Database Programming Project 2

Nada Mustafa 10005872 T-4:query 1-8
Ranaa ahmed 10000755 T-6:query 9-16
Amira ayman : 10003691 T-5query 17-24

Query 1

Physical Plan Before:

	QUERY PLAN text
1	Hash Join (cost=1.31..256.85 rows=298 width=175) (actual time=0.231..1.030 rows=403 loops=1)
2	Hash Cond: (fd.fdgrp_cd = fg.fdgrp_cd)
3	-> Seq Scan on food_des fd (cost=0.00..233.46 rows=7146 width=151) (actual time=0.004..0.423 rows=7146 loops=1)
4	-> Hash (cost=1.30..1.30 rows=1 width=24) (actual time=0.009..0.010 rows=1 loops=1)
5	Buckets: 1024 Batches: 1 Memory Usage: 9kB
6	-> Seq Scan on fd_group fg (cost=0.00..1.30 rows=1 width=24) (actual time=0.006..0.007 rows=1 loops=1)
7	Filter: (fddrp_desc = 'Breakfast Cereals'::text)
8	Rows Removed by Filter: 23
9	Planning Time: 0.494 ms
10	Execution Time: 1.079 ms

Highest Cost:The seq scan on food_des at line 3, `cost=233.`

Slowest Runtime:The **Hash Join** at line 1, `actual time=0.597ms`.

Largest Number of Rows:The **Seq Scan on food_des fd** at line 3,`rows=7146`.

Physical Plan After:
create index idx1 food_des(fdgrp_cd)
Index: `fdgrp_cd` on `food_des` (B-Tree).
Justification: Optimizes the join condition on `fdgrp_cd`.

	QUERY PLAN text
1	Nested Loop (cost=0.28..166.13 rows=298 width=175) (actual time=0.043..0.216 rows=403 loops=1)
2	-> Seq Scan on fd_group fg (cost=0.00..1.30 rows=1 width=24) (actual time=0.013..0.016 rows=1 loops=1)
3	Filter: (fddrp_desc = 'Breakfast Cereals'::text)
4	Rows Removed by Filter: 23
5	-> Index Scan using idx1 on food_des fd (cost=0.28..161.85 rows=298 width=151) (actual time=0.027..0.115 rows=403 loops=1)
6	Index Cond: (fdgrp_cd = fg.fdgrp_cd)
7	Planning Time: 0.475 ms
8	Execution Time: 0.261 ms

Highest Cost:The **index scan in line 5**, `cost=162`.

Slowest Runtime:The **index scan in line 5** ,`actual time=0.115`.

Largest Number of Rows:The **Index Scan on food_des fd** at line 5 and the **Nested Loop** at line 1 ,`rows=403`.

Before Index Creation:

- Used **Sequential Scan** on `food_des`, with a **high cost of 233** and a **Hash Join runtime of 0.597 ms**.
- The **Sequential Scan** processed the **largest number of rows (7146)**.

After Index Creation:

- Switched to an **Index Scan** on `food_des`, reducing the **cost to 162** and improving the **runtime to 0.115 ms**.
- The **Index Scan** and **Nested Loop** processed a reduced number of **403 rows**.

Reason for the Observed Behavior:

- **Before Index**: Sequential Scan accessed all rows, leading to higher cost and slower runtime due to processing unnecessary rows.
- **After Index**: The Index Scan efficiently filtered rows, reducing the number of rows processed, which lowered both cost and runtime significantly.

Query 2

Physical Plan Before:

	QUERY PLAN text
1	Seq Scan on weight (cost=0.00..287.61 rows=6666 width=50) (actual time=0.012..0.994 rows=6666 loops=1)
2	Filter: (seq = '1'::bpchar)
3	Rows Removed by Filter: 6343
4	Planning Time: 0.078 ms
5	Execution Time: 1.182 ms

Highest Cost:The **Sequential Scan on weight** at line 1, `cost=288`.
Slowest Runtime:The **Sequential Scan on weight** at line 1, `actual time=0.994`.
Largest Number of Rows:The **Sequential Scan on weight** at line 1 `rows=6666`+6343=13009

Physical Plan After:
create index idx_2 on weight (seq) where seq = '1';
Index: Partial index on `seq` (B-Tree).
Justification: Speeds up filtering for `seq = '1'`

	QUERY PLAN text
1	Bitmap Heap Scan on weight (cost=67.28..275.60 rows=6666 width=50) (actual time=0.319..1.332 rows=6666 loops=1)
2	Recheck Cond: (seq = '1'::bpchar)
3	Heap Blocks: exact=124
4	-> Bitmap Index Scan on idx_2 (cost=0.00..65.61 rows=6666 width=0) (actual time=0.294..0.295 rows=6666 loops=1)
5	Planning Time: 0.362 ms
6	Execution Time: 1.640 ms

Highest Cost:The **Bitmap Heap Scan on weight** at line 1, `cost=210`.
Slowest Runtime:The **Bitmap Heap Scan on weight** at line 1, `actual time=1.04`.
Largest Number of Rows:Both the **Bitmap Heap Scan** and **Bitmap Index Scan** processed `6666` rows.

Before Index Creation:

- Used **Sequential Scan** on `weight`, with a **high cost of 288** and an **execution time of 0.994 ms**.

- The **Sequential Scan** initially processed **13009 rows (6666 kept + 6343 removed by the filter)**.

After Index Creation:

- Switched to a **Bitmap Heap Scan** with a **Bitmap Index Scan**, reducing the **cost to 210** but increasing the **execution time to 1.04 ms**.
- Both the **Bitmap Heap Scan** and **Bitmap Index Scan** processed **6666 rows**.

