CA2 Individual Report

Name	Jovan Yeo Zhan Rui
Student Id	P2322881
Class	DIT/2B/21
Github Repository URL	https://github.com/soc-DBS/dbs-assignment-JovanYe0
Github Account ID	JovanYe0

For each criterion, provide the relevant evidence such as screenshots, explanation and links to pull requests/commits/files that demonstrate the fulfilment of the requirement. Replace each "**?**" with your Self Rating.

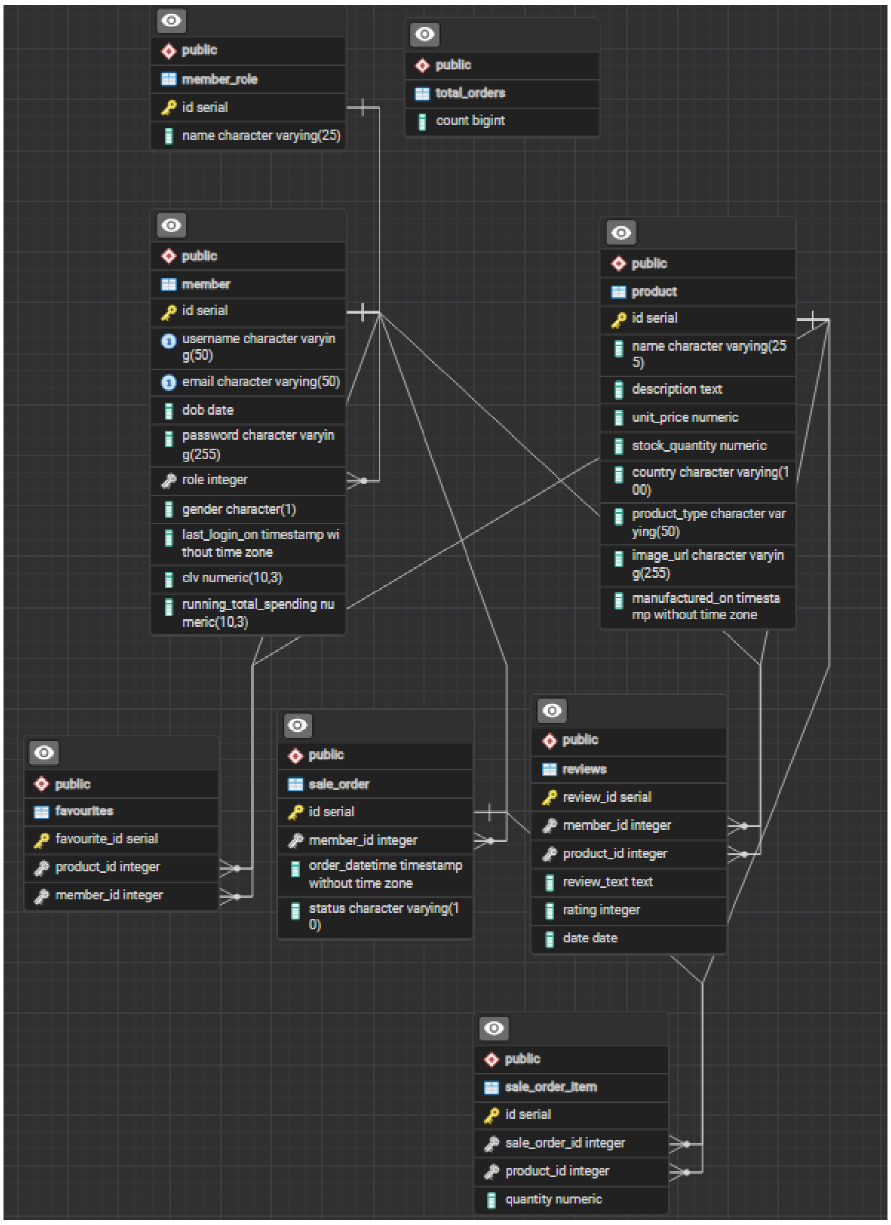
For Self Rating, you may rate yourself accordingly if you feel that you:

