CA2 Individual Report

Name	Lucas Dong
Student Id	P2429535
Class	DIT/FT/2B/21
Github Repository URL	https://github.com/soc-DBS/dbs-assignment-lutanicdadude.git
Github Account ID	lutanicdadude

For each criterion, provide links to pull requests/commits/files that demonstrate the completion of the requirement. Replace each "**?**" with your Self Rating.

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

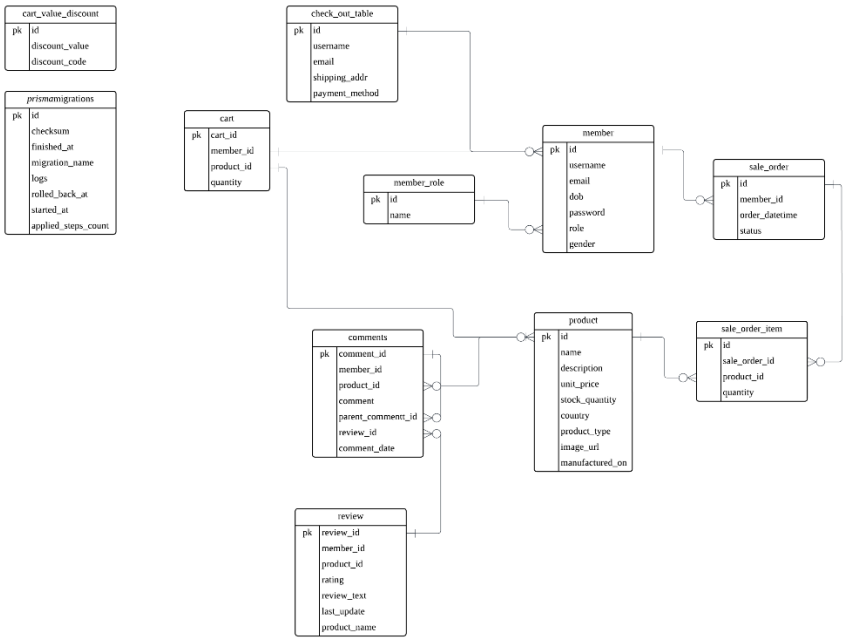
0 Have little or **no** understanding. and did not attempt the requirement.
1 Have **limited** understanding to demonstrate competency for the criterion.
2 Have **basic** understanding and only able to replicate examples from tutorials/practicals.
3 Have **adequate** understanding and can extend from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

Important

a) You are required to provide for each criterion:
- **Documentation** and description of the work done.
- **One to three** of your best implementations with URL **link** to respective repository files/commits/pull requests.
- 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 & ORM Modeling	The new tables are cart_value_discount, check_out_table and _prisma_migrations. https://github.com/soc-DBS/dbs-assignment-lutanicdadude/blob/master/prisma/schema.prisma I assumed that the user would enter their email, address and their payment type during checkout. I also assumed that there would be a promotional code section where the user can get more discount.	5/5

			
2	Cart Management feature Implementation	For Carts controller I have createCartItems, updateCartItems, retrieveCartItems, deleteCartItems, getCartSummary, checkProduct_id and check_cart_table. As for models, I have creating_cart, getting_cart, retrieving_cart_items , checking_product_id, updating_cart_items, deleting_cart_table and checking_quantity. For routes I have post, get, put and delete for /add_cart_item, /summary and a few more /. As for the front end, I have made it so that the user is able to view all the items in the cart and update or delete any of the items with ease. The respective data for each item is also shown, together with the overall number of items and total prices. For ORM functions I used findUnique, findFirst, findMany, create, update and delete. - Controller - https://github.com/soc-DBS/dbs-assignment-lutanicdadude/blob/master/controllers/ORMcartsController.js - Model - https://github.com/soc-DBS/dbs-assignment-lutanicdadude/blob/master/models/ORMcarts.js - Route - https://github.com/soc-DBS/dbs-assignment-lutanicdadude/blob/master/routes/carts.js - Front-end - https://github.com/soc-DBS/dbs-assignment-lutanicdadude/tree/master/public/cart - Prisma - https://github.com/soc-DBS/dbs-assignment-lutanicdadude/blob/master/prisma/migrations/20250806140634_adding_cart/migration.sql - Test case for creating item in cart and the result	5/5

```
28  -- Test 1: Valid Insert
29  CALL create_cart(1, 1, 2);
30
```

