CA2 Individual Report

Name	Ong Jin Kai
Student Id	P2429465
Class	DIT/FT/2B/21
Github Repository URL	https://github.com/soc-DBS/dbs-assignment-cutiepatootiekai
Github Account ID	cutiepatootiekai

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:

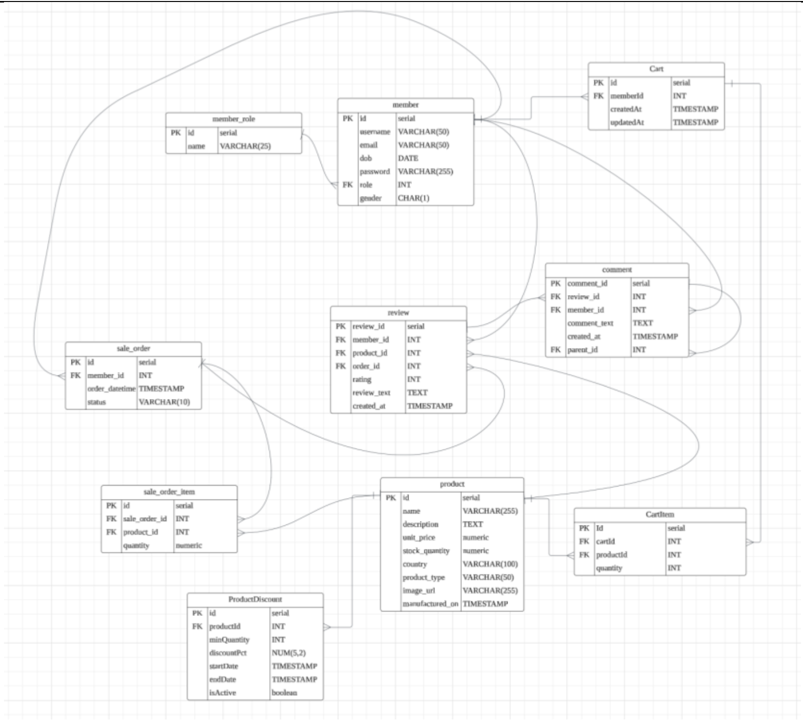
0 Have little or **no** understanding. and did not attempt the requirement.
1 Have **limited** understanding to demonstrate competency for the criterion.
2 Have **basic** understanding and only able to replicate examples from tutorials/practicals.
3 Have **adequate** understanding and can extend from 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.

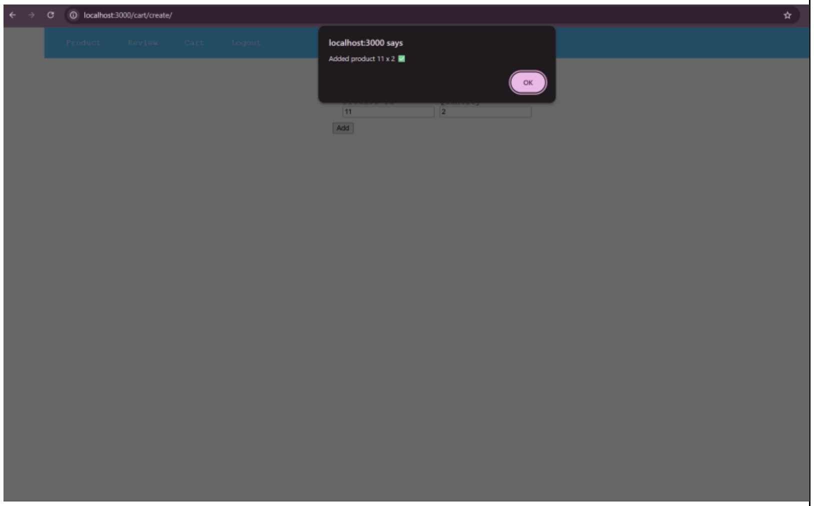
Important

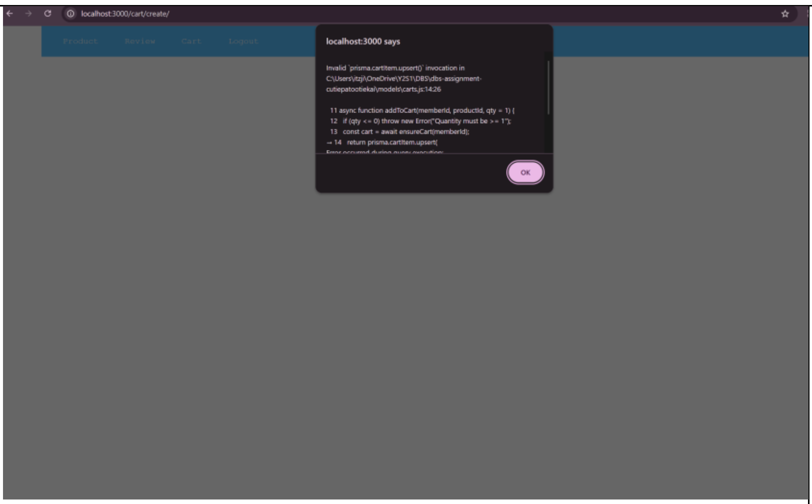
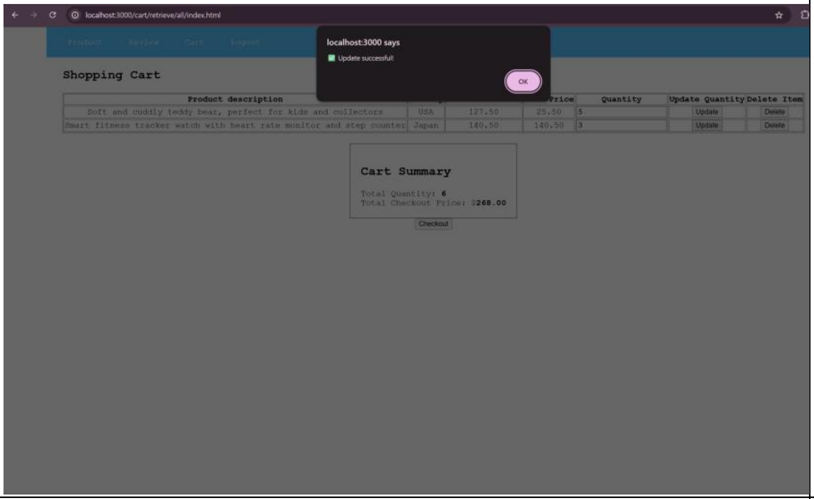
a) You are required to provide for each criterion:
- **Documentation** and description of the work done.
- **One to three** of your best implementations with URL **link** to respective repository files/commits/pull requests.
- You should also provide **screenshots** where relevant.

