D191 – Performance Assessment

Matthew Scillitani

IMPORTANT: The code in its entirety, with comments, can be found at the bottom of this paper.

Section A: Real-world business report

(Part A) A **business problem** of DVD Rental Company's is that managers can't easily see the sales revenue of their employees every quarter. (A1) The **data used** for the report will require payment and staff information. Specifically needed are the payment and staff ID numbers, which are INTEGER type, the amount of money transferred, which is NUMERIC type, and the names of staff members, which is VARCHAR type. (A2) The **two tables** necessary to find this information are "payment" and "staff".

(A3) The **specific fields that will be included** in the **detailed report** are payment_id, amount, staff_id, and full_name. For the **summary report**, the amount and full_name fields are required and will be extracted from the detailed report. (A4) The field full_name will require a **transformation** to concatenate the first_name and last_name fields for easier viewing of data.

(A5) A **business use** of the detailed report is being able to view all the payment transactions every employee has ever processed. A **business use** of the summary report is that the total sales revenue for every employee can be seen, allowing managers to assess staff performance.

(A6) These reports **should be refreshed** quarterly before every quarterly performance report. This frequency is standard for most e-commerce businesses and will allow managers to see how well each staff member is performing and to update stakeholders regarding quarterly revenue.

Section B: SQL code to create detailed and summary tables

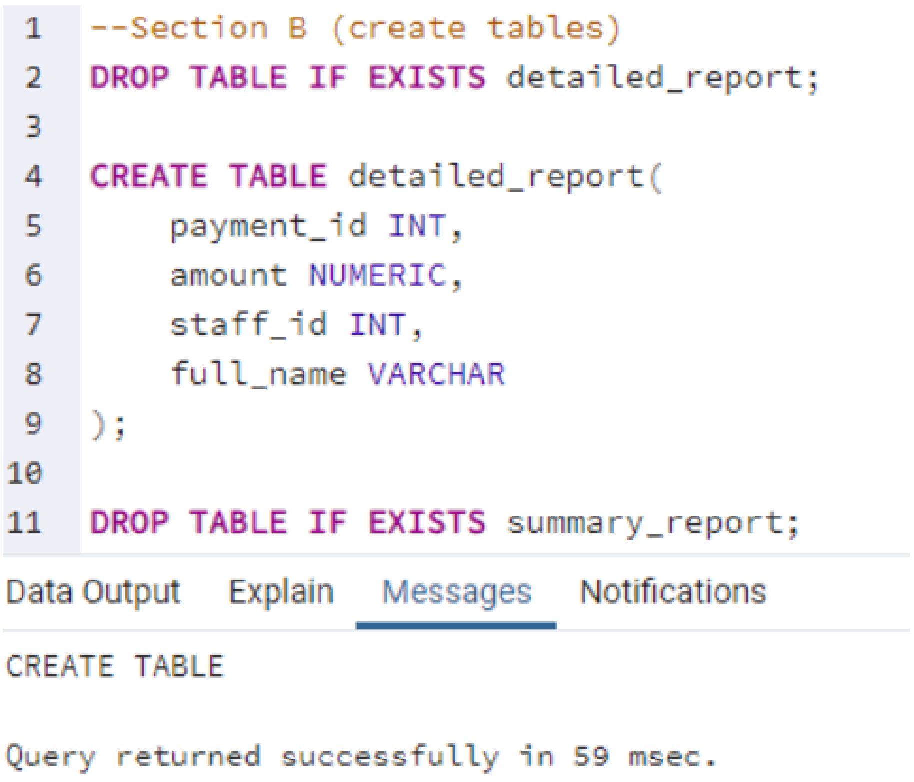
--Detailed Report

```
DROP TABLE IF EXISTS detailed_report;
CREATE TABLE detailed_report(
    payment_id INT,
    amount NUMERIC,
    staff_id INT,
    full_name VARCHAR
);
```

--Summary Report

```
DROP TABLE IF EXISTS summary_table;
CREATE TABLE summary_report (
    full_name VARCHAR,
    total NUMERIC,
    transactions INT
);
```

Verification of section B:



```
--Section B (create tables)
DROP TABLE IF EXISTS detailed_report;

CREATE TABLE detailed_report(
    payment_id INT,
    amount NUMERIC,
    staff_id INT,
    full_name VARCHAR
);

DROP TABLE IF EXISTS summary_report;
```