Data Output | Messages | Notifications

```
CALL

Query returned successfully in 89 msec.
```

-

```
31  -- Test 2: Invalid Product (should throw error)
32  CALL create_cart(1, 9999, 2);
33
```

Data Output | Messages | Notifications

```
ERROR:  Invalid product_id: Product does not exist.
CONTEXT:  PL/pgSQL function create_cart(integer,integer,integer) line 6 at RAISE

SQL state: P0001
```

-

```
34  -- Test 3: Quantity Zero (should throw error)
35  CALL create_cart(1, 1, 0);
```

Data Output | Messages | Notifications

```
ERROR:  Quantity must be more than 1.
CONTEXT:  PL/pgSQL function create_cart(integer,integer,integer) line 11 at RAISE

SQL state: P0001
```

-

```
37  -- Test 4: Quantity Negative (should throw error)
38  CALL create_cart(1, 1, -5);
39
```

Data Output | Messages | Notifications

```
ERROR:  Quantity must be more than 1.
CONTEXT:  PL/pgSQL function create_cart(integer,integer,integer) line 11 at RAISE

SQL state: P0001
```

-

```
40  -- Test 5: Boundary Quantity = 1 (should pass if >=1 is allowed)
41  CALL create_cart(1, 1, 1);
42
```

Data Output | Messages | Notifications

```
CALL

Query returned successfully in 42 msec.
```

-

- Test case for deleting item in cart

-
```
-- Test: Delete existing cart item
CALL delete_cart_item(
    1,
    1
);
```

Data Output | Messages | Notifications

```
CALL

Query returned successfully in 47 msec.
```

-
```
-- Test: Delete non-existing cart item
CALL delete_cart_item(
    9999,
    8888
);
```

Data Output | Messages | Notifications

```
ERROR:  Cart item not found.
CONTEXT:  PL/pgSQL function delete_cart_item(integer,integer) line 10 at RAISE

SQL state: P0001
```

- Test case for get cart detail by member id

-
```
-- test case
SELECT * FROM get_cart_details_by_member(1);
```

Data Output | Messages | Notifications

member_id integer	product_id integer	quantity integer	description text	country character varying	unit_price numeric	name character varying
1	1	2	Soft and cuddly teddy bear, perfect for kids and collectors	USA	25.50	Plush Teddy Bear

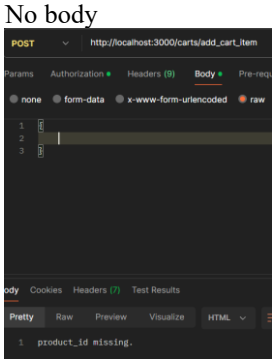
-
```
SELECT * FROM get_cart_details_by_member(9999);
```

Data Output | Messages | Notifications

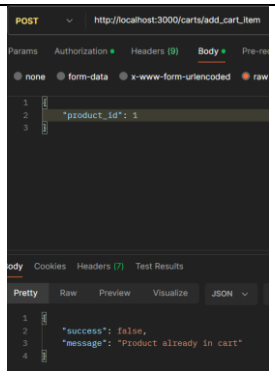
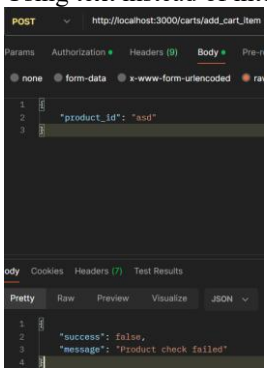
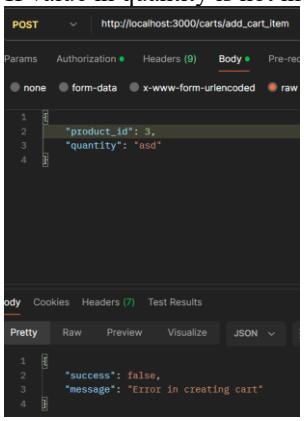
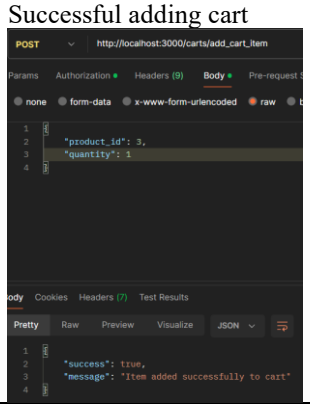
member_id integer	product_id integer	quantity integer	description text	country character varying	unit_price numeric	name character varying