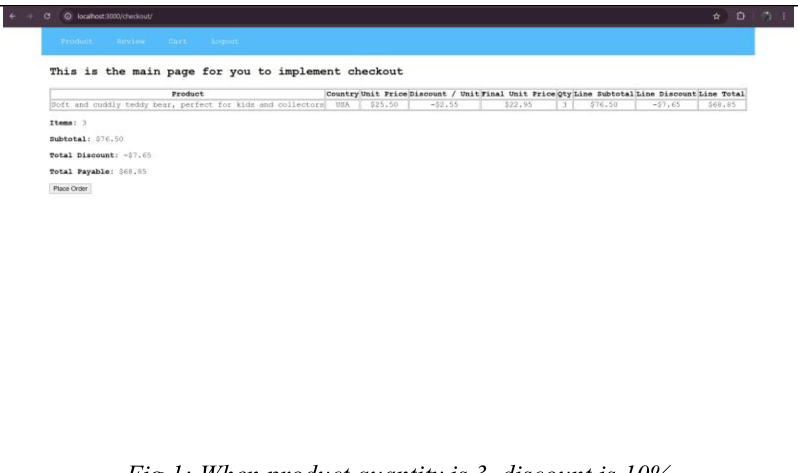
b) You are to ensure the hyperlink in this document works. **Failure to do so will result in a 50% deduction of marks.**

No.	Criterion	Describe What Was Done	Self Rating
1	Database Design & ORM Modeling		3/5

		 member_role PK id serial name VARCHAR(25) member PK id serial username VARCHAR(50) email VARCHAR(50) dob DATE password VARCHAR(255) FK role INT gender CHAR(1) Cart PK id serial FK memberId INT createdAt TIMESTAMP updatedAt TIMESTAMP comment PK comment_id serial FK review_id INT FK member_id INT comment_text TEXT created_at TIMESTAMP FK parent_id INT review PK review_id serial FK member_id INT FK product_id INT FK order_id INT rating INT review_text TEXT created_at TIMESTAMP sale_order PK id serial FK member_id INT order_datetime TIMESTAMP status VARCHAR(10) sale_order_item PK id serial FK sale_order_id INT FK product_id INT quantity numeric product PK id serial name VARCHAR(255) description TEXT unit_price numeric stock_quantity numeric country VARCHAR(100) product_type VARCHAR(50) image_url VARCHAR(255) manufactured_on TIMESTAMP CartItem PK Id serial FK cartId INT FK productId INT quantity INT ProductDiscount PK id serial FK productId INT minQuantity INT discountPct NUM(5,2) startDate TIMESTAMP endDate TIMESTAMP isActive boolean *The new models are Cart, CartItem, ProductDiscount.* *https://github.com/soc-DBS/dbs-assignment-cutiepatootiekai/blob/master/prisma/schema.prisma* ***Stock only reduced at checkout*** • *We don't "reserve" items when they're added to the cart.* • *Whoever checks out first gets the stock; if stock isn't enough, the order fails at checkout.* • *This keeps things simple—no timers or cleanup for holds.* ***One best discount per product line*** • *If multiple discounts could apply, we pick the* ***single*** *one that gives the* ***lowest price****.* • *We* ***don't stack*** *discounts.* ***When a discount is valid*** • *It must be* ***active****,* • *Today must be* ***between start and end dates*** *(no end date = still okay),* • *And you must buy at least the* ***minimum quantity*** *(if any).* • *That's how most promos work in shops, and it's easy to check in SQL.*	
2	Cart Management feature		4/5

	Implementation	**Backend** • createCartItems • updateCartItems • retrieveCartItems • deleteCartItems • getCartSummary • ensureCart • addToCart • updateLine • removeLine • getCart • getSummary **Frontend** • refreshCart • fetchCartItems • fetchCartSummary https://github.com/soc-DBS/dbs-assignment-cutiepatootiekai/blob/master/controllers/cartsController.js https://github.com/soc-DBS/dbs-assignment cutiepatootiekai/blob/master/models/carts.js https://github.com/soc-DBS/dbs-assignment-cutiepatootiekai/tree/master/public/cart https://github.com/soc-DBS/dbs-assignment-cutiepatootiekai/blob/master/routes/carts.js  localhost:3000/cart/create/ localhost:3000 says Added product 11 x 2 ✅ 	

		 localhost:3000/cart/create/ Product Review Cart Logout localhost:3000 says Invalid `prisma.cartItem.upsert()` invocation in C:\Users\itzji\OneDrive\Y2S1\DBS\dbs-assignment-cutiepatootiekai\models\carts.js:14:26 11 async function addToCart(memberId, productId, qty = 1) { 12 if (qty <= 0) throw new Error("Quantity must be >= 1"); 13 const cart = await ensureCart(memberId); → 14 return prisma.cartItem.upsert(Error occurred during query execution: OK  localhost:3000/cart/retrieve/all/index.html localhost:3000 says ✅ Update successful! OK Shopping Cart Product description · Price · Quantity · Update Quantity · Delete Item Soft and cuddly teddy bear, perfect for kids and collectors · USA · 127.50 · 25.50 · 5 · Update · Delete Smart fitness tracker watch with heart rate monitor and step counter · Japan · 140.50 · 140.50 · 3 · Update · Delete Cart Summary Total Quantity: 6 Total Checkout Price: $268.00 Checkout	
3	Checkout feature Implementation	***Backend*** • *getCheckoutSummary* • *bestDiscountPerUnit* • *computeSummary* ***Frontend*** • *currency* • *renderSummary* • *loadSummary* [*https://github.com/soc-DBS/dbs-assignment-cutiepatootiekai/tree/master/public/checkout*](https://github.com/soc-DBS/dbs-assignment-cutiepatootiekai/tree/master/public/checkout) [*https://github.com/soc-DBS/dbs-assignment-cutiepatootiekai/blob/master/controllers/checkoutController.js*](https://github.com/soc-DBS/dbs-assignment-cutiepatootiekai/blob/master/controllers/checkoutController.js) [*https://github.com/soc-DBS/dbs-assignment-cutiepatootiekai/blob/master/models/checkout.js*](https://github.com/soc-DBS/dbs-assignment-cutiepatootiekai/blob/master/models/checkout.js) [*https://github.com/soc-DBS/dbs-assignment-cutiepatootiekai/blob/master/routes/checkout.js*](https://github.com/soc-DBS/dbs-assignment-cutiepatootiekai/blob/master/routes/checkout.js)	4/5



localhost:3000/checkout/

This is the main page for you to implement checkout

Product	Country	Unit Price	Discount / Unit	Final Unit Price	Qty	Line Subtotal	Line Discount	Line Total
Soft and cuddly teddy bear, perfect for kids and collectors	USA	$25.50	-$2.55	$22.95	3	$76.50	-$7.65	$68.85