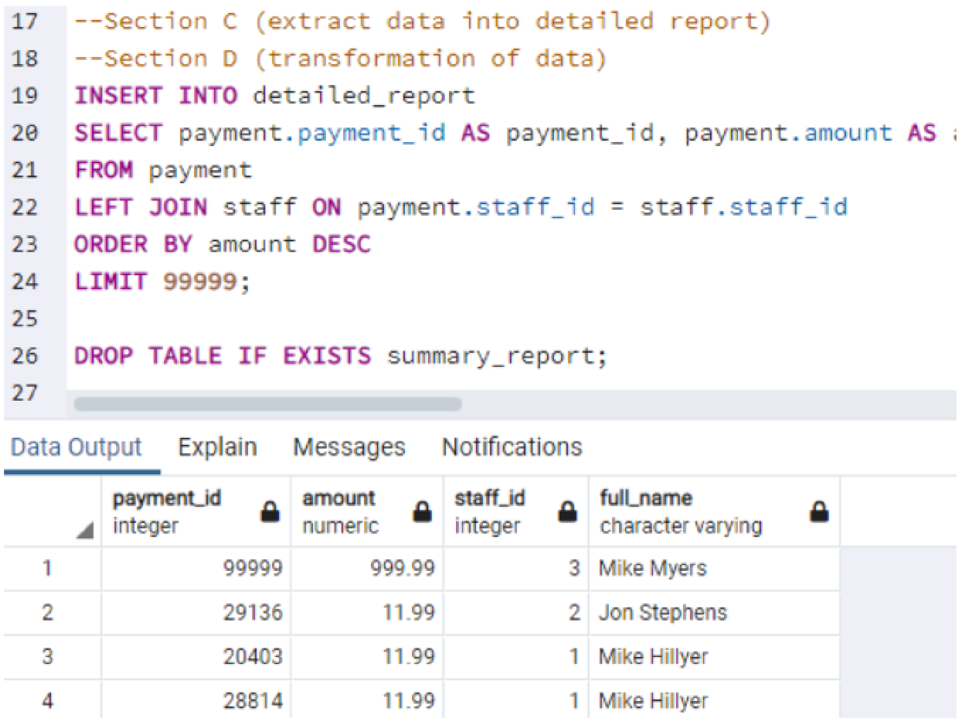
```
CREATE TABLE

Query returned successfully in 59 msec.
```

Section C: SQL query to extract raw data for the Detailed report

```
INSERT INTO detailed_report
```

SELECT payment.payment_id AS payment_id, payment.amount AS amount, staff.staff_id AS staff_id, CONCAT(staff.first_name, ' ', staff.last_name) AS full_name
FROM payment
LEFT JOIN staff ON payment.staff_id = staff.staff_id
ORDER BY amount DESC
LIMIT 99999;

Verification of section C:



```
--Section C (extract data into detailed report)
--Section D (transformation of data)
INSERT INTO detailed_report
SELECT payment.payment_id AS payment_id, payment.amount AS 
FROM payment
LEFT JOIN staff ON payment.staff_id = staff.staff_id
ORDER BY amount DESC
LIMIT 99999;

DROP TABLE IF EXISTS summary_report;