0 Have little or **no** understanding. and did not attempt the requirement.
1 Have **limited** understanding to demonstrate competency for the criterion.
2 Have **basic** understanding and only able to replicate examples from tutorials and practical.
3 Have **adequate** understanding and can apply what you have learnt to fulfil requirements.
4 Have **solid** understanding in the specific criterion, able to extend from what you have learnt to deliver on the requirements without much references.
5 Have **excellent** understanding and delivered the requirements according to latest industry guidelines, best practices and documentations.

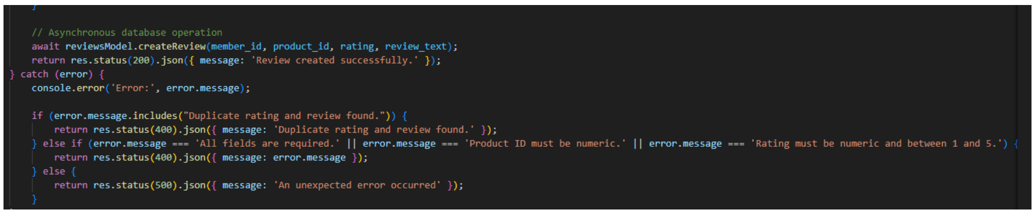
Important

a) You are required to provide for each criterion:
   - A brief **description**
   - **One to two** of your best implementations with URL **link** to respective repository pull request/commits/files.
   - You should also provide **screenshots** where relevant.

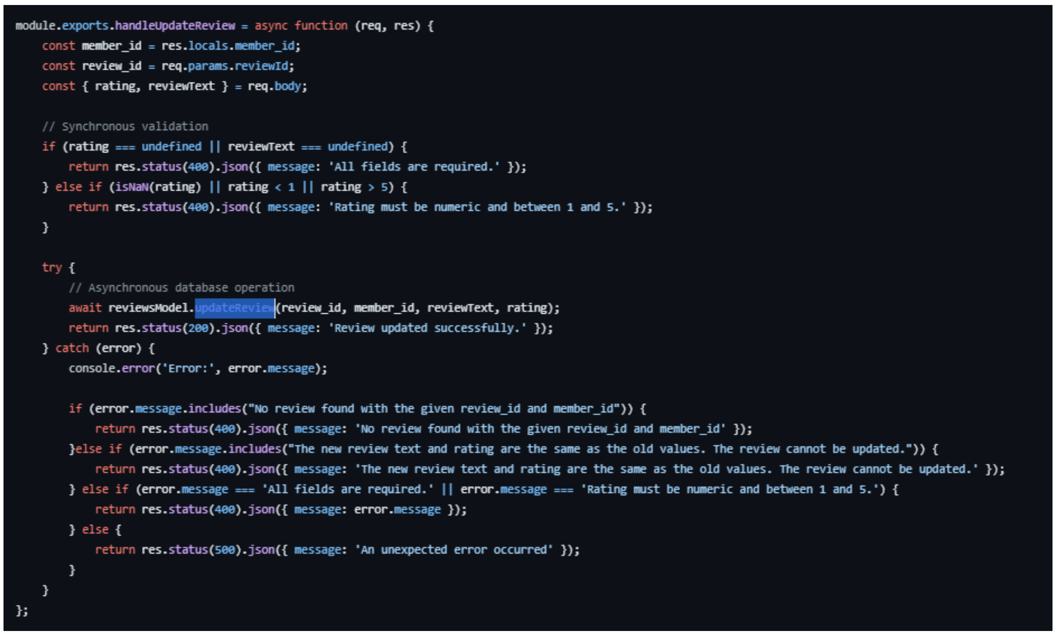
b) You are to ensure the hyperlink in this document works. **Failure to do so will result in a 50% deduction of marks.**