Reason for the Observed Behavior:

- **Before Index**: The Sequential Scan processed all rows, resulting in a higher cost but lower execution time due to no index overhead.
- **After Index**: The Bitmap operations introduced additional processing steps, increasing execution time slightly while reducing the cost. This shows the index's overhead outweighed its benefit for this query.

Query 3

Physical Plan Before:

	QUERY PLAN text
1	Seq Scan on weight (cost=0.00..320.13 rows=144 width=50) (actual time=0.016..0.635 rows=60 loops=1)
2	Filter: ((amount > '3'::double precision) AND (seq = '1'::bpchar))
3	Rows Removed by Filter: 12949
4	Planning Time: 0.240 ms
5	Execution Time: 0.647 ms

Highest Cost:The **Sequential Scan on weight** at line 1 ,`cost=320`.
Slowest Runtime:The **Sequential Scan on weight** at line 1 ,`actual time=0.635`.
Largest Number of Rows:The **Sequential Scan on weight** at line 1 , 12949+60=13009 rows.

Physical Plan After:
create index idx_seq_amount on weight(seq,amount);
Index: Composite index on `seq` and `amount` (B-Tree).
Justification: Optimizes filtering on `seq` and `amount`.

	QUERY PLAN text
1	Bitmap Heap Scan on weight (cost=5.76..139.14 rows=144 width=50) (actual time=0.034..0.064 rows=60 loops=1)
2	Recheck Cond: ((seq = '1'::bpchar) AND (amount > '3'::double precision))
3	Heap Blocks: exact=20
4	-> Bitmap Index Scan on idx_seq_amount (cost=0.00..5.72 rows=144 width=0) (actual time=0.026..0.026 rows=60 loops=1)
5	Index Cond: ((seq = '1'::bpchar) AND (amount > '3'::double precision))
6	Planning Time: 0.091 ms
7	Execution Time: 0.082 ms

Highest Cost:The **Bitmap Heap Scan on weight** at line 1 ,`cost=133`.
Slowest Runtime:The **Bitmap Heap Scan on weight** at line 1 ,`actual time=0.038`.
Largest Number of Rows:Both the **Bitmap Heap Scan** and **Bitmap Index Scan** process `60` rows.

Before Index Creation:

- Used **Sequential Scan** on `weight`, with a **high cost of 320** and an **execution time of 0.635 ms**.
- The **Sequential Scan** initially processed **13009 rows (12949 removed + 60 kept)**.

After Index Creation:

- Switched to a **Bitmap Heap Scan** with a **Bitmap Index Scan**, reducing the **cost to 133** and significantly improving the **execution time to 0.038 ms**.
- Both the **Bitmap Heap Scan** and **Bitmap Index Scan** processed **60 rows**.

Reason for the Observed Behavior:

- **Before Index**: Sequential Scan processed all rows, leading to higher cost and slower runtime due to unnecessary row access.
- **After Index**: The combined index on `seq` and `amount` allowed efficient filtering, dramatically reducing the number of rows scanned and lowering both cost and runtime significantly.

Query 4

Physical Plan Before:

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	HashSetOp Except (cost=0.00..803.27 rows=6679 width=28) (actual time=5.878..5.880 rows=0 loops=1)
2	-> Append (cost=0.00..752.89 rows=20155 width=28) (actual time=0.008..3.579 rows=20155 loops=1)
3	-> Subquery Scan on "*SELECT* 1" (cost=0.00..385.18 rows=13009 width=10) (actual time=0.007..1.620 rows=13009 loops=1)
4	-> Seq Scan on weight (cost=0.00..255.09 rows=13009 width=6) (actual time=0.006..0.824 rows=13009 loops=1)
5	-> Subquery Scan on "*SELECT* 2" (cost=0.28..266.93 rows=7146 width=10) (actual time=0.058..1.043 rows=7146 loops=1)
6	-> Index Only Scan using food_des_pkey on food_des (cost=0.28..195.47 rows=7146 width=6) (actual time=0.056..0.609 rows=7146 loops=1)
7	Heap Fetches: 0
8	Planning Time: 0.124 ms
9	Execution Time: 5.919 ms

Highest Cost: The **seq scan on weight at line 4** ,`cost=255`
Slowest Runtime:The **HashSetOp Except** at line 1 ,`actual time=2.3 ms`
Largest Number of Rows:The **Append** at line 2 , `20,155` rows

Physical Plan After:
CREATE INDEX idx_weight_ndb_no ON weight(ndb_no);
Index: `ndb_no` on `weight` (B-Tree).
Justification: Improves set operations on `ndb_no`.

Turned off seq scan

	QUERY PLAN text
1	HashSetOp Except (cost=0.29..871.61 rows=6679 width=28) (actual time=6.470..6.471 rows=0 loops=1)
2	-> Append (cost=0.29..821.22 rows=20155 width=28) (actual time=0.079..3.884 rows=20155 loops=1)
3	-> Subquery Scan on "*SELECT* 1" (cost=0.29..453.51 rows=13009 width=10) (actual time=0.079..1.919 rows=13009 loops=1)
4	-> Index Only Scan using idx_weight_ndb_no on weight (cost=0.29..323.42 rows=13009 width=6) (actual time=0.077..1.116 rows=13009 loops=1)
5	Heap Fetches: 0
6	-> Subquery Scan on "*SELECT* 2" (cost=0.28..266.93 rows=7146 width=10) (actual time=0.032..0.924 rows=7146 loops=1)
7	-> Index Only Scan using food_des_pkey on food_des (cost=0.28..195.47 rows=7146 width=6) (actual time=0.030..0.509 rows=7146 loops=1)
8	Heap Fetches: 0
9	Planning Time: 0.077 ms
10	Execution Time: 6.517 ms

Highest Cost: The **index only scan** at line 4 ,`cost=323`
Slowest Runtime:The **HashSetOp Except** at line 1 ,`actual time= 2.59 ms`
Largest Number of Rows:The **Append** at line 2 , `20,155` rows

Before Index Creation:

- Used **Sequential Scan** on `weight`, with a **cost of 255** and a **HashSetOp Except runtime of 2.3 ms**.
- The **Append** operation processed the **largest number of rows (20,155)**.