```

Data Output Explain Messages Notifications

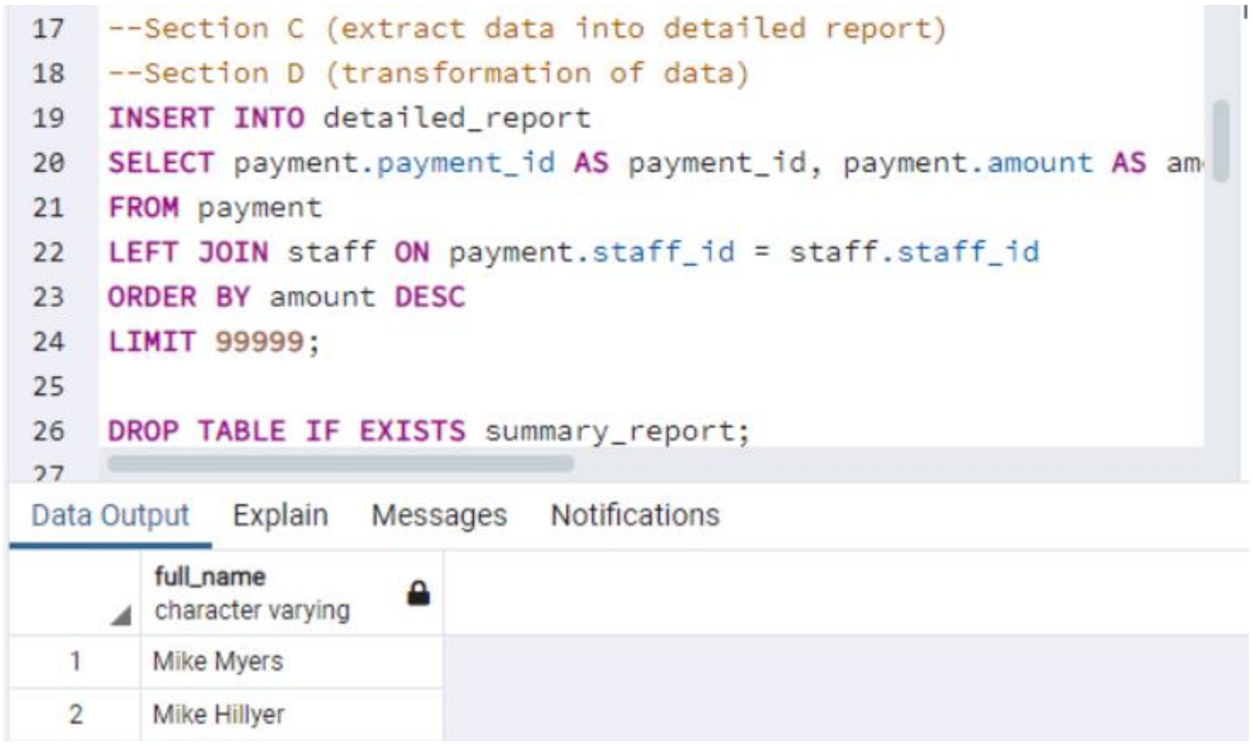
	payment_id integer	amount numeric	staff_id integer	full_name character varying
1	99999	999.99	3	Mike Myers
2	29136	11.99	2	Jon Stephens
3	20403	11.99	1	Mike Hillyer
4	28814	11.99	1	Mike Hillyer

Section D: Function to perform transformation from part A4

--The transformation to concatenate first_name and last_name from the staff table to full_name in the detailed report.

INSERT INTO detailed_report
SELECT payment.payment_id AS payment_id, payment.amount AS amount, staff.staff_id AS staff_id, CONCAT(staff.first_name, ' ', staff.last_name) AS full_name
FROM payment
LEFT JOIN staff ON payment.staff_id = staff.staff_id
ORDER BY amount DESC
LIMIT 99999;

Verification of section D:



```
--Section C (extract data into detailed report)
--Section D (transformation of data)
INSERT INTO detailed_report
SELECT payment.payment_id AS payment_id, payment.amount AS am
FROM payment
LEFT JOIN staff ON payment.staff_id = staff.staff_id
ORDER BY amount DESC
LIMIT 99999;

DROP TABLE IF EXISTS summary_report;
```

	full_name character varying
1	Mike Myers
2	Mike Hillyer

Section E: Trigger on detailed table

--Trigger should be called whenever a staff member's name is changed or added. For example, if someone is married or a new employee is hired. Names must be correct to keep track of performance.

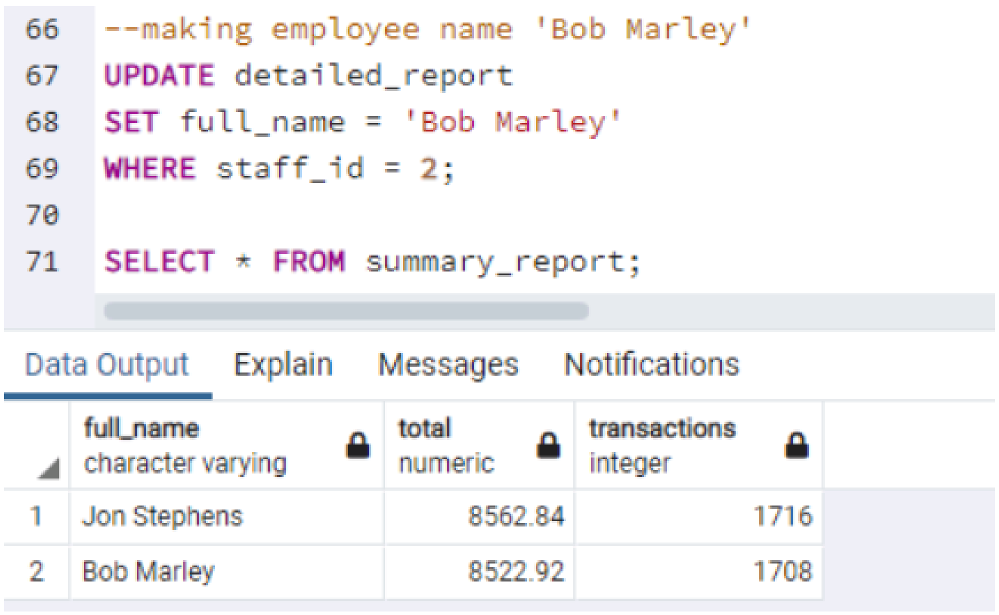
--Function called by trigger

```
CREATE OR REPLACE FUNCTION update_tables()
    RETURNS TRIGGER
    LANGUAGE plpgsql
AS $$
BEGIN
    UPDATE summary_report
    SET full_name = NEW.full_name
    WHERE full_name = OLD.full_name;
RETURN NEW;
END;
$$;
```