No.	Criterion	Describe What Was Done	Self Rating
1	Database Design	• *Provide clear screenshot of ERD or link to screenshot of ERD* • *What are the new entities introduced?* • *What are the DDL or create table statements?*  public member_role id serial name character varying(25) public total_orders count bigint public member id serial username character varying(50) email character varying(50) dob date password character varying(255) role integer gender character(1) last_login_on timestamp without time zone clv numeric(10,3) running_total_spending numeric(10,3) public product id serial name character varying(255) description text unit_price numeric stock_quantity numeric country character varying(100) product_type character varying(50) image_url character varying(255) manufactured_on timestamp without time zone public favourites favourite_id serial product_id integer member_id integer public sale_order id serial member_id integer order_datetime timestamp without time zone status character varying(10) public reviews review_id serial member_id integer product_id integer review_text text rating integer date date public sale_order_item id serial sale_order_id integer product_id integer quantity numeric The new table that I have done is reviews and favourites with the create table statement for review being CREATE TABLE public.reviews (review_id integer NOT NULL, member_id integer NOT NULL, product_id integer NOT NULL, review_text text, rating integer NOT NULL, date date DEFAULT CURRENT_DATE);	4/5

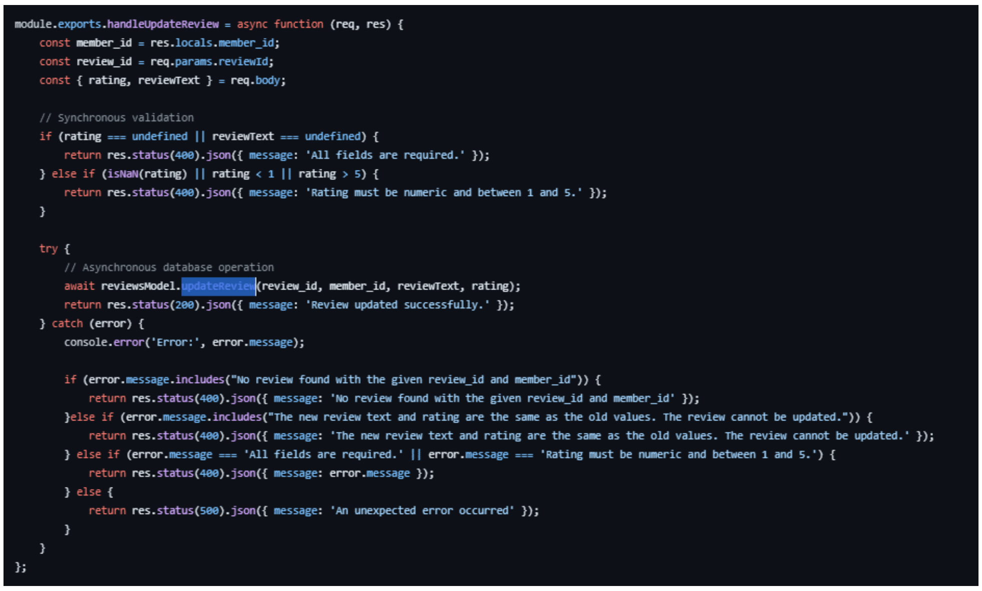
		While favourites create table statement is
		
		CREATE TABLE Favourites (
		    favourite_id SERIAL NOT NULL,
		    product_id int NOT NULL,
		    member_id int NOT NULL,
		    PRIMARY KEY (favourite_id),
		    FOREIGN KEY (product_id) REFERENCES Product(id),
		FOREIGN KEY (member_id) REFERENCES member(id)
		);
		