- PostMan vailidations

- No body

- 

- Product already in cart

		-  - Using text instead of integer -  - If value in quantity is not integer -  - Successful adding cart -  	
3	Checkout feature Implement ation	In the controller I have checkPromoCode, checkCartForOrder and submitCheckOut. In models I have checkingPromoCode, checkingCartForOrder and submitCheckOut. In route I have 2 posts for /checkPromo and /submitCart. As for the front end, I have taken inspiration from other shopping websites like shopee. The check out page is intuitive where the user can input their credentials and even add in additional promotional code to get discounts. The order summary is dynamic where it will change according to how many items	5/5

there are in the cart, with the total prices and if there are additional discounts. As for ORM functions, I have findMany. For submitCheckOut I tried using ORM functions but I needed to use the stored procedure so I did not use it.

- Controller
- https://github.com/soc-DBS/dbs-assignment-lutanicdadude/blob/master/controllers/checkOutController.js
- Model
- https://github.com/soc-DBS/dbs-assignment-lutanicdadude/blob/master/models/checkOut.js
- Route
- https://github.com/soc-DBS/dbs-assignment-lutanicdadude/blob/master/routes/checkOut.js
- Front end
- https://github.com/soc-DBS/dbs-assignment-lutanicdadude/tree/master/public/checkout
- Prisma
- https://github.com/soc-DBS/dbs-assignment-lutanicdadude/blob/master/prisma/migrations/20250806141216_adding_check_out_table/migration.sql
-
- Validate whether promo code is real

```
CREATE OR REPLACE FUNCTION check_promo_discount(p_code character varying(20))
RETURNS BOOLEAN AS $$
DECLARE
    exists boolean;
BEGIN
    SELECT EXISTS (
        SELECT 1 FROM cart_value_discount WHERE discount_code = p_code
    ) INTO exists;

    RETURN exists;
END;
$$ LANGUAGE plpgsql;
```

-
- If Promotional code is wrong

```
14  SELECT * FROM check_promo_discount('whee');
```

Data Output | Messages | Notifications

	id integer	discount_value double precision	discount_code character varying (20)

-
- If Promotional code is correct

```
14  SELECT * FROM check_promo_discount('SUMMER25');
```

Data Output | Messages | Notifications

	id integer	discount_value double precision	discount_code character varying (20)
1	3	0.25	SUMMER25

-
- Check to see if there is anything in cart

```
CREATE OR REPLACE FUNCTION check_cart_for_order(p_member_id INT)
RETURNS VOID AS $$
BEGIN
    -- Step 2: Check cart if there is order
    IF NOT EXISTS (
        SELECT 1 FROM cart WHERE member_id = p_member_id
    ) THEN
        RAISE EXCEPTION 'There is no order in cart.';
    END IF;
END;
$$ LANGUAGE plpgsql;
```

-
- *Error if you input non-integer*

```
17  SELECT * FROM check_cart_for_order('asd');
```

Data Output | Messages | Notifications

```
ERROR:  invalid input syntax for type integer: "asd"
LINE 1: SELECT * FROM check_cart_for_order('asd');
                                           ^
```

- *Error if you do not input anything*

```
17  SELECT * FROM check_cart_for_order();
```

Data Output | Messages | Notifications

```
ERROR:  function check_cart_for_order() does not exist
LINE 17: SELECT * FROM check_cart_for_order();
                       ^
HINT:  No function matches the given name and argument
```

- *Error if you input float*

```
17  SELECT * FROM check_cart_for_order(1.1);
```

Data Output | Messages | Notifications

```
ERROR:  function check_cart_for_order(numeric) does not exist
LINE 17: SELECT * FROM check_cart_for_order(1.1);
                       ^
```

- *Successful message*

```
17  SELECT * FROM check_cart_for_order(1);
```

Data Output | Messages | Notifications

	p_cart_id integer	p_product_id integer	p_quantity integer
1	32	1	2
2	33	2	1
3	34	3	1

- Checks to ensure that the user is correct

```
-- Step 1: Member validation
IF NOT EXISTS (
    SELECT 1 FROM member WHERE id = p_member_id
) THEN
    RAISE EXCEPTION 'Member % does not exist.', p_name;
END IF;
```

```
83  -- Test Case 2: Non-existent member
84  CALL place_orders(
85      999,
86      'ghost_user',
87      'ghost@example.com',
88      '404 Nowhere Street',
89      'PayPal'
90  );
```