--Trigger

CREATE TRIGGER table_updater
    BEFORE UPDATE
    ON detailed_report
    FOR EACH ROW
    EXECUTE FUNCTION update_tables();

Verification of section E:



```
--making employee name 'Bob Marley'
UPDATE detailed_report
SET full_name = 'Bob Marley'
WHERE staff_id = 2;

SELECT * FROM summary_report;
```

	full_name character varying	total numeric	transactions integer
1	Jon Stephens	8562.84	1716
2	Bob Marley	8522.92	1708

Section F: Stored procedure to refresh data in both tables

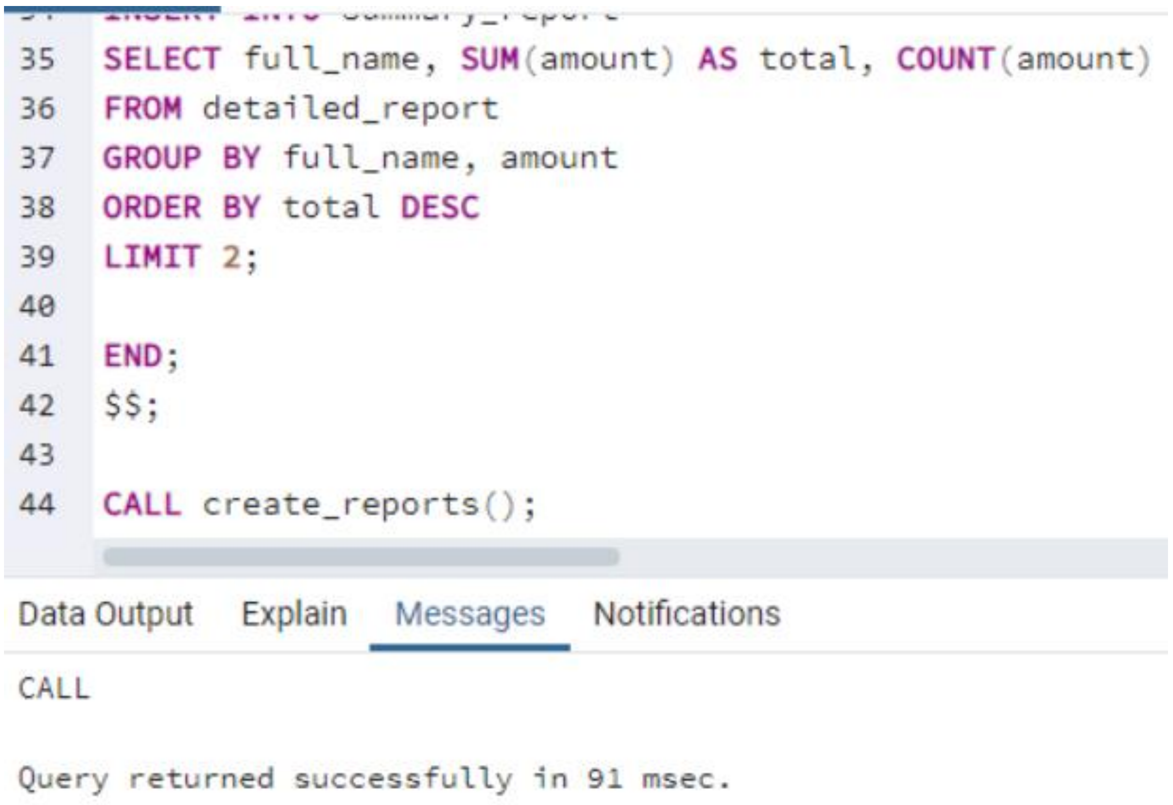
--Procedure should be called **every quarter** for quarterly performance review. These reports may either be scheduled automatically or manually. In the former case, **a job scheduler such as pgAgent** can be set to run the procedure every quarter. In the latter case, someone can go into the DVD rentals database's query tool and use **CALL create_reports();** to refresh the reports and then use either **SELECT * FROM summary_report;** or **SELECT * FROM detailed_report;** to view the summary or detailed reports, respectively.