		link	

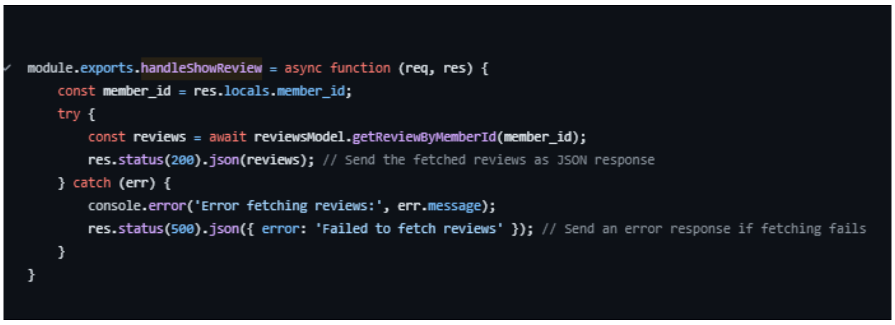
2	Create Entity	• *What is the database stored procedure to create new entities?* • *What are the backend functions (eg model, controller, route) & the front-end changes? (eg html, js)?* • *Any screenshots of how you test for errors and success?* Some of the stored procedure are like create_review where it takes in input from the front end form through the controller and model then is inserted into the review table, the error handling is if there is already a review with the same review text and rating it would refuse to add it in as it is a duplicate review while also ensuring all text box is filled with rating no more than 5 or less than 1 If it passes all of these check the review is created  stored procedure link	4/5

```
        // Asynchronous database operation
        await reviewsModel.createReview(member_id, product_id, rating, review_text);
        return res.status(200).json({ message: 'Review created successfully.' });
    } catch (error) {
        console.error('Error:', error.message);

        if (error.message.includes("Duplicate rating and review found.")) {
            return res.status(400).json({ message: 'Duplicate rating and review found.' });
        } else if (error.message === 'All fields are required.' || error.message === 'Product ID must be numeric.' || error.message === 'Rating must be numeric and between 1 and 5.') {
            return res.status(400).json({ message: error.message });
        } else {
            return res.status(500).json({ message: 'An unexpected error occurred' });
        }
```

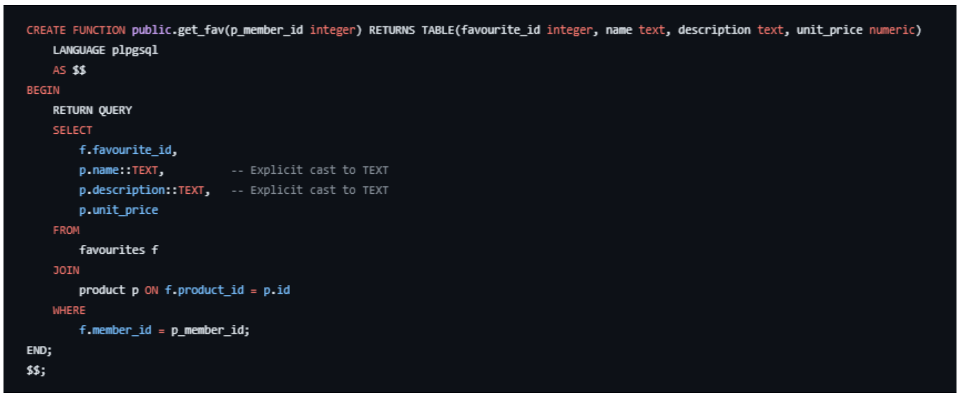
3	Update Entity	• *What is the database stored procedure to create new entities?* • *What are the backend functions (eg model, controller, route) & the front-end changes? (eg html, js)?* • *Any screenshots of how you test for errors and success?* Some of the stored procedure to update table are update_review whose controller is handleUpdateReview which updates the reviews of the user reviews This is done through controller(handleUpdateReview) and model (updateReview) which is executed by an update button in the review retrieve all page that is in every review Which check if the review id is a number and not undefined and then allows for another check if the review text and rating the same as before if it is it shows an alert to show the user that the review is the same  stored procedure link	3/5

```
module.exports.handleUpdateReview = async function (req, res) {
    const member_id = res.locals.member_id;
    const review_id = req.params.reviewId;
    const { rating, reviewText } = req.body;

    // Synchronous validation
    if (rating === undefined || reviewText === undefined) {
        return res.status(400).json({ message: 'All fields are required.' });
    } else if (isNaN(rating) || rating < 1 || rating > 5) {
        return res.status(400).json({ message: 'Rating must be numeric and between 1 and 5.' });
    }

    try {
        // Asynchronous database operation
        await reviewsModel.updateReview(review_id, member_id, reviewText, rating);
        return res.status(200).json({ message: 'Review updated successfully.' });
    } catch (error) {
        console.error('Error:', error.message);

        if (error.message.includes("No review found with the given review_id and member_id")) {
            return res.status(400).json({ message: 'No review found with the given review_id and member_id' });
        }else if (error.message.includes("The new review text and rating are the same as the old values. The review cannot be updated.")) {
            return res.status(400).json({ message: 'The new review text and rating are the same as the old values. The review cannot be updated.' });
        } else if (error.message === 'All fields are required.' || error.message === 'Rating must be numeric and between 1 and 5.') {
            return res.status(400).json({ message: error.message });
        } else {
            return res.status(500).json({ message: 'An unexpected error occurred' });
        }
    }
};
```