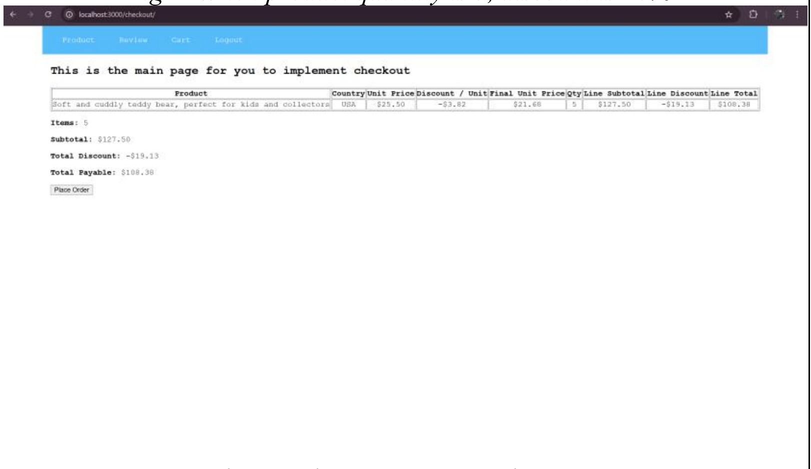
Items: 3

Subtotal: $76.50

Total Discount: -$7.65

Total Payable: $68.85

Fig 1: When product quantity is 3, discount is 10%



localhost:3000/checkout/

This is the main page for you to implement checkout

Product	Country	Unit Price	Discount / Unit	Final Unit Price	Qty	Line Subtotal	Line Discount	Line Total
Soft and cuddly teddy bear, perfect for kids and collectors	USA	$25.50	-$3.82	$21.68	5	$127.50	-$19.13	$108.38

Items: 5

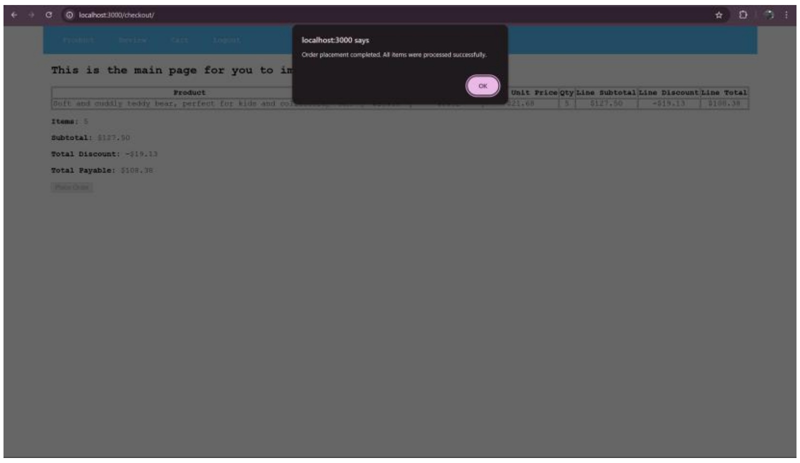
Subtotal: $127.50

Total Discount: -$19.13

Total Payable: $108.38

Fig 2: When product quantity is 5, discount is 15%

Based on the screenshots above, it shows the extensibility of the feature as if there are 2 different discounts for the same product, the feature will select the 1 with the better discounts.

4	Transaction Management	*My stored procedure is place_orders(IN p_member_id integer)* 	3/5

localhost:3000/checkout/

localhost:3000 says

Order placement completed. All items were processed successfully.

This is the main page for you to im

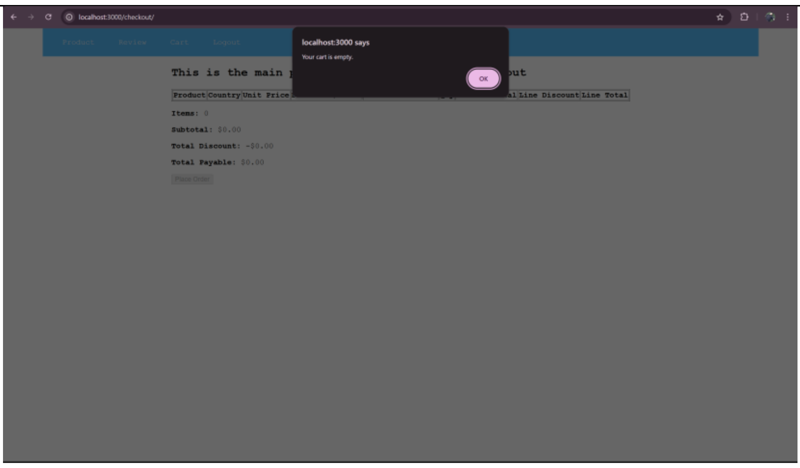
Product				Unit Price	Qty	Line Subtotal	Line Discount	Line Total
Soft and cuddly teddy bear, perfect for kids and co				$21.68	5	$127.50	-$19.13	$108.38

Items: 5

Subtotal: $127.50

Total Discount: -$19.13

Total Payable: $108.38


localhost:3000/checkout/

localhost:3000 says

Your cart is empty.