Data Output | Messages | Notifications

```
ERROR:  column "p_name" does not exist
```

- Checks stock to ensure that there is enough stock

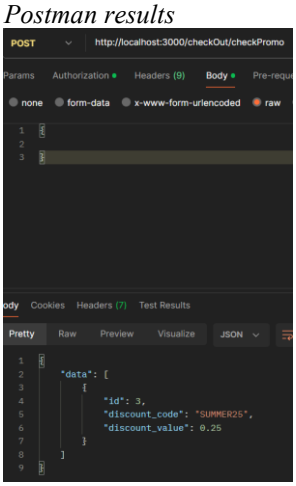
```
-- Step 3a: Check stock
IF (SELECT stock_quantity FROM product WHERE id = cart_item.product_id) >= cart_item.quantity THEN
```

- If there is not enough stock and continue of something goes wrong

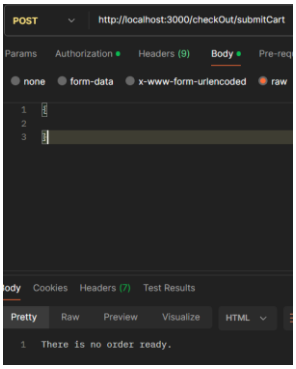
```
    -- Step 4: Not enough stock
    RAISE NOTICE 'Not enough stock for product ID %', cart_item.product_id;
  IF;

  ON WHEN OTHERS THEN
  Step 5: Error handling - skip and continue
  SQLERRM = return a text message describing what went wrong inside the BEGIN and END block
  SE NOTICE 'Error processing cart item ID %: %', cart_item.cart_id, SQLERRM;
```

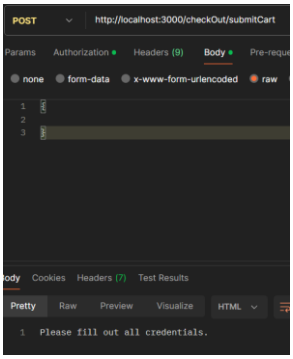
- *Postman results*



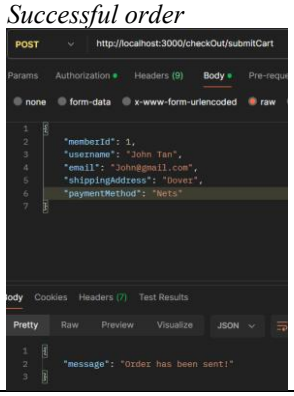
- *No Order in cart*

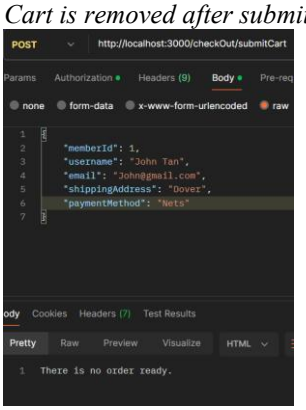


- *Order in cart but no credentials entered*



- *Successful order*



		- *Cart is removed after submitting order* -  - My implementations are flexible as there are multiple error validations at every step, where the code would check whether the data is correct before continuing to the next step. You can easily add on columns to the form and order summary so that it would show more data. This can be done so using the web component that I have made.	
4	Transaction Management	- *DROP PROCEDURE IF EXISTS place_orders;* - - *CREATE OR REPLACE PROCEDURE place_orders(* - *p_member_id INT,* - *p_username character varying (50),* - *p_email character varying (50),* - *p_shipping_address character varying (255),* - *p_payment_method character varying (50)* - *)* - *LANGUAGE plpgsql* - *AS $$* - *DECLARE* - *cart_item RECORD;* - *new_order_id INT;* - *BEGIN* - *-- Step 1: Member validation* - *IF NOT EXISTS (* - *SELECT 1 FROM member WHERE id = p_member_id* - *) THEN* - *RAISE EXCEPTION 'Member % does not exist.', p_name;* - *END IF;* - - *-- Step 2: Create new sale order* - *INSERT INTO sale_order (* - *member_id,* - *order_datetime,* - *status,* - *shipping_address,* - *payment_method* - *)* - *VALUES (* - *p_member_id,* - *CURRENT_TIMESTAMP,* - *'PACKING',* - *p_shipping_address,* - *p_payment_method* - *)* - *RETURNING id INTO new_order_id;* - - *-- Step 3: Loop through each cart item for the member* - *FOR cart_item IN*	4/5

