A. Summarize one real-world written business report that can be created from the DVD Dataset from the “Labs on Demand Assessment Environment and DVD Database” attachment.

One possible business report that can be created is a report that summarizes the revenue generated by different films and categories, by month. The detailed table will include data about the film, the month it was rented, and the amount paid per rental, along with the category of the film. This can allow business users to determine which films do or do not generate a significant amount of revenue, giving them the insight needed to decide which films to buy more copies of, or which films to remove from store shelves.

The summary table will pull from the detailed table the category, the amount of revenue generated for the category, and the month the category generated revenue in, or how much payment was received for each category in each month. This will allow business users to determine which category of film should be promoted more during each month.

1. Identify the specific fields that will be included in the detailed table and the summary table of the report.

Summary table: category_name (VARCHAR(25)), month_rented(VARCHAR(9)),
total_payment_amount(NUMERIC(12,2)), category_id(INT)
PRIMARY KEY (category_id, month_rented),
FOREIGN KEY (category_id) REFERENCES category (category_id)

Detailed table: category_name(VARCHAR(25)), month_rented(VARCHAR(9)),
payment_amount(NUMERIC(12,2)), film_title(VARCHAR(225)), film_id(INT), rental_id(INT)
PRIMARY KEY (film_id, rental_id)
FOREIGN KEY (film_id) REFERENCES film (film_id),
FOREIGN KEY (rental_id) REFERENCES rental(rental_id),

2. Describe the types of data fields used for the report.

Data types used are VARCHAR(lengths of 9, 25, and 255), NUMERIC(12,3), and INT.

3. Identify *at least* two specific tables from the given dataset that will provide the data necessary for the detailed table section and the summary table section of the report.

The “payment” table is necessary to find the amount paid by the customer, and the “category” table is necessary to find the name of the category each film is in.

4. Identify *at least* one field in the detailed table section that will require a custom transformation with a user-defined function and explain why it

should be transformed (e.g., you might translate a field with a value of N to No and Y to Yes).

The rental_date column is changed to a month column (ie. 2015-05-25 gets transformed into ‘May’), allowing for readability, and to allow the data rental store to more easily determine which films or categories are more popular based on the time of year.

5. Explain the different business uses of the detailed table section and the summary table section of the report.

The summary table can be used to determine the popularity of each category by time of year, allowing stores to determine which movie they should purchase at any given time, assuming other factors have been accounted for (basically, if a store has determined that, among a selection of different movies, if they all seem like equally likely candidates for the store to purchase, the store can decide that, since Drama films tend to do better in July than August, if it’s July, perhaps a Drama should be chosen. Of course, only using data from one year can make that call less certain than if more data was available.)

The detailed report could be used to look at which movies in each category perform the best, or perhaps which movies in general perform well. This allows stores to select certain films to remove from store shelves, or perhaps which films to promote, or buy more copies of.

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

Of course, as the summary report is based on months, reports should be refreshed monthly. Also, it may be a good idea to create different tables and reports for each year, so that years can individually be compared, or the same month for two different years can be prepared.

B. Provide original code for function(s) in text format that perform the transformation(s) you identified in part A4.

```
CREATE FUNCTION date_to_month(rent_date TIMESTAMP)
RETURNS VARCHAR(9)
LANGUAGE plpgsql
AS
$$
DECLARE month_var VARCHAR(9);
BEGIN
SELECT to_char(rent_date, ‘Month’) INTO month_var;
RETURN month_var;
END;
$$;
```

C. Provide original SQL code in a text format that creates the detailed and summary tables to hold your report table sections.

```
DROP TABLE IF EXISTS detailed_report;
CREATE TABLE detailed_report (
	category_name VARCHAR(25),
	month_rented VARCHAR(9),
	payment_amount  NUMERIC(12, 2),
	film_title VARCHAR(225),
	film_id INT,
	rental_id INT,
	FOREIGN KEY (film_id) REFERENCES film (film_id),
	FOREIGN KEY (rental_id) REFERENCES rental(rental_id),
	PRIMARY KEY (film_id, rental_id)
);

INSERT INTO detailed_report
SELECT ca.name, date_to_month(r.rental_date), ROUND(AVG(pa.amount), 2), f.title, f.film_id, r.rental_id
FROM film_category AS fc
JOIN category AS ca ON fc.category_id = ca.category_id
JOIN inventory AS i ON fc.film_id = i.film_id
JOIN film AS f ON f.film_id = fc.film_id
JOIN rental AS r ON i.inventory_id = r.inventory_id
JOIN payment AS pa ON r.rental_id = pa.rental_id
GROUP BY r.rental_id, f.film_id, f.title, r.rental_date, ca.name;
```

The odd aggregation in the INSERT statement is due to a very strange group of entries in the database. Rental_id 4591, film_id 492 has 5 different entries, all with the same exact rental_date, rental_duration, and return_date. However, it also has varying payment amounts. 0.99, 1.99, 1.99, 3.99, 3.99, again, all with the same rental_id, film_id, and start and end dates for the duration of the rent. Ideally, I would speak to someone within the store and try to figure out how this happened.