After Index Creation:

- Switched to an **Index Only Scan** on `weight`, increasing the **cost to 323** and slightly increasing the **HashSetOp Except runtime to 2.59 ms**.
- The **Append** operation still processed the **same 20,155 rows**.

Reason for the Observed Behavior:

- **Before Index**: Sequential Scan processed all rows, leading to lower cost and slightly faster runtime due to no index overhead.
- **After Index**: The Index Only Scan introduced additional lookup overhead, increasing the cost and runtime slightly. The number of rows processed remained unchanged.

Query 5

Physical Plan Before:

	QUERY PLAN text
1	HashSetOp Except (cost=0.00..3661.48 rows=6679 width=28) (actual time=42.663..43.703 rows=5077 loops=1)
2	-> Append (cost=0.00..3394.35 rows=106854 width=28) (actual time=0.017..26.155 rows=106854 loops=1)
3	-> Subquery Scan on "*SELECT* 1" (cost=0.00..385.18 rows=13009 width=10) (actual time=0.016..2.734 rows=13009 loops=1)
4	-> Seq Scan on weight (cost=0.00..255.09 rows=13009 width=6) (actual time=0.015..1.544 rows=13009 loops=1)
5	-> Subquery Scan on "*SELECT* 2" (cost=0.00..2474.90 rows=93845 width=10) (actual time=0.031..16.839 rows=93845 loops=1)
6	-> Seq Scan on datsrcln (cost=0.00..1536.45 rows=93845 width=6) (actual time=0.029..9.107 rows=93845 loops=1)
7	Planning Time: 0.120 ms
8	Execution Time: 44.131 ms

Highest Cost: The **seq scan on datsrcln** at line 6,`cost=1540`

Slowest Runtime:The **HashSetOp Except** at line 1 ,`actual time=17.5 ms`

Largest Number of Rows:The **Append** at line 2 , `106,854` rows

Physical Plan After:

CREATE INDEX idx_datsrcln_ndb_no ON datsrcln(ndb_no);

Index: `ndb_no` on `datsrcln` (B-Tree).

Justification: Optimizes set operations for `ndb_no`.

Turned off seq scan

	QUERY PLAN text
1	HashSetOp Except (cost=0.29..4021.33 rows=6679 width=28) (actual time=27.489..27.876 rows=5077 loops=1)
2	-> Append (cost=0.29..3754.20 rows=106854 width=28) (actual time=0.032..17.983 rows=106854 loops=1)
3	-> Subquery Scan on "*SELECT* 1" (cost=0.29..533.51 rows=13009 width=10) (actual time=0.030..1.959 rows=13009 loops=1)
4	-> Index Only Scan using weight_pkey on weight (cost=0.29..403.42 rows=13009 width=6) (actual time=0.028..1.193 rows=13009 loops=1)
5	Heap Fetches: 0
6	-> Subquery Scan on "*SELECT* 2" (cost=0.29..2686.42 rows=93845 width=10) (actual time=0.112..11.160 rows=93845 loops=1)
7	-> Index Only Scan using idx_datsrcln_ndb_no on datsrcln (cost=0.29..1747.97 rows=93845 width=6) (actual time=0.110..5.563 rows=93845 loops=1)
8	Heap Fetches: 0
9	Planning Time: 0.160 ms
10	Execution Time: 28.131 ms

Highest Cost: The **index only scan on datsrcln** at line 7 ,`cost= 1750`
Slowest Runtime:The **HashSetOp Except** at line 1 ,`actual time=9.89`
Largest Number of Rows:The **Append** at line 2 , `106,854` rows

Before Index Creation:

- Used **Sequential Scan** on `datsrcln`, with a **cost of 1540** and a **HashSetOp Except runtime of 17.5 ms**.
- The **Append** operation processed the **largest number of rows (106,854)**.

After Index Creation:

- Switched to an **Index Only Scan** on `datsrcln`, increasing the **cost to 1750** but significantly reducing the **HashSetOp Except runtime to 9.89 ms**.
- The **Append** operation still processed the **same 106,854 rows**.

Reason for the Observed Behavior:

- **Before Index**: Sequential Scan on `datsrcln` led to a higher cost and slower runtime due to full table scans.
- **After Index**: The Index Only Scan improved runtime by limiting row access to the index but increased cost slightly due to index overhead. Rows processed remained the same.

Query 6

Physical Plan Before:

	QUERY PLAN text
1	Hash Join (cost=5889.29..6229.51 rows=3634 width=20) (actual time=7.071..9.131 rows=4560 loops=1)
2	Hash Cond: (w.ndb_no = nd.ndb_no)
3	-> Seq Scan on weight w (cost=0.00..255.09 rows=13009 width=14) (actual time=0.004..0.692 rows=13009 loops=1)
4	-> Hash (cost=5865.97..5865.97 rows=1866 width=18) (actual time=7.061..7.063 rows=2859 loops=1)
5	Buckets: 4096 (originally 2048) Batches: 1 (originally 1) Memory Usage: 172kB
6	-> Nested Loop (cost=0.42..5865.97 rows=1866 width=18) (actual time=0.015..6.650 rows=2859 loops=1)
7	-> Seq Scan on nutr_def nutdef (cost=0.00..3.70 rows=1 width=16) (actual time=0.006..0.017 rows=1 loops=1)
8	Filter: (tagname = 'CA'::text)
9	Rows Removed by Filter: 135
10	-> Index Only Scan using nut_data_pkey on nut_data nd (cost=0.42..5843.19 rows=1908 width=10) (actual time=0.007..6.451 rows=2859 loops=1)
11	Index Cond: (nutr_no = nutdef.nutr_no)
12	Heap Fetches: 0
13	Planning Time: 0.401 ms
14	Execution Time: 9.248 ms

Highest Cost: The `index only scan` at line 10 ,`cost= 5840`

Slowest Runtime:The `index only scan` at line 10 ,`actual time=6.41 ms`

Largest Number of Rows:The `Seq Scan on weight` at line 3 , `13,009` rows

Physical Plan After:

CREATE INDEX idx_weight_ndb_no ON weight(ndb_no);

Index: `ndb_no` on `weight` (B-Tree).