4	Delete Entity	• *What is the database stored procedure to create new entities?*
		• *What are the backend functions (eg model, controller, route) & the front-end changes? (eg html, js)? • Any screenshots of how you test for errors and success?*
		
		Some of stored procedure to delete is delete_review which takes in review id and delete it if it passes the check of if it is a number or the text box is filled
		The controller called deletereview which help check if the review id in the input box is empty or not a number if it is not if the review id pass all of the check it deletes the rwo which the review id belongs to
		
		`module.exports.handleUpdateReview = async function (req, res) {`
		`    const member_id = res.locals.member_id;`
		`    const review_id = req.params.reviewId;`
		`    const { rating, reviewText } = req.body;`
		
		`    // Synchronous validation`
		`    if (rating === undefined || reviewText === undefined) {`
		`        return res.status(400).json({ message: 'All fields are required.' });`
		`    } else if (isNaN(rating) || rating < 1 || rating > 5) {`
		`        return res.status(400).json({ message: 'Rating must be numeric and between 1 and 5.' });`
		`    }`
		
		`    try {`
		`        // Asynchronous database operation`
		`        await reviewsModel.updateReview(review_id, member_id, reviewText, rating);`
		`        return res.status(200).json({ message: 'Review updated successfully.' });`
		`    } catch (error) {`
		`        console.error('Error:', error.message);`
		
		`        if (error.message.includes("No review found with the given review_id and member_id")) {`
		`            return res.status(400).json({ message: 'No review found with the given review_id and member_id' });`
		`        }else if (error.message.includes("The new review text and rating are the same as the old values. The review cannot be updated.")) {`
		`            return res.status(400).json({ message: 'The new review text and rating are the same as the old values. The review cannot be updated.' });`
		`        } else if (error.message === 'All fields are required.' || error.message === 'Rating must be numeric and between 1 and 5.') {`
		`            return res.status(400).json({ message: error.message });`
		`        } else {`
		`            return res.status(500).json({ message: 'An unexpected error occurred' });`
		`        }`
		`    }`
		`};`
		link
		stored procedure
			4/5