This specific issue originates from the “payment” table, where these entries have different information, including payment_date. However, this rental_id is still the only rental_id with multiple entries in this table, as seen by the query:

```
SELECT COUNT(rental_id) FROM payment
GROUP BY rental_id
ORDER BY COUNT(rental_id) DESC;
```

This returns a table where one row has 5, and the rest have 1. I was unsure of how to deal with this, and since I have (film_id, rental_id) as my primary key in the detailed_report table, having multiple entries with the same primary key seemed problematic, so I averaged the payment amount for these entries, and since there is now an aggregate function on a column, the other columns needed to be in a GROUP BY clause. This should not affect the rest of the data, and should only affect these 5 similar entries.

```
DROP TABLE IF EXISTS summary_report;
CREATE TABLE summary_report (
        category_id INT,
        month_rented VARCHAR(9),
        category_name VARCHAR(25),
        total_payment_amount NUMERIC(12, 2),
        PRIMARY KEY (category_id, month_rented),
        FOREIGN KEY (category_id) REFERENCES category (category_id)
);

INSERT INTO summary_report
SELECT ca.category_id, dr.month_rented, dr.category_name, SUM(dr.peyment_amount)
FROM detailed_report AS dr
JOIN film_category AS fc ON dr.film_id = fc.film_id
JOIN category AS ca ON fc.category_id = ca.category_id
GROUP BY ca.category_id, dr.category_name, dr.month_rented;
```

The aggregation of this table is, unlike the previous, meant to allow a user to determine how much each category makes, and in what months they make the most, or which category makes the most in each month.

D. Provide an original SQL query in a text format that will extract the raw data needed for the detailed section of your report from the source database.

Provided above, in part C as part of the larger code text.

E. Provide original SQL code in a text format that creates a trigger on the detailed table of the report that will continually update the summary table as data is added to the detailed table.

```
CREATE OR REPLACE FUNCTION summary_trigger_function()
RETURNS TRIGGER
LAGUAGE plpgsql
AS
$$
BEGIN
DELETE FROM summary_report;
INSERT INTO summary_report
SELECT ca.category_id, dr.month_rented, dr.category_name, SUM(dr.peyment_amount)
FROM detailed_report AS dr
JOIN film_category AS fc ON dr.film_id = fc.film_id
JOIN category AS ca ON fc.category_id = ca.category_id
GROUP BY ca.category_id, dr.category_name, dr.month_rented;
RETURN NEW;
END;
$$;

CREATE TRIGGER update_summary
```

```
AFTER INSERT
ON detailed_report
FOR EACH STATEMENT
EXECUTE PROCEDURE summary_trigger_function();
```

F. Provide an original stored procedure in a text format that can be used to refresh the data in *both* the detailed table and summary table. The procedure should clear the contents of the detailed table and summary table and perform the raw data extraction from part D.

```
CREATE OR REPLACE PROCEDURE refresh_tables()
LANGUAGE plpgsql
AS
$$
BEGIN
DELETE FROM detailed_report;
INSERT INTO detailed_report
SELECT ca.name, date_to_month(r.rental_date), ROUND(AVG(pa.amount), 2), f.title, f.film_id, r.rental_id
FROM film_category AS fc
JOIN category AS ca ON fc.category_id = ca.category_id
JOIN inventory AS i ON fc.film_id = i.film_id
JOIN film AS f ON f.film_id = fc.film_id
JOIN rental AS r ON i.inventory_id = r.inventory_id
JOIN payment AS pa ON r.rental_id = pa.rental_id
GROUP BY r.rental_id, f.film_id, f.title, r.rental_date, ca.name;
RETURN;
END;
$$

CALL refresh_tables();
```

Since the summary_trigger_function() exists, after the insert statement runs, the summary_report table should automatically be updated to have the correct data.

1. Identify a relevant job scheduling tool that can be used to automate the stored procedure.

I would likely use pg_cron for this, as the automation should be fairly simple.

```
CREATE EXTENSION pg_cron;

SELECT cron.schedule(
        ‘monthly_table_refresh’,
        ‘0 17 $ * *’,
        ‘CALL refresh_tables()’
);
```

This would automatically call the refresh_tables procedure at 17:00, or 5:00PM on the last day of each month. It is important to note that pg_cron can only run while the computer it is installed on is active.

G. Provide a Panopto video recording that includes the presenter and a vocalized demonstration of the functionality of the code used for the analysis.

https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=b4497b68-c38a-44e4-8ce7-b367013df2b9