David McDougal

Feb 11, 2023

Advanced Data Management – D191

How Much Money Does Each Employee Generate Per Month?

Video URL: https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=accd1488-5ed2-4a51-8cc2-afa900ae46a7

A. The business problem that I chose to focus on was determining how much revenue each employee generated on a monthly basis. This information could then be used in employee and management one-on-ones to set and go over goals for the employee and for the business. It would be beneficial to the company to see if each employee is reaching the required standards and to determine any bonuses if the employee meets or exceeds expectations.

1. & 2. & 3. Below is the table for the fields that will be used in the detailed table:

Variable	From Table	Datatype	Description
staff_id	staff	integer	Each employees ID number.
amount	payment	numeric(5,2)	Amount of each rental.
payment_date	payment	timestamp	What day each rental occurred.
rental_id	payment	integer	ID for each rental.
first_name	staff	varchar(45)	First Name of the employee.
last_name	staff	varchar(45)	Last name of the Employee.
active	staff	varchar(10)	Determines if an employee is actively employed .
store_id	staff	smallint	What store does the employee work in?

Below is the table for the fields that will be used in the summary table:

Variable Name	From Table	Datatype	Description
employee_name	detailed	varchar(100)	The employee’s full name.
year_date	detailed	varchar(50)	This will be the transformed date from the timestamp for the year (ex. “2007”
month_date	detailed	varchar(50)	This will be the transformed date from a timestamp into a string formatted mont(ex. “Feb”)
monthly_revenue	detailed	numeric(15,2)	Sum revenue per employee.

4. One field that will need to be transformed in the detailed table is the active field. As it is right now it is a little difficult to just read as a lay user and to then understand “true” means “Active”, while “false” means “Not Active”.

5. The detailed view will create and show a breakdown of each rental with the which employee generated the rental and how much the rental was worth. This could be used a reference to see which employee/store was responsible for which rental and how much that rental generated.

The summary view will be used in monthly one-on-one’s to determine if the employee was able to hit a previously specified goal and from there either set new goals or find ways to improve.

6. This report should only be generated on a monthly basis before the monthly one-on-ones.

B. Provide code for transformation from part A4. This will be located in the insert query for the detailed table:

```
 CASE
        WHEN staff.active = true
            THEN 'Active'
        WHEN staff.active = false
            THEN 'Not Active'
        END active,
```

C. Provide code for the creation of the detailed and summary table.

Detailed table:

```
CREATE TABLE detailed (
    staff_id integer,
    amount numeric(5,2),
    payment_date timestamp,
    rental_id integer,
    first_name varchar(45),
    last_name varchar(45),
    active varchar(10),
    store_id smallint
);
```

Summary table:

```
CREATE TABLE summary (
    employee_name varchar(100),
    year_date varchar(50),
    month_date varchar(50),
    monthly_revenue numeric(15,2)
);
```

D. SQL Query to extract raw data and input into detailed table:

```
INSERT INTO detailed (
    staff_id, --staff
    amount, --payment
    payment_date, --payment
    rental_id,--payment
    first_name, --staff
    last_name, --staff
    active, --staff
    store_id --staff
)
SELECT
    staff.staff_id, payment.amount, payment.payment_date,
payment.rental_id, staff.first_name, staff.last_name,
        CASE
        WHEN staff.active = true
            THEN 'Active'
        WHEN staff.active = false
            THEN 'Not Active'
        END active,
    staff.store_id
FROM staff
INNER JOIN payment on payment.staff_id = staff.staff_id;
```

E. **Provide SQL Code that creates a trigger on the detailed table of the report to continually update Summary Table.**

```
-- Create Function
CREATE FUNCTION summary_refresh()
RETURNS TRIGGER AS $BODY$
BEGIN

DELETE FROM summary;
INSERT INTO summary(
SELECT
concat_ws(', ', last_name, first_name) AS employee_name,
extract(year from payment_date) As Year,
to_char(payment_date, 'Mon') AS Month,
sum(amount) AS monthly_revenue
FROM detailed
GROUP BY employee_name, Year, Month
ORDER BY employee_name, Year, Month
);

RETURN NEW;
END;
$BODY$ LANGUAGE plpgsql;

--Create Trigger
CREATE TRIGGER summary_refresh
AFTER INSERT ON detailed
FOR EACH STATEMENT
EXECUTE PROCEDURE summary_refresh();
```