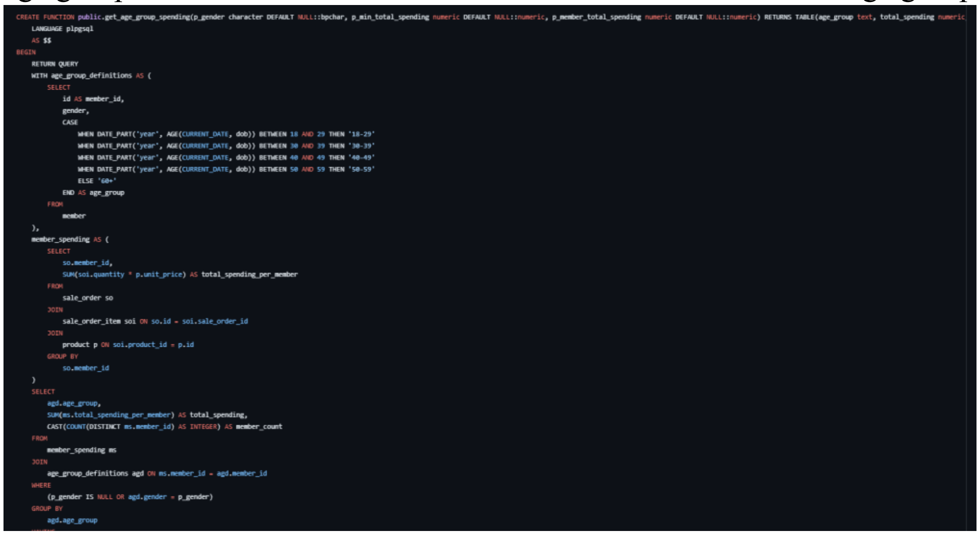
5	Retrieve Entity	• *What is the database stored procedure to create new entities?*
		• *What are the backend functions (eg model, controller, route) & the front-end changes? (eg html, js)? • Any screenshots of how you test for errors and success?*
		Some of stored procedure to retrieve entity are get_all_reviews which takes in the member id from local storage and show all of the reviews who belong to the member the member id is also taken from the controller handleShowReview
		
		`module.exports.handleShowReview = async function (req, res) {`
		`    const member_id = res.locals.member_id;`
		`    try {`
		`        const reviews = await reviewsModel.getReviewByMemberId(member_id);`
		`        res.status(200).json(reviews); // Send the fetched reviews as JSON response`
		`    } catch (err) {`
		`        console.error('Error fetching reviews:', err.message);`
		`        res.status(500).json({ error: 'Failed to fetch reviews' }); // Send an error response if fetching fails`
		`    }`
		`}`
		stored procedure
		link
			3/5

6	Query Quality	• *How well do your queries use operations like join, filter, group and aggregate collectively to enable endto-end features?* • *Any screenshots of how you test for errors and different conditions?* I would use join,filter in my stored procedure so as to normalize my favourite table which stores only the product id. The stored procedure to show all the favourite which joins the favourite and product table and join together by product_id and show the name, description and unit price	4/5



```
CREATE FUNCTION public.get_fav(p_member_id integer) RETURNS TABLE(favourite_id integer, name text, description text, unit_price numeric)
    LANGUAGE plpgsql
    AS $$
BEGIN
    RETURN QUERY
    SELECT
        f.favourite_id,
        p.name::TEXT,          -- Explicit cast to TEXT
        p.description::TEXT,   -- Explicit cast to TEXT
        p.unit_price
    FROM
        favourites f
    JOIN
        product p ON f.product_id = p.id
    WHERE
        f.member_id = p_member_id;
END;
$$;
```

I would use group operation to group the data for clv by creating a case which separate the age group which allows for the table to show the data base on the age group