CREATE OR REPLACE PROCEDURE public.create_reports()
LANGUAGE plpgsql
AS $$

```
BEGIN

DROP TABLE IF EXISTS detailed_report;

CREATE TABLE detailed_report(
    payment_id INT,
    amount NUMERIC,
    staff_id INT,
    full_name VARCHAR
);

INSERT INTO detailed_report
SELECT payment.payment_id AS payment_id, payment.amount AS amount,
    staff.staff_id AS staff_id, CONCAT(staff.first_name, ' ', staff.last_name)
    AS full_name
FROM payment
LEFT JOIN staff ON payment.staff_id = staff.staff_id
ORDER BY amount DESC
LIMIT 99999;

DROP TABLE IF EXISTS summary_report;

CREATE TABLE summary_report (
    full_name VARCHAR,
    total NUMERIC,
    transactions INT
);

INSERT INTO summary_report
SELECT full_name, SUM(amount) AS total, COUNT(amount) AS transactions
FROM detailed_report
GROUP BY full_name, amount
ORDER BY total DESC
LIMIT 2;

END;
$$;

CALL create_reports();

CREATE OR REPLACE FUNCTION update_tables()
    RETURNS TRIGGER
    LANGUAGE plpgsql
AS $$
```

```
BEGIN
    UPDATE summary_report
    SET full_name = NEW.full_name
    WHERE full_name = OLD.full_name;
RETURN NEW;
END;
$$;
```

Verification of section F:



```
SELECT full_name, SUM(amount) AS total, COUNT(amount)
FROM detailed_report
GROUP BY full_name, amount
ORDER BY total DESC
LIMIT 2;

END;
$$;

CALL create_reports();
```

```
CALL

Query returned successfully in 91 msec.
```

Section H: Web Sources

No web sources were used to acquire data or code.

All the code put together

--Procedure, to be called every quarter

```
CREATE OR REPLACE PROCEDURE public.create_reports()
LANGUAGE plpgsql
AS $$
BEGIN
```

```
DROP TABLE IF EXISTS detailed_report;

--Detailed report
CREATE TABLE detailed_report(
        payment_id INT,
        amount NUMERIC,
        staff_id INT,
        full_name VARCHAR
);

--Extracting data into detailed report
INSERT INTO detailed_report
SELECT payment.payment_id AS payment_id, payment.amount AS amount,
staff.staff_id AS staff_id, CONCAT(staff.first_name, ' ', staff.last_name) AS
full_name
FROM payment
LEFT JOIN staff ON payment.staff_id = staff.staff_id
ORDER BY amount DESC
LIMIT 99999;

DROP TABLE IF EXISTS summary_report;

--Summary report
CREATE TABLE summary_report (
        full_name VARCHAR,
        total NUMERIC,
        transactions INT
);

--Extracting data into summary report
INSERT INTO summary_report
SELECT full_name, SUM(amount) AS total, COUNT(amount) AS transactions
FROM detailed_report
GROUP BY full_name, amount
ORDER BY total DESC
LIMIT 2;

END;
$$;
--End of procedure

--Calling procedure
CALL create_reports();
```

```
--Create function to be triggered whenever new hire or name
changed (i.e., married)
CREATE OR REPLACE FUNCTION update_tables()
        RETURNS TRIGGER
        LANGUAGE plpgsql
AS $$
BEGIN
        UPDATE summary_report
        SET full_name = NEW.full_name
        WHERE full_name = OLD.full_name;
RETURN NEW;
END;
$$;

--Trigger
CREATE TRIGGER table_updater
        BEFORE UPDATE
        ON detailed_report
        FOR EACH ROW
        EXECUTE FUNCTION update_tables();

--Updating data to verify it works
UPDATE detailed_report
SET full_name = 'Elvis Presley'
WHERE staff_id = 2;

--To verify
SELECT * FROM summary_report;
```