Justification: Accelerates joins on `ndb_no`.

Turned off seq scan

	QUERY PLAN text
1	Hash Join (cost=5901.40..6437.66 rows=3634 width=20) (actual time=7.895..10.316 rows=4560 loops=1)
2	Hash Cond: (w.ndb_no = nd.ndb_no)
3	-> Index Scan using idx_weight_ndb_no on weight w (cost=0.29..451.42 rows=13009 width=14) (actual time=0.010..1.165 rows=13009 loops=1)
4	-> Hash (cost=5877.79..5877.79 rows=1866 width=18) (actual time=7.872..7.873 rows=2859 loops=1)
5	Buckets: 4096 (originally 2048) Batches: 1 (originally 1) Memory Usage: 172kB
6	-> Nested Loop (cost=0.56..5877.79 rows=1866 width=18) (actual time=0.030..7.422 rows=2859 loops=1)
7	-> Index Scan using nutr_def_pkey on nutr_def nutdef (cost=0.14..15.52 rows=1 width=16) (actual time=0.012..0.031 rows=1 loops=1)
8	Filter: (tagname = 'CA'::text)
9	Rows Removed by Filter: 135
10	-> Index Only Scan using nut_data_pkey on nut_data nd (cost=0.42..5843.19 rows=1908 width=10) (actual time=0.015..7.194 rows=2859 loops=1)
11	Index Cond: (nutr_no = nutdef.nutr_no)
12	Heap Fetches: 0
13	Planning Time: 1.007 ms
14	Execution Time: 10.465 ms

Highest Cost: The `index only scan` at line 10 ,`cost= 5840`

Slowest Runtime:The `index only scan` at line 10 ,`actual time=7.19 ms`

Largest Number of Rows:The `index scan on weight` at line 3 , `13,009` rows

Before Index Creation:

- Used Index Only Scan on `nut_data`, with a cost of 5843 and a runtime of 6.451 ms.
- The Sequential Scan on `weight` processed 13,009 rows, the largest number of rows.

After Index Creation:

- Switched to an Index Scan on `nut_data`, maintaining the cost at 5843 but increasing the runtime to 7.194 ms.
- The Index Scan processed 13,009 rows.

Reason for the Observed Behavior:

- **Before Index**: The Sequential Scan on `weight` and nested loops resulted in fewer rows being processed but led to slower runtime for individual operations due to lack of optimization.
- **After Index**: The indexed condition improved access patterns for `weight.ndb_no`, but the runtime increased slightly due to index overhead and the persistent bottleneck in the join operations.

Query 7

Physical Plan Before:

	QUERY PLAN text
1	Hash Join (cost=6216.71..6799.87 rows=18172 width=20) (actual time=31.769..36.245 rows=28060 loops=1)
2	Hash Cond: (w.ndb_no = nd.ndb_no)
3	-> Seq Scan on weight w (cost=0.00..255.09 rows=13009 width=14) (actual time=0.006..0.625 rows=13009 loops=1)
4	-> Hash (cost=6100.06..6100.06 rows=9332 width=18) (actual time=31.745..31.750 rows=14486 loops=1)
5	Buckets: 16384 Batches: 1 Memory Usage: 851kB
6	-> Hash Join (cost=3.76..6100.06 rows=9332 width=18) (actual time=0.021..29.373 rows=14486 loops=1)
7	Hash Cond: (nd.nutr_no = nutdef.nutr_no)
8	-> Seq Scan on nut_data nd (cost=0.00..5409.25 rows=253825 width=10) (actual time=0.002..11.884 rows=253825 loops=1)
9	-> Hash (cost=3.70..3.70 rows=5 width=16) (actual time=0.015..0.017 rows=5 loops=1)
10	Buckets: 1024 Batches: 1 Memory Usage: 9kB
11	-> Seq Scan on nutr_def nutdef (cost=0.00..3.70 rows=5 width=16) (actual time=0.006..0.013 rows=5 loops=1)
12	Filter: (tagname ~~ 'CA%'::text)
13	Rows Removed by Filter: 131
14	Planning Time: 0.475 ms
15	Execution Time: 36.915 ms

Highest Cost: The `seq scan on nut_data` at line 8 ,`cost= 5410`
Slowest Runtime:The `Hash Join` at line 6 ,`actual time=17.5ms`
Largest Number of Rows:The `Seq Scan on nut_data` at line 8 , `253,825` rows

Physical Plan After:
Turned off seq scan

CREATE INDEX idx_weight_ndb_no ON weight(ndb_no);
CREATE INDEX idx_nut_data_nutr_no ON nut_data(nutr_no);

Indexes: `ndb_no` on `weight`, `nutr_no` on `nut_data` (B-Tree).

Justification: Optimizes join conditions on `ndb_no` and `nutr_no`.