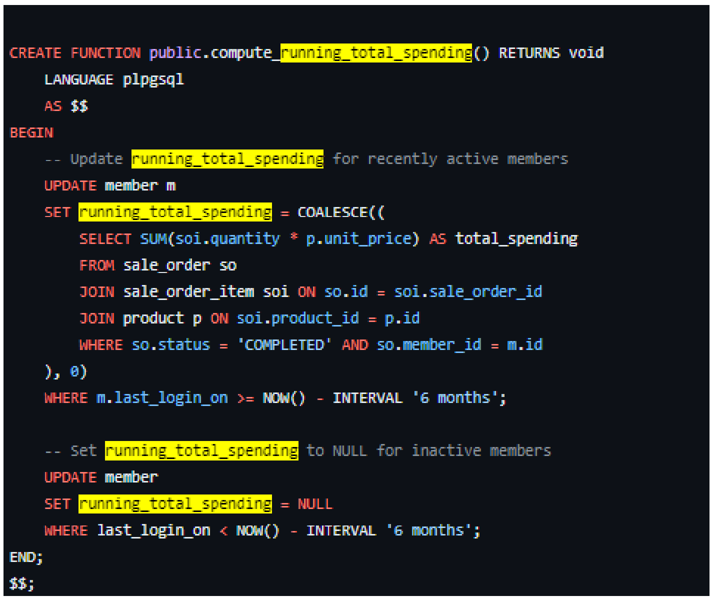
```
CREATE FUNCTION public.get_age_group_spending(p_gender character DEFAULT NULL::bpchar, p_min_total_spending numeric DEFAULT NULL::numeric, p_member_total_spending numeric DEFAULT NULL::numeric) RETURNS TABLE(age_group text, total_spending numeric
    LANGUAGE plpgsql
    AS $$
BEGIN
    RETURN QUERY
    WITH age_group_definitions AS (
        SELECT
            id AS member_id,
            gender,
            CASE
                WHEN DATE_PART('year', AGE(CURRENT_DATE, dob)) BETWEEN 18 AND 29 THEN '18-29'
                WHEN DATE_PART('year', AGE(CURRENT_DATE, dob)) BETWEEN 30 AND 39 THEN '30-39'
                WHEN DATE_PART('year', AGE(CURRENT_DATE, dob)) BETWEEN 40 AND 49 THEN '40-49'
                WHEN DATE_PART('year', AGE(CURRENT_DATE, dob)) BETWEEN 50 AND 59 THEN '50-59'
                ELSE '60+'
            END AS age_group
        FROM
            member
    ),
    member_spending AS (
        SELECT
            so.member_id,
            SUM(soi.quantity * p.unit_price) AS total_spending_per_member
        FROM
            sale_order so
        JOIN
            sale_order_item soi ON so.id = soi.sale_order_id
        JOIN
            product p ON soi.product_id = p.id
        GROUP BY
            so.member_id
    )
    SELECT
        agd.age_group,
        SUM(ms.total_spending_per_member) AS total_spending,
        CAST(COUNT(DISTINCT ms.member_id) AS INTEGER) AS member_count
    FROM
        member_spending ms
    JOIN
        age_group_definitions agd ON ms.member_id = agd.member_id
    WHERE
        (p_gender IS NULL OR agd.gender = p_gender)
    GROUP BY
        agd.age_group
```

stored procedure

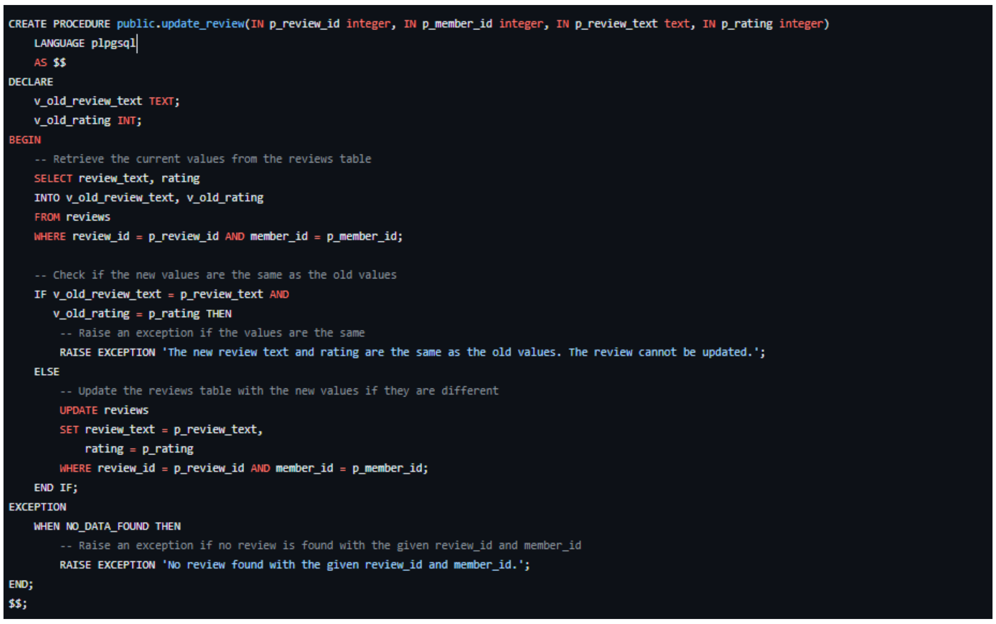
7 | User defined Functions Program Logic | 4/5