This is the main p out

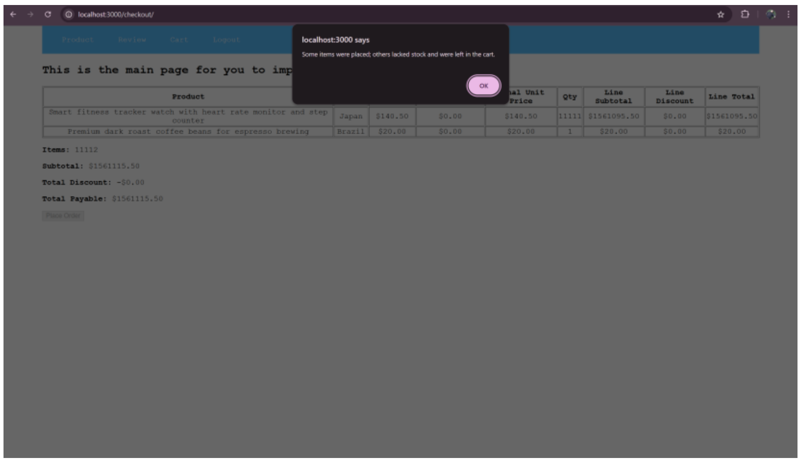
Product	Country	Unit Price		al	Line Discount	Line Total

Items: 0

Subtotal: $0.00

Total Discount: -$0.00

Total Payable: $0.00


localhost:3000/checkout/

localhost:3000 says

Some items were placed; others lacked stock and were left in the cart.

This is the main page for you to imp

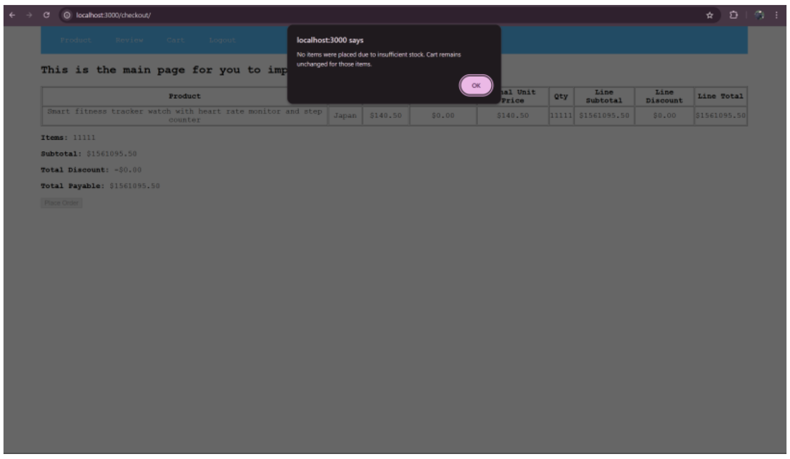
Product				al Unit Price	Qty	Line Subtotal	Line Discount	Line Total
Smart fitness tracker watch with heart rate monitor and step counter	Japan	$140.50	$0.00	$140.50	11111	$1561095.50	$0.00	$1561095.50
Premium dark roast coffee beans for espresso brewing	Brazil	$20.00	$0.00	$20.00	1	$20.00	$0.00	$20.00

Items: 11112

Subtotal: $1561115.50

Total Discount: -$0.00

Total Payable: $1561115.50


localhost:3000/checkout/

localhost:3000 says

No items were placed due to insufficient stock. Cart remains unchanged for those items.

This is the main page for you to imp

Product				al Unit Price	Qty	Line Subtotal	Line Discount	Line Total
Smart fitness tracker watch with heart rate monitor and step counter	Japan	$140.50	$0.00	$140.50	11111	$1561095.50	$0.00	$1561095.50

Items: 11111

Subtotal: $1561095.50

Total Discount: -$0.00

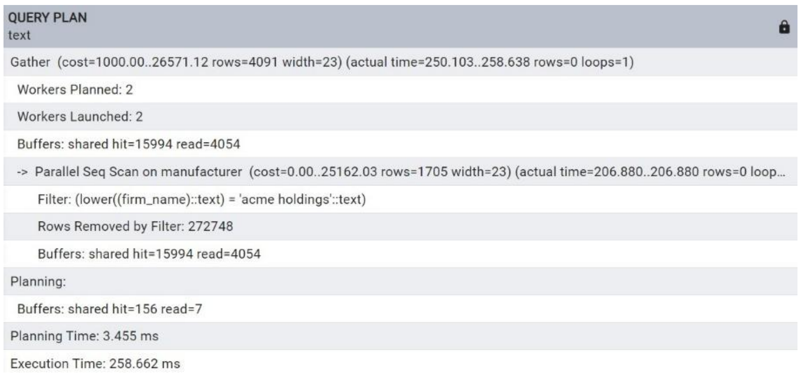
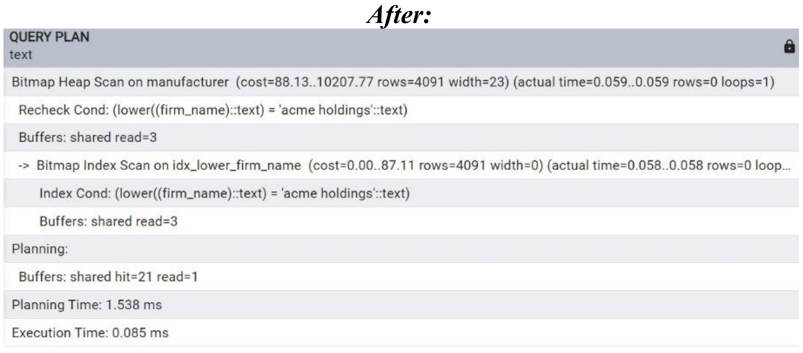
Total Payable: $1561095.50

Backend

- *confirmCheckout*
- *confirm*
- *placeOrdersForMember*

[*https://github.com/soc-DBS/dbs-assignment-cutiepatootiekai/blob/master/routes/checkout.js*](https://github.com/soc-DBS/dbs-assignment-cutiepatootiekai/blob/master/routes/checkout.js)