	QUERY PLAN text
1	Hash Join (cost=637.38..8957.03 rows=18172 width=20) (actual time=3.566..18.357 rows=28060 loops=1)
2	Hash Cond: (nd.ndb_no = w.ndb_no)
3	-> Nested Loop (cost=23.35..7986.31 rows=9332 width=18) (actual time=0.491..11.015 rows=14486 loops=1)
4	-> Index Scan using nutr_def_pkey on nutr_def nutdef (cost=0.14..15.52 rows=5 width=16) (actual time=0.007..0.045 rows=5 loops=1)
5	Filter: (tagname ~~ 'CA%'::text)
6	Rows Removed by Filter: 131
7	-> Bitmap Heap Scan on nut_data nd (cost=23.21..1575.08 rows=1908 width=10) (actual time=0.472..1.970 rows=2897 loops=5)
8	Recheck Cond: (nutr_no = nutdef.nutr_no)
9	Heap Blocks: exact=8953
10	-> Bitmap Index Scan on idx_nut_data_nutr_no (cost=0.00..22.73 rows=1908 width=0) (actual time=0.268..0.268 rows=2897 loops=5)
11	Index Cond: (nutr_no = nutdef.nutr_no)
12	-> Hash (cost=451.42..451.42 rows=13009 width=14) (actual time=3.065..3.065 rows=13009 loops=1)
13	Buckets: 16384 Batches: 1 Memory Usage: 713kB
14	-> Index Scan using idx_weight_ndb_no on weight w (cost=0.29..451.42 rows=13009 width=14) (actual time=0.017..1.797 rows=13009 loops=1)
15	Planning Time: 0.489 ms
16	Execution Time: 19.058 ms

Highest Cost: The `nested loop` at line 3 ,`cost= 6400`

Slowest Runtime:The `Bitmap Heap Join` at line 7 ,`actual time=8.51ms`

Largest Number of Rows:The `Hash Join` at line 1 , `28,060` rows

Before Index Creation:

- Used **Sequential Scan** on `nut_data`, with a **high cost of 5410** and a **Hash Join runtime of 17.5 ms**.
- The **Sequential Scan** on `nut_data` processed the **largest number of rows (253,825)**.

After Index Creation:

- Switched to **Nested Loop** and **Bitmap Heap Join**, increasing the **cost to 6400** but reducing the **runtime to 8.51 ms**.
- The **Hash Join** processed a reduced dataset of **28,060 rows**, improving efficiency.

Reason for the Observed Behavior:

- **Before Index**: Sequential Scans processed all rows, leading to high costs and slower runtime due to lack of filtering.

- **After Index**: The indexed conditions allowed faster lookups and reduced the number of rows processed, significantly lowering the runtime while slightly increasing cost due to index overhead.

Query 8

Physical Plan Before:

	QUERY PLAN text
1	Aggregate (cost=238.24..238.25 rows=1 width=8) (actual time=0.792..0.793 rows=1 loops=1)
2	-> Seq Scan on food_des (cost=0.00..233.46 rows=1911 width=0) (actual time=0.020..0.728 rows=1911 loops=1)
3	Filter: (fat_factor IS NULL)
4	Rows Removed by Filter: 5235
5	Planning Time: 0.048 ms
6	Execution Time: 0.816 ms

Highest Cost: The `Seq Scan on food_des` at line 2,`cost=233`
Slowest Runtime:The `Seq Scan on food_des` at line 2,`actual time=0.728ms`
Largest Number of Rows:The `Seq Scan on food_des` at line 2 , `1911` rows

Physical Plan After:
CREATE INDEX idx_food_des_fat_factor_null ON food_des(fat_factor) WHERE fat_factor IS NULL;
Index: Partial index on `fat_factor` (B-Tree).
Justification: Speeds up filtering on `fat_factor IS NULL`.

	QUERY PLAN text
1	Aggregate (cost=65.72..65.73 rows=1 width=8) (actual time=0.216..0.217 rows=1 loops=1)
2	-> Index Only Scan using idx_food_des_fat_factor_null on food_des (cost=0.28..60.94 rows=1911 width=0) (actual time=0.023..0.152 rows=1911 loops=1)
3	Heap Fetches: 0
4	Planning Time: 0.155 ms
5	Execution Time: 0.235 ms

Highest Cost: The `Index only scan` at line 2 ,`cost=60.9`
Slowest Runtime:The `Index only scan` at line 2 ,`actual time=0.152`
Largest Number of Rows:The `Index only scan` at line 2 , `1911` rows

Before Index Creation:

- Used **Sequential Scan** on `food_des`, with a **high cost of 233** and a **runtime of 0.728 ms**.
- The **Sequential Scan** processed **1911 rows**, the largest number of rows.

After Index Creation:

- Switched to **Index Only Scan** on `food_des`, reducing the **cost to 60.9** and the **runtime to 0.152 ms**.
- The **Index Only Scan** still processed **1911 rows**, matching the rows filtered in the sequential scan.

Reason for the Observed Behavior:

- **Before Index**: Sequential Scan had to read the entire table and apply the filter, leading to higher costs and slower runtime.
- **After Index**: The index allowed for faster retrieval of rows where `fat_factor IS NULL`, significantly reducing the cost and runtime while processing the same number of rows.

Query 9

Physical Plan Before:

	QUERY PLAN text
1	Seq Scan on food_des (cost=0.00..233.46 rows=1911 width=151) (actual time=0.022..1.117 rows=1911 loops=1)
2	Filter: (fat_factor IS NULL)
3	Rows Removed by Filter: 5235
4	Planning Time: 2.277 ms
5	Execution Time: 1.214 ms

Highest Cost: The `Seq Scan` at line 1 ,`cost=233.46`
Slowest Runtime:The `Seq Scan` at line 1 ,`actual time=1.117`
Largest Number of Rows: all rows initially processed by the **Sequential Scan** before applying the filter, which is **5235 + 1911 = 7146 rows**.
Physical Plan After:
CREATE INDEX idx9 ON food_des (fat_factor);
Index: `fat_factor` on `food_des` (B-Tree).
Justification: Improves filtering on `fat_factor`.