F. **Provide an original stored procedure that can be used to refresh the data in detailed table and summary table.**
   a. **Identify a relevant job scheduling tool that can be used to automate the stored procedure:**
      i. I think a good tool for refreshing the data would be to use Agent pgAgent.
      ii. It would need to be refreshed every month so that the data is available for the employee's one-on-one.

```
CREATE PROCEDURE refresh_tables()
AS $BODY$
BEGIN

DELETE FROM detailed;

INSERT INTO detailed(
    staff_id, --staff
    amount, --payment
    payment_date, --payment
    rental_id,--payment
    first_name, --staff
    last_name, --staff
    active, --staff
    store_id --staff
)
SELECT
    staff.staff_id, payment.amount, payment.payment_date,
payment.rental_id, staff.first_name, staff.last_name,
        CASE
        WHEN staff.active = true
            THEN 'Active'
        WHEN staff.active = false
            THEN 'Not Active'
        END active,
    staff.store_id
FROM staff
INNER JOIN payment on payment.staff_id = staff.staff_id;
END;
$BODY$ LANGUAGE plpgsql;
```

G. Video Link:
https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=accd1488-5ed2-4a51-8cc2-afa900ae46a7

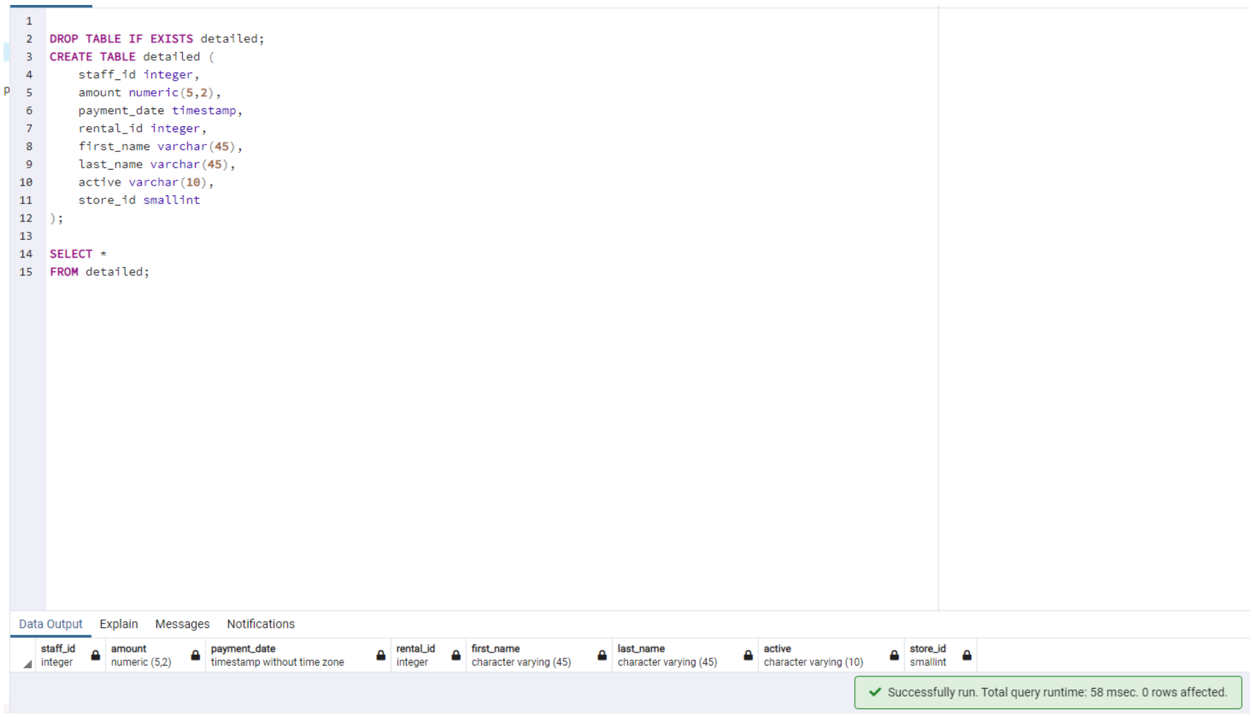
H. Sources used

a. **'In SQL(postgresql) How to group based on a "timestamp without time zone" column?', https://stackoverflow.com/questions/28039019/in-sqlpostgresql-how-to-group-based-on-a-timestamp-without-time-zone-column**
b. **'Create Trigger', https://hasura.io/learn/database/postgresql/triggers/1-create-trigger/**
c. **'An Overview of Job Scheduling Tools for PostgreSQL', Hugo Dias, Feb. 2020, https://severalnines.com/blog/overview-job-scheduling-tools-postgresql/**