[*https://github.com/soc-DBS/dbs-assignment-cutiepatootiekai/blob/master/controllers/checkoutController.js*](https://github.com/soc-DBS/dbs-assignment-cutiepatootiekai/blob/master/controllers/checkoutController.js)

		https://github.com/soc-DBS/dbs-assignment-cutiepatootiekai/blob/master/models/checkout.js https://github.com/soc-DBS/dbs-assignment-cutiepatootiekai/tree/master/public/checkout	
5	Indexing	1. **Case-insensitive firm lookup** SELECT id, firm_name FROM manufacturer WHERE lower(firm_name) = lower('Acme Holdings'); Purpose: Find a company by name regardless of letter case. Index: B-tree function(express) index CREATE INDEX idx_lower_firm_name ON manufacturer (lower(firm_name)); ***Before:***  QUERY PLAN text Gather (cost=1000.00..26571.12 rows=4091 width=23) (actual time=250.103..258.638 rows=0 loops=1) Workers Planned: 2 Workers Launched: 2 Buffers: shared hit=15994 read=4054 -> Parallel Seq Scan on manufacturer (cost=0.00..25162.03 rows=1705 width=23) (actual time=206.880..206.880 rows=0 loop... Filter: (lower((firm_name)::text) = 'acme holdings'::text) Rows Removed by Filter: 272748 Buffers: shared hit=15994 read=4054 Planning: Buffers: shared hit=156 read=7 Planning Time: 3.455 ms Execution Time: 258.662 ms ***After:***  QUERY PLAN text Bitmap Heap Scan on manufacturer (cost=88.13..10207.77 rows=4091 width=23) (actual time=0.059..0.059 rows=0 loops=1) Recheck Cond: (lower((firm_name)::text) = 'acme holdings'::text) Buffers: shared read=3 -> Bitmap Index Scan on idx_lower_firm_name (cost=0.00..87.11 rows=4091 width=0) (actual time=0.058..0.058 rows=0 loop... Index Cond: (lower((firm_name)::text) = 'acme holdings'::text) Buffers: shared read=3 Planning: Buffers: shared hit=21 read=1 Planning Time: 1.538 ms Execution Time: 0.085 ms Reason: Creating a B-tree expression index on lower(firm_name) made the predicate indexable, so the optimizer replaced a Sequential Scan with a Bitmap Index Scan + Bitmap Heap Scan, avoiding a full table read and per-row LOWER() evaluation.	3/5

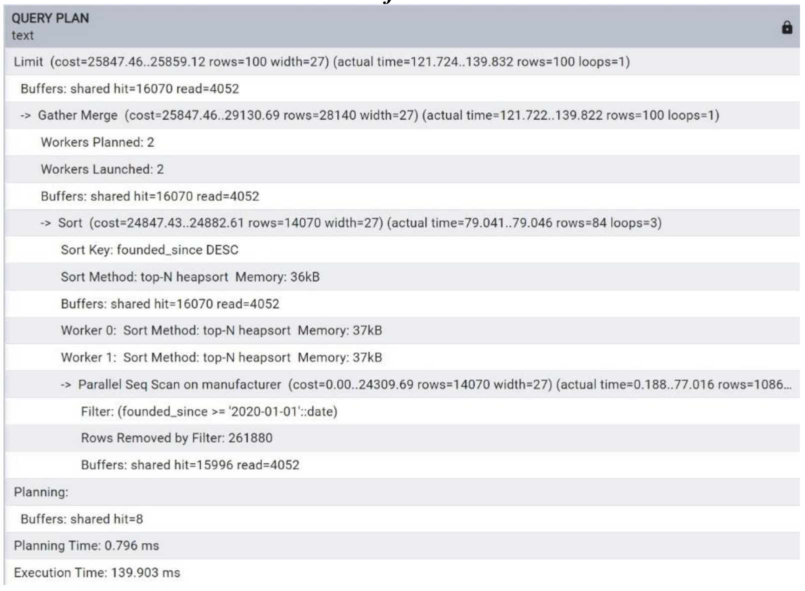
2. **Recent companies by founded date (latest first)**

```
SELECT id, firm_name, founded_since
FROM manufacturer
WHERE founded_since >= DATE '2020-01-01'
ORDER BY founded_since DESC
LIMIT 100;
```

Purpose: List recently founded firms, newest first, top 100.
Index: B-tree Partial, covering index

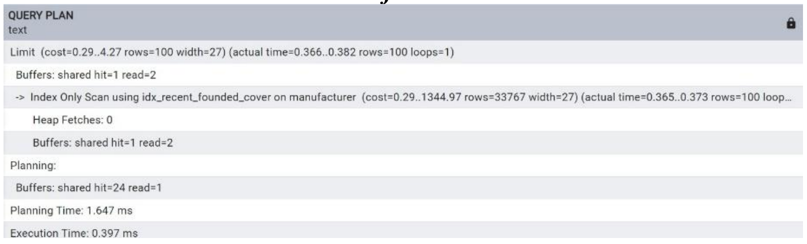
```
CREATE INDEX idx_recent_founded_cover
ON manufacturer (founded_since DESC)
INCLUDE (id, firm_name)
WHERE founded_since >= DATE '2020-01-01';
```

Before:



```
QUERY PLAN
text
Limit  (cost=25847.46..25859.12 rows=100 width=27) (actual time=121.724..139.832 rows=100 loops=1)
  Buffers: shared hit=16070 read=4052
  ->  Gather Merge  (cost=25847.46..29130.69 rows=28140 width=27) (actual time=121.722..139.822 rows=100 loops=1)
        Workers Planned: 2
        Workers Launched: 2
        Buffers: shared hit=16070 read=4052
        ->  Sort  (cost=24847.43..24882.61 rows=14070 width=27) (actual time=79.041..79.046 rows=84 loops=3)
              Sort Key: founded_since DESC
              Sort Method: top-N heapsort  Memory: 36kB
              Buffers: shared hit=16070 read=4052
              Worker 0:  Sort Method: top-N heapsort  Memory: 37kB
              Worker 1:  Sort Method: top-N heapsort  Memory: 37kB
              ->  Parallel Seq Scan on manufacturer  (cost=0.00..24309.69 rows=14070 width=27) (actual time=0.188..77.016 rows=1086...
                    Filter: (founded_since >= '2020-01-01'::date)
                    Rows Removed by Filter: 261880
                    Buffers: shared hit=15996 read=4052
Planning:
  Buffers: shared hit=8
Planning Time: 0.796 ms
Execution Time: 139.903 ms
```

After:



```
QUERY PLAN
text
Limit  (cost=0.29..4.27 rows=100 width=27) (actual time=0.366..0.382 rows=100 loops=1)
  Buffers: shared hit=1 read=2
  ->  Index Only Scan using idx_recent_founded_cover on manufacturer  (cost=0.29..1344.97 rows=33767 width=27) (actual time=0.365..0.373 rows=100 loop...
        Heap Fetches: 0
        Buffers: shared hit=1 read=2
Planning:
  Buffers: shared hit=24 read=1
Planning Time: 1.647 ms
Execution Time: 0.397 ms
```

Reason : Creating a partial, covering B-tree on (founded_since DESC) INCLUDE (id, firm_name) WHERE founded_since >= '2020-01-01' aligned the WHERE and ORDER BY, enabling an Index Only Scan with early LIMIT and removing both the full scan and the sort.