	QUERY PLAN text
1	Bitmap Heap Scan on food_des (cost=39.09..220.20 rows=1911 width=151) (actual time=0.137..0.427 rows=1911 loops=1)
2	Recheck Cond: (fat_factor IS NULL)
3	Heap Blocks: exact=102
4	-> Bitmap Index Scan on idx9 (cost=0.00..38.61 rows=1911 width=0) (actual time=0.124..0.124 rows=1911 loops=1)
5	Index Cond: (fat_factor IS NULL)
6	Planning Time: 0.197 ms
7	Execution Time: 0.501 ms

Highest Cost: The **Bitmap Heap Scan** at line 1 ,`cost=182`
Slowest Runtime:The **Bitmap Heap Scan** at line 1 ,`actual time=0.303`
Largest Number of Rows: :The **Bitmap Heap Scan** at line 1 and **Bitmap Index Scan** at line 4, `1911` rows

Physical Plan Before:

- Used a **Sequential Scan**, resulting in high cost (**233.46**) and slower runtime (**1.117 ms**).
- Scanned **7146 rows**, with **5235 rows** removed by the filter.

Physical Plan After:

- Switched to a **Bitmap Heap Scan** with an index on `fat_factor`, reducing cost (**182**) and improving runtime (**0.303 ms**).
- Processed **1911 rows** efficiently using the index.

Reason for the Observed Behavior:

- The creation of the index on `fat_factor` enabled the database to quickly locate rows where `fat_factor IS NULL`, avoiding the need to scan the entire table. This significantly reduced the number of operations needed to evaluate the query, leading to a lower cost and faster runtime.

Query 10

Physical Plan Before:

	QUERY PLAN text
1	Aggregate (cost=238.24..238.25 rows=1 width=8) (actual time=1.268..1.269 rows=1 loops=1)
2	-> Seq Scan on food_des (cost=0.00..233.46 rows=1911 width=8) (actual time=0.017..1.158 rows=1911 loops=1)
3	Filter: (fat_factor IS NULL)
4	Rows Removed by Filter: 5235
5	Planning Time: 0.154 ms
6	Execution Time: 1.293 ms

Highest Cost: The `Seq Scan` at line 2 ,`cost=233`
Slowest Runtime:The `Seq Scan` at line 2,`actual time=1.16`
Largest Number of Rows:all rows initially processed by the **Sequential Scan** before applying the filter, which is **5235 + 1911 = 7146 rows**.

Physical Plan After:
CREATE INDEX idx10 ON food_des (fat_factor)
Index: `fat_factor` on `food_des` (B-Tree).
Justification: Optimizes filtering for `fat_factor IS NULL`.

	QUERY PLAN text
1	Aggregate (cost=224.98..224.99 rows=1 width=8) (actual time=0.406..0.406 rows=1 loops=1)
2	-> Bitmap Heap Scan on food_des (cost=39.09..220.20 rows=1911 width=8) (actual time=0.066..0.246 rows=1911 loops=1)
3	Recheck Cond: (fat_factor IS NULL)
4	Heap Blocks: exact=102
5	-> Bitmap Index Scan on idx10 (cost=0.00..38.61 rows=1911 width=0) (actual time=0.055..0.056 rows=1911 loops=1)
6	Index Cond: (fat_factor IS NULL)
7	Planning Time: 0.135 ms
8	Execution Time: 0.426 ms

Highest Cost: The `Bitmap heap scan` at line 2 ,`cost=182`
Slowest Runtime:The `Bitmap heap scan` at line 2 ,`actual time=0.19ms`
Largest Number of Rows:The **Bitmap Heap Scan** at line 2 and **Bitmap Index Scan** at line 5, `1911` rows.

Before Index Creation:

- **Used Sequential Scan on `food_des`**, with a **high cost of 233** and an execution time of **1.16 ms**.
- The Sequential Scan initially processed **7146 rows** (1911 kept + 5235 removed by the filter).

After Index Creation:

- **Switched to a Bitmap Heap Scan** with a Bitmap Index Scan, reducing the cost to **182** and significantly decreasing the execution time to **0.19 ms**.

- Both the Bitmap Heap Scan and Bitmap Index Scan processed **1911 rows**, matching the filter condition.

Reason for the Observed Behavior:

- **Before Index**: The Sequential Scan processed all rows, leading to a higher cost and execution time as the filter (`fat_factor IS NULL`) had to be applied across the entire dataset.
- **After Index**: The Bitmap operations leveraged the newly created index (`idx10`) to efficiently locate rows matching the filter condition. This optimization reduced the processing cost and execution time.

Query 11

Physical Plan Before:

	QUERY PLAN text
1	Aggregate (cost=238.24..238.25 rows=1 width=8) (actual time=0.738..0.739 rows=1 loops=1)
2	-> Seq Scan on food_des (cost=0.00..233.46 rows=1911 width=8) (actual time=0.012..0.675 rows=1911 loops=1)
3	Filter: (fat_factor IS NULL)
4	Rows Removed by Filter: 5235
5	Planning Time: 0.147 ms
6	Execution Time: 0.754 ms

Highest Cost: The `Seq Scan` at line 2 ,`cost=233`
Slowest Runtime:The `Seq Scan` at line 2,`actual time=0.675`
Largest Number of Rows:all rows initially processed by the **Sequential Scan** before applying the filter, which is **5235 + 1911 = 7146 rows**.

Physical Plan After:
CREATE INDEX idx11 ON food_des (fat_factor) WHERE fat_factor IS NULL;
Index: Partial index on `fat_factor` (B-Tree).
Justification: Filters rows for `fat_factor IS NULL`.