I. Snap Shots of Successful Queries

a. Tables Created successfully

Detailed



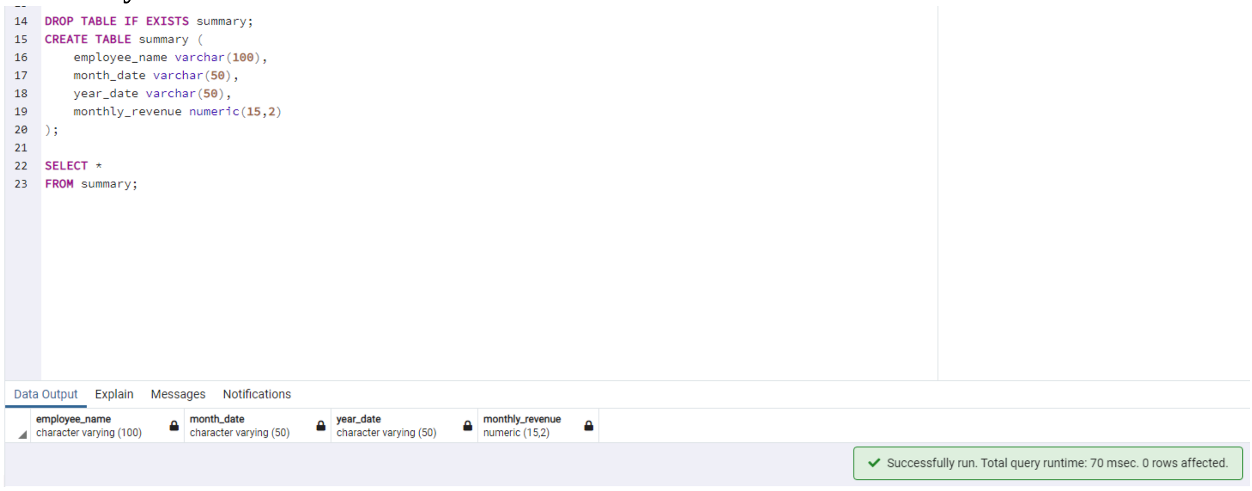
```

DROP TABLE IF EXISTS detailed;
CREATE TABLE detailed (
    staff_id integer,
    amount numeric(5,2),
    payment_date timestamp,
    rental_id integer,
    first_name varchar(45),
    last_name varchar(45),
    active varchar(10),
    store_id smallint
);

SELECT *
FROM detailed;
```

Data Output Explain Messages Notifications

staff_id integer	amount numeric (5,2)	payment_date timestamp without time zone	rental_id integer	first_name character varying (45)	last_name character varying (45)	active character varying (10)	store_id smallint

Successfully run. Total query runtime: 58 msec. 0 rows affected.

Summary



```
DROP TABLE IF EXISTS summary;
CREATE TABLE summary (
    employee_name varchar(100),
    month_date varchar(50),
    year_date varchar(50),
    monthly_revenue numeric(15,2)
);

SELECT *
FROM summary;
```

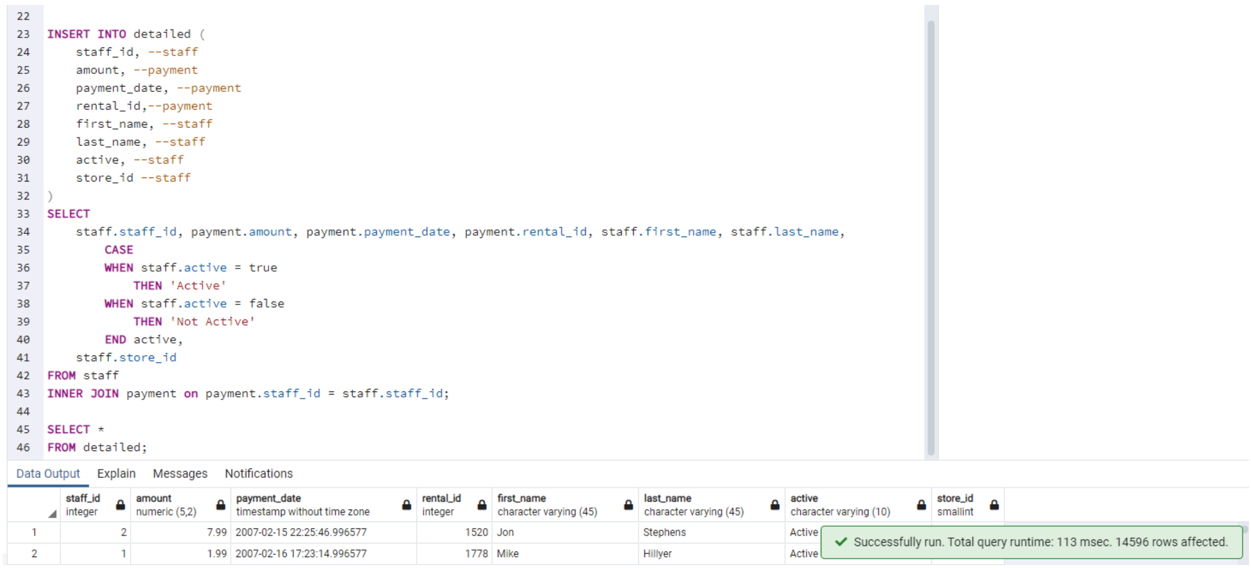
Data Output Explain Messages Notifications