- *SELECT * FROM cart WHERE member_id = p_member_id*
- *LOOP*
- *BEGIN*
- *-- Step 3a: Check stock*
- *IF (SELECT stock_quantity FROM product WHERE id = cart_item.product_id) >= cart_item.quantity THEN*
-
- *-- Step 3b: Deduct stock*
- *UPDATE product*
- *SET stock_quantity = stock_quantity - cart_item.quantity*
- *WHERE id = cart_item.product_id;*
-
- *-- Step 3c: Insert into sale_order_item*
- *INSERT INTO sale_order_item (sale_order_id, product_id, quantity)*
- *VALUES (new_order_id, cart_item.product_id, cart_item.quantity);*
-
- *-- Step 3d: Remove from cart*
- *DELETE FROM cart WHERE cart_id = cart_item.cart_id;*
-
- *ELSE*
- *-- Step 4: Not enough stock*
- *RAISE NOTICE 'Not enough stock for product ID %', cart_item.product_id;*
- *END IF;*
-
- *EXCEPTION WHEN OTHERS THEN*
- *-- Step 5: Error handling – skip and continue*
- *-- SQLERRM = return a text message describing what went wrong inside the BEGIN and END block*
- *RAISE NOTICE 'Error processing cart item ID %: %', cart_item.cart_id, SQLERRM;*
- *END;*
- *END LOOP;*
- *END;*
- *$$;*
- Ensures that the member is correct

```
-- Step 1: Member validation
IF NOT EXISTS (
    SELECT 1 FROM member WHERE id = p_member_id
) THEN
    RAISE EXCEPTION 'Member % does not exist.', p_name;
END IF;
```

-
- *Check stock and deduct accordingly*

```
-- Step 3a: Check stock
IF (SELECT stock_quantity FROM product WHERE id = cart_item.product_id) >= cart_item.quantity THEN

    -- Step 3b: Deduct stock
    UPDATE product
    SET stock_quantity = stock_quantity - cart_item.quantity
    WHERE id = cart_item.product_id;
```

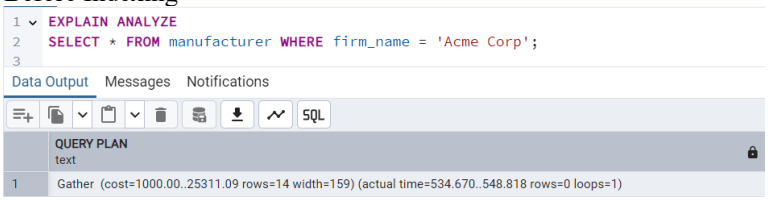
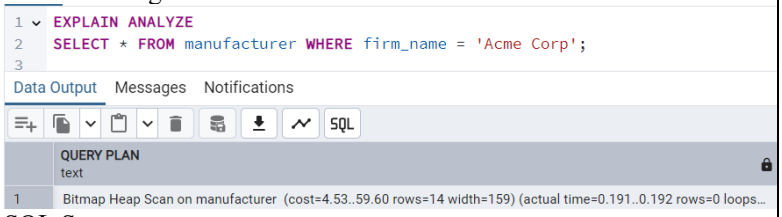
-
- Error message for not enough stock

```
-- Step 4: Not enough stock
RAISE NOTICE 'Not enough stock for product ID %', cart_item.product_id;
```

- Skip if there is an error with item

```
EXCEPTION WHEN OTHERS THEN
    -- Step 5: Error handling - skip and continue
    -- SQLERRM = return a text message describing what went wrong inside the BEGIN and END block
    RAISE NOTICE 'Error processing cart item ID %: %', cart_item.cart_id, SQLERRM;
END;
```

-
- Postman validation is the same as checkout.