- *How well do your user defined function(s) use program logic and control structures to handle program complexity and fulfil requirements?*
- *Any screenshots of how you test for errors and success?*

I would use user defined function like compute_running_total_spending() to calculate their total spending on the website and set the total to null if they last login more than 6 months ago



```
CREATE FUNCTION public.compute_running_total_spending() RETURNS void
    LANGUAGE plpgsql
    AS $$
BEGIN
    -- Update running_total_spending for recently active members
    UPDATE member m
    SET running_total_spending = COALESCE((
        SELECT SUM(soi.quantity * p.unit_price) AS total_spending
        FROM sale_order so
        JOIN sale_order_item soi ON so.id = soi.sale_order_id
        JOIN product p ON soi.product_id = p.id
        WHERE so.status = 'COMPLETED' AND so.member_id = m.id
    ), 0)
    WHERE m.last_login_on >= NOW() - INTERVAL '6 months';

    -- Set running_total_spending to NULL for inactive members
    UPDATE member
    SET running_total_spending = NULL
    WHERE last_login_on < NOW() - INTERVAL '6 months';
END;
$$;
```

		stored procedure	
8	Stored procedures Program Logic	*• How well do your stored procedure(s) use program logic and control structures to program complexity and fulfil requirements? • Any screenshots of how you test for errors and success?* I would use stored procedure like update_review which takes in the rating reviewtext input to update the review where the member_id and review_id belongs to so as to update the correct review and reject a update if the rating reviewtext input are the same as it were already while making sure the input are of the correct data type  link	4/5

```
CREATE PROCEDURE public.update_review(IN p_review_id integer, IN p_member_id integer, IN p_review_text text, IN p_rating integer)
    LANGUAGE plpgsql
    AS $$
DECLARE
    v_old_review_text TEXT;
    v_old_rating INT;
BEGIN
    -- Retrieve the current values from the reviews table
    SELECT review_text, rating
    INTO v_old_review_text, v_old_rating
    FROM reviews
    WHERE review_id = p_review_id AND member_id = p_member_id;

    -- Check if the new values are the same as the old values
    IF v_old_review_text = p_review_text AND
       v_old_rating = p_rating THEN
        -- Raise an exception if the values are the same
        RAISE EXCEPTION 'The new review text and rating are the same as the old values. The review cannot be updated.';
    ELSE
        -- Update the reviews table with the new values if they are different
        UPDATE reviews
        SET review_text = p_review_text,
            rating = p_rating
        WHERE review_id = p_review_id AND member_id = p_member_id;
    END IF;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        -- Raise an exception if no review is found with the given review_id and member_id
        RAISE EXCEPTION 'No review found with the given review_id and member_id.';
END;
$$;
```

9	Report Quality	*Based on the quality of documentation for above criteria. No inputs required here.*	?/5
10	Demonstration & Interview	*Based on the assessment during demonstration & interview. No inputs required here.*	-