employee_name character varying (100)	month_date character varying (50)	year_date character varying (50)	monthly_revenue numeric (15,2)

Successfully run. Total query runtime: 70 msec. 0 rows affected.

b. Tables filled with Data

Detailed



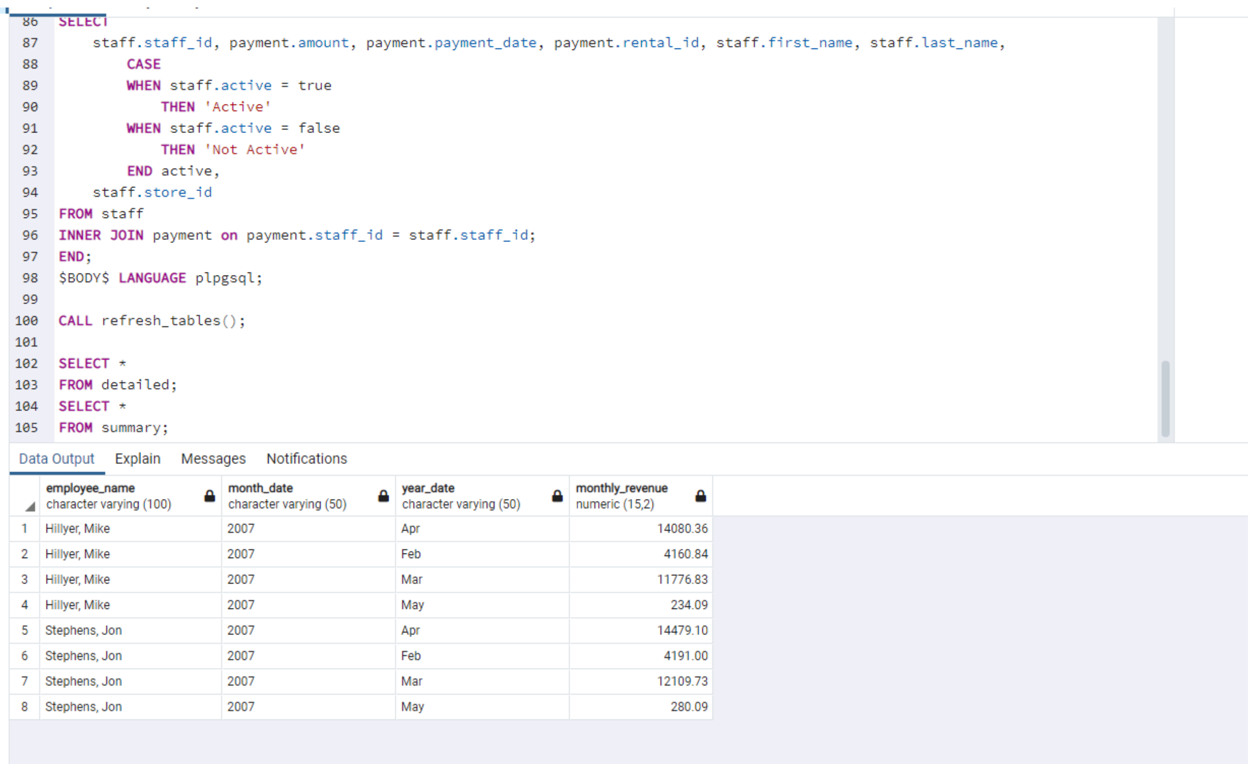
```
INSERT INTO detailed (
    staff_id, --staff
    amount, --payment
    payment_date, --payment
    rental_id,--payment
    first_name, --staff
    last_name, --staff
    active, --staff
    store_id --staff
)
SELECT
    staff.staff_id, payment.amount, payment.payment_date, payment.rental_id, staff.first_name, staff.last_name,
        CASE
        WHEN staff.active = true
            THEN 'Active'
        WHEN staff.active = false
            THEN 'Not Active'
        END active,
    staff.store_id
FROM staff
INNER JOIN payment on payment.staff_id = staff.staff_id;

SELECT *
FROM detailed;
```

Data Output Explain Messages Notifications

	staff_id integer	amount numeric (5,2)	payment_date timestamp without time zone	rental_id integer	first_name character varying (45)	last_name character varying (45)	active character varying (10)	store_id smallint
1	2	7.99	2007-02-15 22:25:46.996577	1520	Jon	Stephens	Active	
2	1	1.99	2007-02-16 17:23:14.996577	1778	Mike	Hillyer	Active	

✓ Successfully run. Total query runtime: 113 msec. 14596 rows affected.