		I have not use d an ORM function as I wanted to use a stored and it was safer and more efficient to do it that way instead of creating create and delete ORM functions. For Model, I have checking_promo_code, checking_cart_for_order and submit_check_out. For Controller I have checkPromoCode, checkCartForOrder and submitCheckOut.for Route I have 2 post for /checkPromo and /submitCart. As for the front-end, I only have a popup below the checkout to tell the user whether the order has been sent or if there is an error. - Controller - https://github.com/soc-DBS/dbs-assignment-lutanicdadude/blob/master/controllers/checkOutController.js - Model - https://github.com/soc-DBS/dbs-assignment-lutanicdadude/blob/master/models/checkOut.js - Route - https://github.com/soc-DBS/dbs-assignment-lutanicdadude/blob/master/routes/checkOut.js - Front-end - https://github.com/soc-DBS/dbs-assignment-lutanicdadude/tree/master/public/checkout -	
5	Indexing	- SELECT * FROM manufacturer WHERE firm_name = '_firm name_'; - B-Tree Index - Before Indexing  - - After Indexing  - - SQL Statement - CREATE INDEX idx_firm_name ON public.manufacturer USING btree (firm_name) - Why its faster? - The query searches for a specific firm_name without scanning through all 818K rows. With the B-Tree index, it can directly jump to the matching rows. - EXPLAIN ANALYSE - SELECT id, firm_name, origin, description, employee_count, is_operational, product_category FROM manufacturer WHERE origin = '_USA_' AND is_operational = _true_ ORDER BY _employee_count DESC_; - B-Tree Index - Before Indexing	4/5

```
EXPLAIN ANALYSE
SELECT id, firm_name, origin, description, employee_count, is_operational, product_category
FROM manufacturer
WHERE origin = 'USA'
AND is_operational = true
ORDER BY
employee_count DESC;
```

Data Output | Messages | Notifications

	QUERY PLAN text
1	Sort (cost=29814.64..29977.88 rows=65295 width=87) (actual time=452.010..473.554 rows=65464 loops=1)

- After Indexing

```
EXPLAIN ANALYSE
SELECT id, firm_name, origin, description, employee_count, is_operational, product_category
FROM manufacturer
WHERE origin = 'USA'
AND is_operational = true
ORDER BY
employee_count DESC;
```

Data Output | Messages | Notifications

	QUERY PLAN text
1	Index Only Scan using idx_usa_operational_empcount_covering on manufacturer (cost=0.42..5264.68 rows=65295 width=87) (actual time=1.149..43.946 rows=65464 loops=...

- SQL Statement
- CREATE INDEX idx_usa_operational_empcount_covering
- ON public.manufacturer USING btree (origin, is_operational, employee_count DESC, id, firm_name, description, product_category)
- Why its faster?
- The SQL statement filters and sorts efficiently as the first 2 columns allow the data base to quickly locate the relevant rows and the employee_count is already stored in sorted order in the index.

- SELECT * FROM manufacturer ORDER BY founded_since _DESC_ LIMIT 10;
- B-Tree Index;
- Before Indexing

```
EXPLAIN ANALYZE
SELECT * FROM manufacturer ORDER BY founded_since DESC LIMIT 10;
```

Data Output | Messages | Notifications

	QUERY PLAN text
1	Limit (cost=31824.86..31826.02 rows=10 width=159) (actual time=323.170..339.676 rows=10 loops=1)

- After Indexing

```
EXPLAIN ANALYZE
SELECT * FROM manufacturer ORDER BY founded_since DESC LIMIT 10;
```

Data Output | Messages | Notifications

	QUERY PLAN text
1	Limit (cost=0.42..1.59 rows=10 width=159) (actual time=0.387..0.922 rows=10 loops=1)

- \
- SQL Statement
- CREATE INDEX idx_founded_since ON public.manufacturer USING btree (founded_since DESC)
- Why its faster?
- The Index stores rows already stored by founded_since DESC. The database can retrieve the first 10 rows directly from the index

- SELECT id, firm_name, origin, employee_count, founded_since, product_category FROM manufacturer WHERE is_operational = true AND origin IN ('USA', 'Germany', 'Japan') AND employee_count BETWEEN 100 AND 1000 AND founded_since >= '2000-01-01' ORDER BY employee_count DESC, founded_since ASC;
- Composite B-Tree Index
- Before Index

Query | Query History

```
1  EXPLAIN ANALYSE
2  SELECT id, firm_name, origin, employee_count, founded_since, product_category
3  FROM manufacturer
4  WHERE is_operational = true
5    AND origin IN ('USA', 'Germany', 'Japan')
6    AND employee_count BETWEEN 100 AND 1000
7    AND founded_since >= '2000-01-01'
8  ORDER BY employee_count DESC, founded_since ASC;
```

Data Output | Messages | Notifications

	QUERY PLAN text
1	Sort (cost=21219.83..21243.07 rows=9299 width=56) (actual time=75.108..76.514 rows=9465 loops=1)