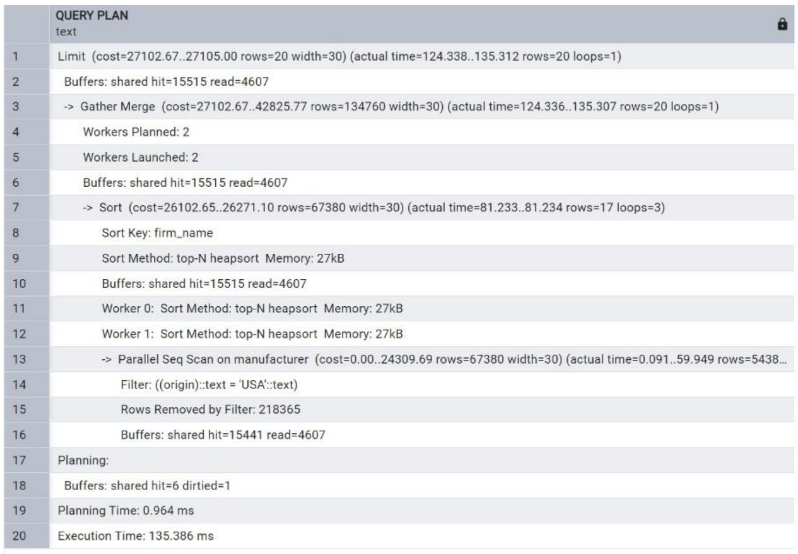
3. **Country filter (USA) sorted by name**

```
SELECT id, firm_name, origin
FROM manufacturer
WHERE origin = 'USA'
ORDER BY firm_name
LIMIT 20;
```

Purpose: First 20 USA firms, alphabetically.
Index: B-tree composite index

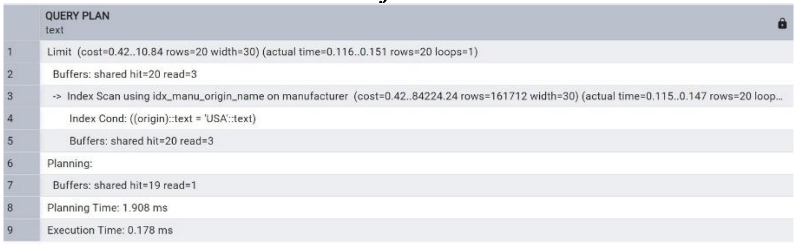
```
CREATE INDEX idx_manu_origin_name ON manufacturer (origin,
firm_name);
```

Before:



```
QUERY PLAN
text
Limit  (cost=27102.67..27105.00 rows=20 width=30) (actual time=124.338..135.312 rows=20 loops=1)
  Buffers: shared hit=15515 read=4607
  ->  Gather Merge  (cost=27102.67..42825.77 rows=134760 width=30) (actual time=124.336..135.307 rows=20 loops=1)
        Workers Planned: 2
        Workers Launched: 2
        Buffers: shared hit=15515 read=4607
        ->  Sort  (cost=26102.65..26271.10 rows=67380 width=30) (actual time=81.233..81.234 rows=17 loops=3)
              Sort Key: firm_name
              Sort Method: top-N heapsort  Memory: 27kB
              Buffers: shared hit=15515 read=4607
              Worker 0:  Sort Method: top-N heapsort  Memory: 27kB
              Worker 1:  Sort Method: top-N heapsort  Memory: 27kB
              ->  Parallel Seq Scan on manufacturer  (cost=0.00..24309.69 rows=67380 width=30) (actual time=0.091..59.949 rows=5438...
                    Filter: ((origin)::text = 'USA'::text)
                    Rows Removed by Filter: 218365
                    Buffers: shared hit=15441 read=4607
Planning:
  Buffers: shared hit=6 dirtied=1
Planning Time: 0.964 ms
Execution Time: 135.386 ms
```

After:



```
QUERY PLAN
text
Limit  (cost=0.42..10.84 rows=20 width=30) (actual time=0.116..0.151 rows=20 loops=1)
  Buffers: shared hit=20 read=3
  ->  Index Scan using idx_manu_origin_name on manufacturer  (cost=0.42..84224.24 rows=161712 width=30) (actual time=0.115..0.147 rows=20 loop...
        Index Cond: ((origin)::text = 'USA'::text)
        Buffers: shared hit=20 read=3
Planning:
  Buffers: shared hit=19 read=1
Planning Time: 1.908 ms
Execution Time: 0.178 ms
```

Reason: Creating a composite B-tree on (origin, firm_name) let Postgres use the index as a pointer straight to the origin='USA' range instead of scanning the whole table, and because B-trees support **ordered access**, rows come out already sorted by firm_name—so the planner switches from a Sequential Scan + sort to an ordered Index Scan that can stop at LIMIT 20.

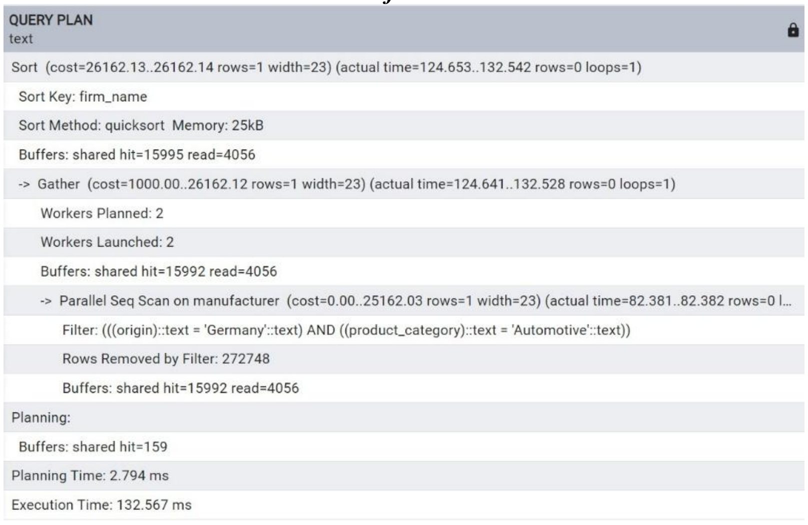
4. **Origin + category filter, sorted by name**

