming their card was charged twice.

Section A – Part 6:

At a bare minimum the report should be refreshed at the end of every month so that the data can be aggregated and stored for each employee. Ideally the information should be updated every week or even every day so that it's easier to maintain and crosscheck the data for accuracy. The shareholders use the daily/weekly/monthly sales data to forecast future business decisions as well as make decisions on internal promotions and sales bonuses.

Section B:

Function 1 is used to concat the names of staff members from the detailed table.

```
create or replace function Transformations(staff_id1 int)
returns varchar
language plpgsql
as $$
declare full_name varchar;
begin
return (select first_name || ' ' || last_names from staff
   where staff_id = staff_id1);

end;
$$
```

010743036
Brandon R Pinkston
3/2/2023

```
create or replace function Transformations(staff_id1 int)
returns varchar
language plpgsql
as $$
declare full_name varchar;
begin
return (select first_name || ' ' || last_names from staff
       where staff_id = staff_id1);

end;
$$
```

Function 2 is used to extract either month or year data from the payment_date field included in the detailed table.

```
create or replace function Time_Transformations(decider int, date1 timestamp)
returns int
language plpgsql
as $$
declare month int;
declare year int;
begin
if decider = 1 then
select extract(month from date1) into month ;
return month;
elsif decider = 2 then
select extract(year from date1) into year ;
return year;

end if;
```

end;
$$

```
create or replace function Time_Transformations(decider int, date1 timestamp)
returns int
language plpgsql
as $$
declare month int;
declare year int;
begin
if decider = 1 then
select extract(month from date1) into month ;
return month;

elsif decider = 2 then
select extract(year from date1) into year ;
return year;

end if;
end;
$$
```

Section C:

```
create table detailed_table (
staff_id int,
first_name VARCHAR(100),
last_name VARCHAR(100),
amount numeric(10,2),
payment_date timestamp
);

create table summary_table (
staff_id int,
full_name VARCHAR(100),s
Sales_sum numeric(10,2),
Sales_count int,
payment_month int,
payment_year int
);
```

010743036
Brandon R Pinkston
3/2/2023

```
create table detailed_table (
staff_id int,
first_name VARCHAR(100),
last_name VARCHAR(100),
amount numeric(10,2),
payment_date timestamp
);

create table summary_table (
staff_id int,
full_name VARCHAR(100),s
Sales_sum numeric(10,2),
Sales_count int,
payment_month int,
payment_year int
);
```

Section D:

insert into detailed_table(
select staff_id, first_name, last_name, amount, payment_date
from staff inner join payment using(staff_id)
order by staff_id);

```
insert into detailed_table(
select staff_id, first_name, last_name, amount, payment_date
from staff inner join payment using(staff_id)
order by staff_id);
```

Section E:

create or replace function trig_function()

```
returns trigger
language plpgsql
as $$
begin
delete from summary_table;
insert into summary_table (
select staff_id, Transformations(staff_id) as full_name, sum(amount) as sales_sum,
count(amount) as sales_count, Time_Transformations(1, payment_date) as sales_month,
Time_Transformations(2, payment_date) as sales_year
from detailed_table
group by staff_id, sales_month, sales_year
order by staff_id, sales_year

);
return new;
end;
$$

Create trigger detailed_table_to_summary_table
after insert or update or delete
on detailed_table
for each statement
execute procedure trig_function();
```

```
create or replace function trig_function()
returns trigger
language plpgsql
as $$
begin
delete from summary_table;
insert into summary_table (
select staff_id, Transformations(staff_id) as full_name, sum(amount) as sales_sum,
count(amount) as sales_count, Time_Transformations(1, payment_date) as sales_month,
Time_Transformations(2, payment_date) as sales_year
from detailed_table
group by staff_id, sales_month, sales_year
order by staff_id, sales_year

);
return new;
end;
$$

Create trigger detailed_table_to_summary_table
after insert or update or delete
on detailed_table
for each statement
execute procedure trig_function();
```

Section F:

```
create or replace procedure clear_and_update_data()
language plpgsql
as $$
begin
truncate table detailed_table;
truncate table summary_table;
insert into detailed_table(
select staff_id, first_name, last_name, amount, payment_date
from staff inner join payment using(staff_id)
order by staff_id);
return;
end;
$$;
```

```
create or replace procedure clear_and_update_data()
language plpgsql
as $$
begin
truncate table detailed_table;
truncate table summary_table;
insert into detailed_table(
select staff_id, first_name, last_name, amount, payment_date
from staff inner join payment using(staff_id)
order by staff_id);
return;
end;
$$;
```

Section F – Q1:

A great scheduling tool that I would use is pgAgent. pgAgent is not installed with the pre-packaged PostgreSQL and pgAdmin download, but there is already capability baked into pgAdmin for scheduling with pgAgent. pgAgents configuration is also stored on the postgreSQL database cluster (Dias, 2023) so for me it would be the obvious and most convenient choice for job scheduling in the future.

Section H: Sources

*Sources below were used for section F-Q1 only, any other expressions in this report are 100% of my own creation and did not reference any other works*

Dias, Hugo. "An Overview of Job Scheduling Tools for Postgresql." *Severalnines*, 3 Feb. 2020, https://severalnines.com/blog/overview-job-scheduling-tools-postgresql/.

*Sources below were only used to reference SQL syntax*

"Create Trigger." *PostgreSQL Documentation*, 9 Feb. 2023, https://www.postgresql.org/docs/current/sql-createtrigger.html.

"Multiple Event SQL Triggers." *DB2 For i*, https://www.ibm.com/docs/en/i/7.1?topic=triggers-multiple-event-sql.

010743036
Brandon R Pinkston
3/2/2023