	QUERY PLAN text
1	Aggregate (cost=228.20..228.21 rows=1 width=8) (actual time=0.607..0.608 rows=1 loops=1)
2	-> Bitmap Heap Scan on food_des (cost=42.31..223.42 rows=1911 width=8) (actual time=0.105..0.367 rows=1911 loops=1)
3	Recheck Cond: (fat_factor IS NULL)
4	Heap Blocks: exact=102
5	-> Bitmap Index Scan on idx11 (cost=0.00..41.83 rows=1911 width=0) (actual time=0.055..0.055 rows=1911 loops=1)
6	Planning Time: 0.134 ms
7	Execution Time: 0.629 ms

Highest Cost: The `Bitmap heap scan` at line 2 ,`cost=182`
Slowest Runtime:The `Bitmap heap scan` at line 2 ,`actual time=0.312ms`
Largest Number of Rows:The **Bitmap Heap Scan** at line 2 and **Bitmap Index Scan** at line 5, `1911` rows.

Before Index Creation:

- **Used Sequential Scan on `food_des`**, with a **high cost of 233** and an execution time of **0.675 ms**.
- The Sequential Scan initially processed **7146 rows** (1911 kept + 5235 removed by the filter).

After Index Creation:

- **Switched to a Bitmap Heap Scan** with a Bitmap Index Scan, reducing the cost to **182** and lowering the execution time to **0.312 ms**.
- Both the Bitmap Heap Scan and Bitmap Index Scan processed **1911 rows**, matching the filter condition.

Reason for the Observed Behavior:

- **Before Index**: The Sequential Scan processed all rows, leading to higher costs and execution times because the filter (`fat_factor IS NULL`) was applied across the entire dataset.
- **After Index**: The Bitmap operations efficiently utilized the newly created partial index (`idx11`), focusing only on rows matching the filter condition. This significantly reduced the processing cost and runtime.

Query 12

Physical Plan Before:

	QUERY PLAN text
1	Aggregate (cost=246.55..246.56 rows=1 width=8) (actual time=1.351..1.352 rows=1 loops=1)
2	-> Seq Scan on food_des (cost=0.00..233.46 rows=5235 width=8) (actual time=0.007..1.099 rows=5235 loops=1)
3	Filter: (fat_factor IS NOT NULL)
4	Rows Removed by Filter: 1911
5	Planning Time: 0.127 ms
6	Execution Time: 1.380 ms

Highest Cost: The `Seq Scan on food_des` at line 2 ,`cost=233`
Slowest Runtime:The `Seq Scan on food_des` at line 2 ,`actual time=1.1ms`

Largest Number of Rows:The `Seq Scan on food_des` at line 2, scanned rows=1911+5235=7146
Physical Plan After:
CREATE INDEX idx12 ON food_des (fat_factor)
Index: `fat_factor` on `food_des` (B-Tree).
Justification: Facilitates filtering on `fat_factor IS NOT NULL`.
Turned seq scan off

	QUERY PLAN text
1	Aggregate (cost=336.29..336.30 rows=1 width=8) (actual time=1.178..1.179 rows=1 loops=1)
2	-> Bitmap Heap Scan on food_des (cost=108.85..323.20 rows=5235 width=8) (actual time=0.210..0.728 rows=5235 loops=1)
3	Recheck Cond: (fat_factor IS NOT NULL)
4	Heap Blocks: exact=149
5	-> Bitmap Index Scan on idx12 (cost=0.00..107.54 rows=5235 width=0) (actual time=0.194..0.194 rows=5235 loops=1)
6	Index Cond: (fat_factor IS NOT NULL)
7	Planning Time: 0.195 ms
8	Execution Time: 1.204 ms

Highest Cost: The `Bitmap heap scan on food_des` at line 2 ,`cost=216`
Slowest Runtime:The `Bitmap heap scan on food_des` at line 2 ,`actual time=0.534ms`
Largest Number of Rows:The `Bitmap heap scan on food_des` at line 2 and `Bitmap index scan`, rows=5235

Before Index Creation:

- **Used Sequential Scan on `food_des`**, with a **high cost of 233** and an execution time of **1.1 ms**.
- The Sequential Scan initially processed **7146 rows** (5235 kept + 1911 removed by the filter).

After Index Creation:

- **Switched to a Bitmap Heap Scan** with a Bitmap Index Scan, reducing the cost to **216** and lowering the execution time to **0.534 ms**.
- Both the Bitmap Heap Scan and Bitmap Index Scan processed **5235 rows**, matching the filter condition.

Reason for the Observed Behavior:

- **Before Index**: The Sequential Scan processed all rows, resulting in higher costs and execution times as the filter (`fat_factor IS NOT NULL`) was applied across the entire dataset.

- **After Index**: The Bitmap operations leveraged the newly created index (`idx12`) to efficiently locate rows matching the filter condition. This reduced the processing cost and runtime significantly.

Query 13

Physical Plan Before:

	QUERY PLAN text
1	Hash Join (cost=17.65..1803.98 rows=256 width=5) (actual time=0.078..10.843 rows=27 loops=1)
2	Hash Cond: (dsl.nutr_no = nutdef.nutr_no)
3	-> Hash Join (cost=12.59..1798.23 rows=256 width=4) (actual time=0.047..10.804 rows=27 loops=1)
4	Hash Cond: (dsl.datasrc_id = ds.datasrc_id)
5	-> Seq Scan on datsrcln dsl (cost=0.00..1536.45 rows=93845 width=11) (actual time=0.002..3.961 rows=93845 loops=1)
6	-> Hash (cost=12.57..12.57 rows=1 width=7) (actual time=0.029..0.030 rows=1 loops=1)
7	Buckets: 1024 Batches: 1 Memory Usage: 9kB
8	-> Seq Scan on data_src ds (cost=0.00..12.57 rows=1 width=7) (actual time=0.021..0.028 rows=1 loops=1)
9	Filter: (vol_city = 'Cincinnati'::text)
10	Rows Removed by Filter: 365
11	-> Hash (cost=3.36..3.36 rows=136 width=9) (actual time=0.028..0.028 rows=136 loops=1)
12	Buckets: 1024 Batches: 1 Memory Usage: 14kB
13	-> Seq Scan on nutr_def nutdef (cost=0.00..3.36 rows=136 width=9) (actual time=0.007..0.016 rows=136 loops=1)
14	Planning Time: 0.264 ms
15	Execution Time: 10.869 ms