SELECT firm_name, id
FROM manufacturer
WHERE origin = 'Germany' AND product_category = 'Automotive'
ORDER BY firm_name;

Purpose: Germany automotive makers, A→Z.
Index: B-tree covering composite index

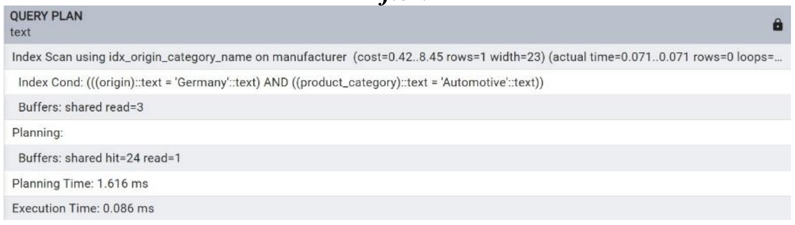
CREATE INDEX idx_origin_category_name
ON manufacturer (origin, product_category, firm_name);

Before:



```
QUERY PLAN
text
Sort  (cost=26162.13..26162.14 rows=1 width=23) (actual time=124.653..132.542 rows=0 loops=1)
  Sort Key: firm_name
  Sort Method: quicksort  Memory: 25kB
  Buffers: shared hit=15995 read=4056
  ->  Gather  (cost=1000.00..26162.12 rows=1 width=23) (actual time=124.641..132.528 rows=0 loops=1)
        Workers Planned: 2
        Workers Launched: 2
        Buffers: shared hit=15992 read=4056
        ->  Parallel Seq Scan on manufacturer  (cost=0.00..25162.03 rows=1 width=23) (actual time=82.381..82.382 rows=0 l...
              Filter: (((origin)::text = 'Germany'::text) AND ((product_category)::text = 'Automotive'::text))
              Rows Removed by Filter: 272748
              Buffers: shared hit=15992 read=4056
Planning:
  Buffers: shared hit=159
Planning Time: 2.794 ms
Execution Time: 132.567 ms
```

After:



```
QUERY PLAN
text
Index Scan using idx_origin_category_name on manufacturer  (cost=0.42..8.45 rows=1 width=23) (actual time=0.071..0.071 rows=0 loops=...
  Index Cond: (((origin)::text = 'Germany'::text) AND ((product_category)::text = 'Automotive'::text))
  Buffers: shared read=3
Planning:
  Buffers: shared hit=24 read=1
Planning Time: 1.616 ms
Execution Time: 0.086 ms
```

Reason: Creating a composite B-tree on (origin, product_category, firm_name) lines up with both the equality filters and the ORDER BY, letting PostgreSQL use an Index Scan to jump via the index "pointer" straight to the matching, already-sorted key range—so it reads far fewer pages and avoids a table scan and extra sort.

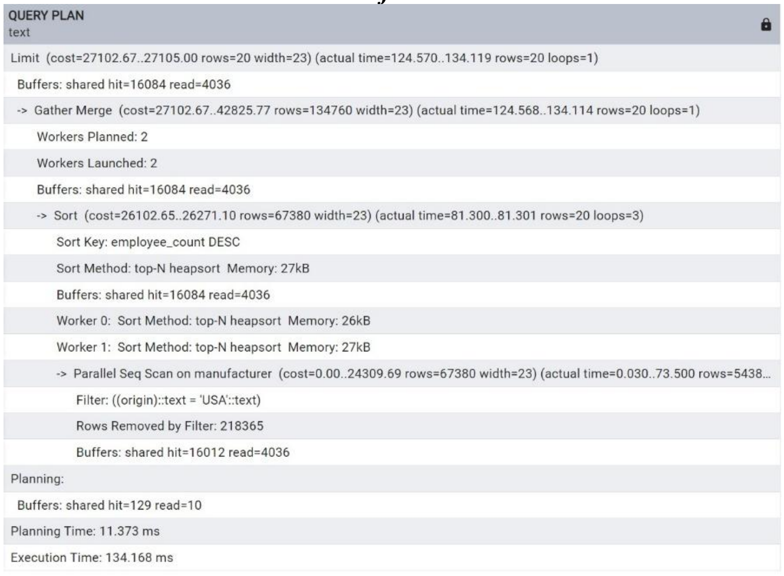
5. **Top employers in a country (USA)**

```
SELECT firm_name, employee_count
FROM manufacturer
WHERE origin = 'USA'
ORDER BY employee_count DESC
LIMIT 20;
```

Purpose: Biggest US employers, top 20 by headcount.
Index: B-tree composite index

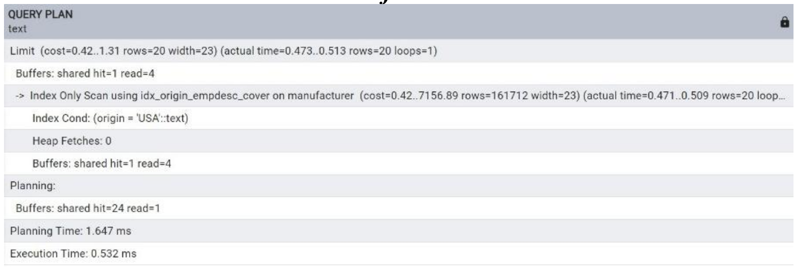
```
CREATE INDEX idx_origin_empdesc_cover
ON manufacturer (origin, employee_count DESC)
INCLUDE (firm_name);
```

Before:



```
QUERY PLAN
text
Limit  (cost=27102.67..27105.00 rows=20 width=23) (actual time=124.570..134.119 rows=20 loops=1)
  Buffers: shared hit=16084 read=4036
  -> Gather Merge  (cost=27102.67..42825.77 rows=134760 width=23) (actual time=124.568..134.114 rows=20 loops=1)
        Workers Planned: 2
        Workers Launched: 2
        Buffers: shared hit=16084 read=4036
        -> Sort  (cost=26102.65..26271.10 rows=67380 width=23) (actual time=81.300..81.301 rows=20 loops=3)
              Sort Key: employee_count DESC
              Sort Method: top-N heapsort  Memory: 27kB
              Buffers: shared hit=16084 read=4036
              Worker 0:  Sort Method: top-N heapsort  Memory: 26kB
              Worker 1:  Sort Method: top-N heapsort  Memory: 27kB
              -> Parallel Seq Scan on manufacturer  (cost=0.00..24309.69 rows=67380 width=23) (actual time=0.030..73.500 rows=5438...
                    Filter: ((origin)::text = 'USA'::text)
                    Rows Removed by Filter: 218365
                    Buffers: shared hit=16012 read=4036
Planning:
  Buffers: shared hit=129 read=10
Planning Time: 11.373 ms
Execution Time: 134.168 ms
```