- After Index

```
1  EXPLAIN ANALYSE
2  SELECT id, firm_name, origin, employee_count, founded_since, product_category
3  FROM manufacturer
4  WHERE is_operational = true
5    AND origin IN ('USA', 'Germany', 'Japan')
6    AND employee_count BETWEEN 100 AND 1000
7    AND founded_since >= '2000-01-01'
8  ORDER BY employee_count DESC, founded_since ASC;
```

Data Output | Messages | Notifications

	QUERY PLAN text
1	Sort (cost=18524.47..18547.71 rows=9299 width=56) (actual time=43.345..44.917 rows=9465 loops=1)

- SQL Statement
- CREATE INDEX idx_operational_origin_emp_founded ON manufacturer(is_operational, origin, employee_count, founded_since DESC);
- Why its faster?
- The statement filters efficiently where the is_operational and origin allows the database to quickly find the relevant rows. The data is also stored in its sorted order so that retrieving the data would be faster.

- SELECT id, firm_name, origin, founded_since, employee_count FROM manufacturer WHERE origin = '_USA_' AND is_operational = _true_
- ORDER BY founded_since _DESC_LIMIT 20;
- B-Tree Index
- Before Indexing

Query | Query History

```
1  EXPLAIN ANALYSE
2  SELECT origin AS country, SUM(employee_count) AS total_employees
3  FROM manufacturer
4  GROUP BY origin
5  ORDER BY total_employees DESC;
```

Data Output | Messages | Notifications

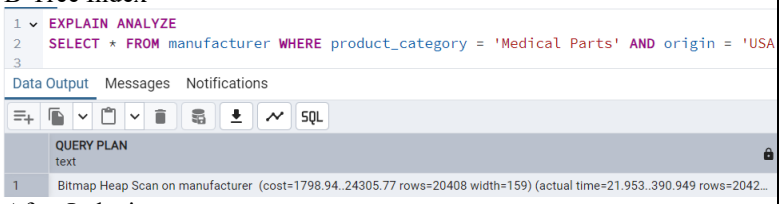
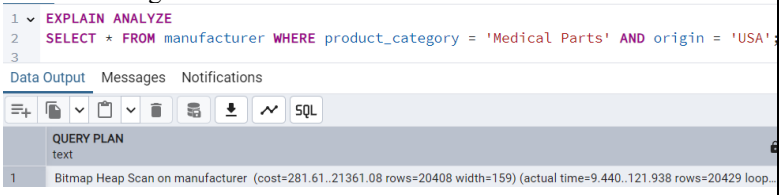
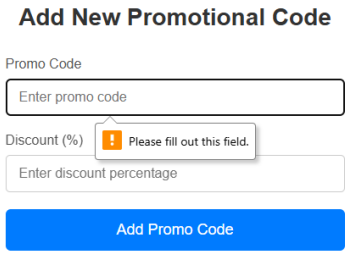
	QUERY PLAN text
1	Sort (cost=26163.48..26163.49 rows=5 width=15) (actual time=392.663..408.793 rows=5 loops=1)

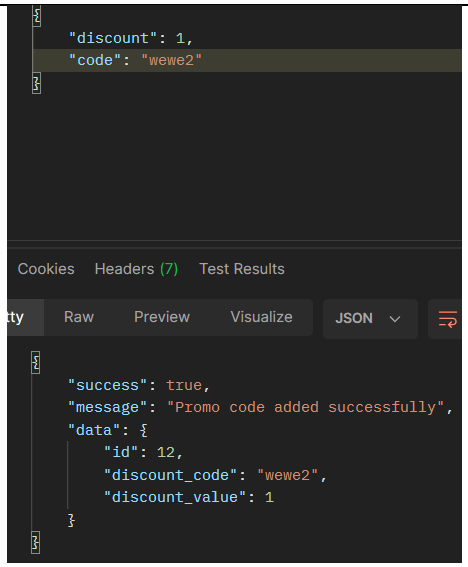
- After Indexing

```
1  EXPLAIN ANALYSE
2  SELECT origin AS country, SUM(employee_count) AS total_employees
3  FROM manufacturer
4  GROUP BY origin
5  ORDER BY total_employees DESC;
```