Summary



```
SELECT
    staff.staff_id, payment.amount, payment.payment_date, payment.rental_id, staff.first_name, staff.last_name,
        CASE
        WHEN staff.active = true
            THEN 'Active'
        WHEN staff.active = false
            THEN 'Not Active'
        END active,
    staff.store_id
FROM staff
INNER JOIN payment on payment.staff_id = staff.staff_id;
END;
$BODY$ LANGUAGE plpgsql;

CALL refresh_tables();

SELECT *
FROM detailed;
SELECT *
FROM summary;
```

Data Output Explain Messages Notifications

	employee_name character varying (100)	month_date character varying (50)	year_date character varying (50)	monthly_revenue numeric (15,2)
1	Hillyer, Mike	2007	Apr	14080.36
2	Hillyer, Mike	2007	Feb	4160.84
3	Hillyer, Mike	2007	Mar	11776.83
4	Hillyer, Mike	2007	May	234.09
5	Stephens, Jon	2007	Apr	14479.10
6	Stephens, Jon	2007	Feb	4191.00
7	Stephens, Jon	2007	Mar	12109.73
8	Stephens, Jon	2007	May	280.09

Completed Code:

```sql
DROP TABLE IF EXISTS detailed;
CREATE TABLE detailed (
    staff_id integer,
    amount numeric(5,2),
    payment_date timestamp,
    rental_id integer,
    first_name varchar(45),
    last_name varchar(45),
    active varchar(10),
    store_id smallint
);

DROP TABLE IF EXISTS summary;
CREATE TABLE summary (
    employee_name varchar(100),
    month_date varchar(50),
    year_date varchar(50),
    monthly_revenue numeric(15,2)
);

INSERT INTO detailed (
    staff_id, --staff
    amount, --payment
    payment_date, --payment
    rental_id,--payment
    first_name, --staff
    last_name, --staff
    active, --staff
    store_id --staff
)
SELECT
    staff.staff_id, payment.amount, payment.payment_date,
payment.rental_id, staff.first_name, staff.last_name,
        CASE
        WHEN staff.active = true
            THEN 'Active'
        WHEN staff.active = false
            THEN 'Not Active'
        END active,
    staff.store_id
FROM staff
INNER JOIN payment on payment.staff_id = staff.staff_id;

-- Create Function
CREATE FUNCTION summary_refresh_new()
RETURNS TRIGGER AS $BODY$ --
https://hasura.io/learn/database/postgresql/triggers/1-create-trigger/
BEGIN

DELETE FROM summary;
INSERT INTO summary(
SELECT
concat_ws(', ', last_name, first_name) AS employee_name,
to_char(payment_date, 'Mon') AS Month,
extract(year from payment_date) As Year,
sum(amount) AS monthly_revenue
FROM detailed
GROUP BY employee_name, Year, Month
ORDER BY employee_name, Year, Month
);

RETURN NEW;
END;
$BODY$ LANGUAGE plpgsql;

--Create Trigger
CREATE TRIGGER summary_refresh
AFTER INSERT ON detailed
FOR EACH STATEMENT
EXECUTE PROCEDURE summary_refresh_new();

CREATE PROCEDURE refresh_tables()
AS $BODY$
BEGIN

DELETE FROM detailed;

INSERT INTO detailed(
    staff_id, --staff
    amount, --payment
    payment_date, --payment
    rental_id,--payment
    first_name, --staff
    last_name, --staff
```

```sql
    active, --staff
    store_id --staff
)
SELECT
    staff.staff_id, payment.amount, payment.payment_date,
payment.rental_id, staff.first_name, staff.last_name,
        CASE
        WHEN staff.active = true
            THEN 'Active'
        WHEN staff.active = false
            THEN 'Not Active'
        END active,
    staff.store_id
FROM staff
INNER JOIN payment on payment.staff_id = staff.staff_id;
END;
$BODY$ LANGUAGE plpgsql;

CALL refresh_tables();

SELECT *
FROM detailed;
SELECT *
FROM summary;
```