After:



```
QUERY PLAN
text
Limit  (cost=0.42..1.31 rows=20 width=23) (actual time=0.473..0.513 rows=20 loops=1)
  Buffers: shared hit=1 read=4
  -> Index Only Scan using idx_origin_empdesc_cover on manufacturer  (cost=0.42..7156.89 rows=161712 width=23) (actual time=0.471..0.509 rows=20 loop...
        Index Cond: (origin = 'USA'::text)
        Heap Fetches: 0
        Buffers: shared hit=1 read=4
Planning:
  Buffers: shared hit=24 read=1
Planning Time: 1.647 ms
Execution Time: 0.532 ms
```

Reason: Creating a covering composite B-tree on (origin, employee_count DESC) INCLUDE (firm_name) aligns the filter and DESC order, so the optimizer can walk the index in order and stop after the first 20 entries via an Index Only Scan, avoiding a full-table read and separate sort.

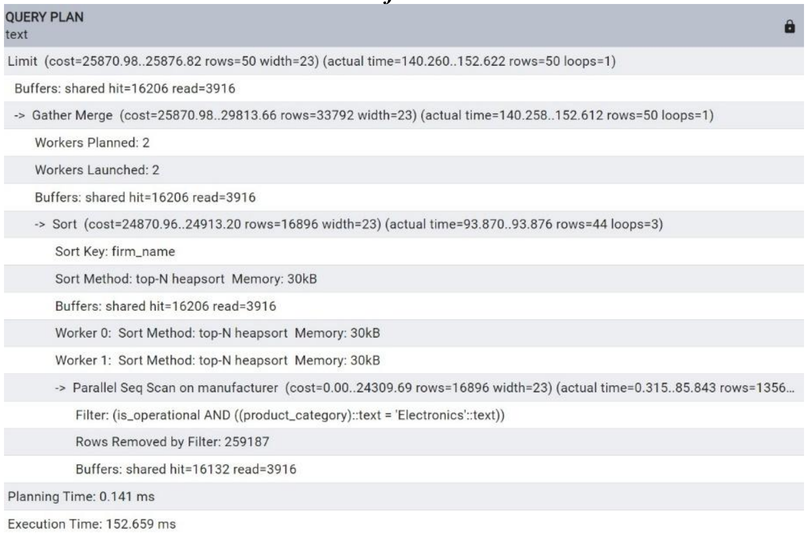
6. **Operational electronics firms, sorted by name**

```
SELECT firm_name, id
FROM manufacturer
WHERE product_category = 'Electronics' AND is_operational = true
ORDER BY firm_name
LIMIT 50;
```

Purpose: Active electronics companies, A→Z, first 50.
Index: B-tree Partial, covering, composite

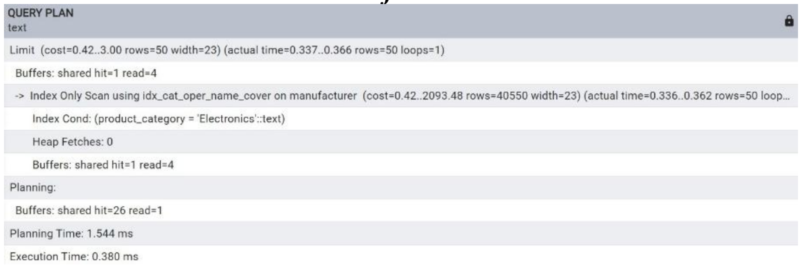
```
CREATE INDEX idx_cat_oper_name_cover
ON manufacturer (product_category, is_operational, firm_name)
INCLUDE (id)
WHERE is_operational = true;
```

Before:



```
QUERY PLAN
text
Limit  (cost=25870.98..25876.82 rows=50 width=23) (actual time=140.260..152.622 rows=50 loops=1)
  Buffers: shared hit=16206 read=3916
  -> Gather Merge  (cost=25870.98..29813.66 rows=33792 width=23) (actual time=140.258..152.612 rows=50 loops=1)
       Workers Planned: 2
       Workers Launched: 2
       Buffers: shared hit=16206 read=3916
       -> Sort  (cost=24870.96..24913.20 rows=16896 width=23) (actual time=93.870..93.876 rows=44 loops=3)
            Sort Key: firm_name
            Sort Method: top-N heapsort  Memory: 30kB
            Buffers: shared hit=16206 read=3916
            Worker 0:  Sort Method: top-N heapsort  Memory: 30kB
            Worker 1:  Sort Method: top-N heapsort  Memory: 30kB
            -> Parallel Seq Scan on manufacturer  (cost=0.00..24309.69 rows=16896 width=23) (actual time=0.315..85.843 rows=1356...
                 Filter: (is_operational AND ((product_category)::text = 'Electronics'::text))
                 Rows Removed by Filter: 259187
                 Buffers: shared hit=16132 read=3916
Planning Time: 0.141 ms
Execution Time: 152.659 ms
```

After:



```
QUERY PLAN
text
Limit  (cost=0.42..3.00 rows=50 width=23) (actual time=0.337..0.366 rows=50 loops=1)
  Buffers: shared hit=1 read=4
  -> Index Only Scan using idx_cat_oper_name_cover on manufacturer  (cost=0.42..2093.48 rows=40550 width=23) (actual time=0.336..0.362 rows=50 loop...
       Index Cond: (product_category = 'Electronics'::text)
       Heap Fetches: 0
       Buffers: shared hit=1 read=4
Planning:
  Buffers: shared hit=26 read=1
Planning Time: 1.544 ms
Execution Time: 0.380 ms
```

Reason: Creating a partial, covering composite B-tree on (product_category, is_operational, firm_name) INCLUDE (id) WHERE is_operational = true aligned the filter and A→Z order, so PostgreSQL used an Index Only Scan over a small, already-sorted subset and stopped at LIMIT 50, avoiding a table scan and extra sort.

9	Report Quality	*Based on quality of documentation for above criteria.* *No inputs required here.*	3/5
10	Demonstration & Interview	*Based on assessment during demonstration & interview.* *No inputs required here.*	-