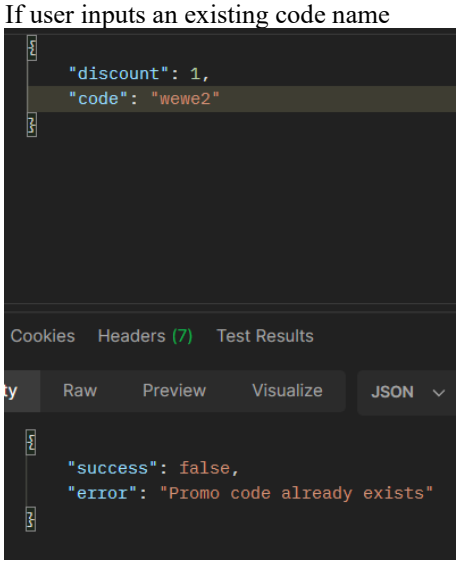
Data Output | Messages | Notifications

	QUERY PLAN text
1	Sort (cost=13165.80..13165.82 rows=5 width=15) (actual time=311.525..323.788 rows=5 loops=1)

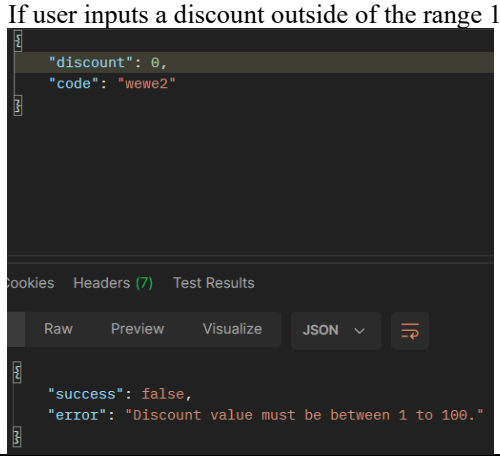
		- SQL Statement - CREATE INDEX idx_origin_operational_founded - ON public.manufacturer USING btree (origin, is_operational, founded_since DESC) - Why its faster? - The index creates a smaller row subset and the sorting avoids extra sorting, allowing the database to retrieve the top 20 results easily. - SELECT * FROM manufacturer WHERE product_category = '_Medical Parts_' AND origin = '_USA_'; - B-Tree Index  - After Indexing  - SQL Statement - CREATE INDEX idx_category_origin ON public.manufacturer USING btree (product_category, origin) - Why its faster? - The product_category and origin can quickly locate the matching rows and reduces the number of rows needed to be scanned.	
9	Extra Functionality (Allow Admin to add new promo code)	- I have findFirst and create in the controller - Controller - https://github.com/soc-DBS/dbs-assignment-lutanicdadude/blob/master/controllers/promoController.js - Route - https://github.com/soc-DBS/dbs-assignment-lutanicdadude/blob/master/routes/promo.js - Validations: - If user does not input anything  - - Successful addition of new promo code	



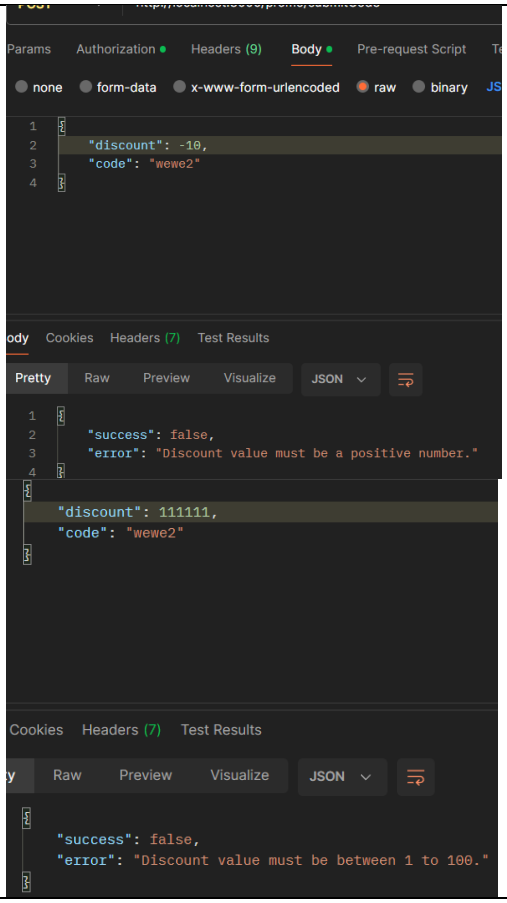
-
- If user inputs an existing code name



-
- If user inputs a discount outside of the range 1 to 100



-

		-  -	
10	Report Quality	*Based on quality of documentation for above criteria.* *No inputs required here.*	4/5
11	Demonstration & Interview	*Based on assessment during demonstration & interview.* *No inputs required here.*	-