Highest Cost: The `Seq Scan on datsrcln` at line 5 ,`cost=1540`
Slowest Runtime:The `Hash Joins` at line 3 ,`actual time=6.81ms`
Largest Number of Rows:The `Seq Scan on datsrcln` at line 5, rows=93,845
Physical Plan After:
CREATE INDEX idx13 ON datsrcln(datasrc_id);
Index: `datasrc_id` on `datsrcln` (B-Tree).
Justification: Speeds up joins on `datasrc_id`.

	QUERY PLAN text
1	Hash Join (cost=11.69..537.91 rows=256 width=5) (actual time=0.152..0.203 rows=27 loops=1)
2	Hash Cond: (dsl.nutr_no = nutdef.nutr_no)
3	-> Nested Loop (cost=6.63..532.16 rows=256 width=4) (actual time=0.087..0.132 rows=27 loops=1)
4	-> Seq Scan on data_src ds (cost=0.00..12.57 rows=1 width=7) (actual time=0.060..0.072 rows=1 loops=1)
5	Filter: (vol_city = 'Cincinnati'::text)
6	Rows Removed by Filter: 365
7	-> Bitmap Heap Scan on datsrcln dsl (cost=6.63..516.57 rows=302 width=11) (actual time=0.017..0.048 rows=27 loops=1)
8	Recheck Cond: (datasrc_id = ds.datasrc_id)
9	Heap Blocks: exact=20
10	-> Bitmap Index Scan on idx13 (cost=0.00..6.56 rows=302 width=0) (actual time=0.011..0.011 rows=27 loops=1)
11	Index Cond: (datasrc_id = ds.datasrc_id)
12	-> Hash (cost=3.36..3.36 rows=136 width=9) (actual time=0.031..0.032 rows=136 loops=1)
13	Buckets: 1024 Batches: 1 Memory Usage: 14kB
14	-> Seq Scan on nutr_def nutdef (cost=0.00..3.36 rows=136 width=9) (actual time=0.006..0.016 rows=136 loops=1)
15	Planning Time: 0.405 ms
16	Execution Time: 0.235 ms

Highest Cost: The `Bitmap Heap scan` at line 7 ,`cost=510`
Slowest Runtime:The `Seq scan on data_src` at line 4 ,`actual time=0.072`
Largest Number of Rows::The `Seq scan on data_src` at line 4, scanned rows=365+1=366

Before Index Creation:

- Used Hash Join between `nutr_def`, `datsrcln`, and `data_src`, with the highest cost being the Sequential Scan on `datsrcln` at line 5, costing 1540.
- The Hash Join at line 3 had the slowest execution time, taking 6.81 ms.
- The Sequential Scan on `datsrcln` initially processed 93,845 rows.

After Index Creation:

- Switched to a Bitmap Heap Scan with a Bitmap Index Scan on `datsrcln`, reducing the highest cost to 510 and significantly lowering the slowest execution time to 0.072 ms.
- The Bitmap Heap Scan and Bitmap Index Scan processed 366 rows (365 kept + 1 removed by the filter).

Reason for the Observed Behavior:

- **Before Index**: The Sequential Scan on `datsrcln` caused higher costs and execution times as all rows (93,845) were processed without any index optimization.

- **After Index**: The Bitmap operations leveraged the newly created index (`idx13`) to efficiently filter rows based on `datasrc_id`, reducing the processing cost and execution time.

Query 14

Physical Plan Before:

	QUERY PLAN text
1	Seq Scan on weight (cost=0.00..287.61 rows=11624 width=50) (actual time=0.008..1.001 rows=11625 loops=1)
2	Filter: (amount < '2'::double precision)
3	Rows Removed by Filter: 1384
4	Planning Time: 0.111 ms
5	Execution Time: 1.400 ms

Highest Cost: The `Seq scan` at line 1 ,`cost=288`

Slowest Runtime:The `Seq scan` at line 1,`actual time=1.001ms`

Largest Number of Rows::The `Seq scan` at line 1, scanned rows= 1384+11625=13009

Physical Plan After:

CREATE INDEX idx14 ON weight (amount) WHERE amount <2;

Index: Partial index on `amount` (B-Tree).

Justification: Improves filtering for `amount < 2`.

Turned off Seq Scan

	QUERY PLAN text
1	Bitmap Heap Scan on weight (cost=197.31..467.61 rows=11624 width=50) (actual time=0.254..0.981 rows=11625 loops=1)
2	Recheck Cond: (amount < '2'::double precision)
3	Heap Blocks: exact=125
4	-> Bitmap Index Scan on idx14 (cost=0.00..194.41 rows=11624 width=0) (actual time=0.237..0.238 rows=11625 loops=1)
5	Planning Time: 0.079 ms
6	Execution Time: 1.287 ms

Highest Cost: The `Bitmap Heap scan` at line 1 ,`cost=273`

Slowest Runtime:The `Bitmap Heap scan` at line 1,`actual time=0.743ms`

Largest Number of Rows:The `Bitmap Heap scan` at line 1 and The `Bitmap Index scan` at line 4, rows= 11624

Before Index Creation:

- Used Sequential Scan on `weight`, with a high cost of 288 and an execution time of 1.001 ms.
- The Sequential Scan initially processed 13,009 rows (11,625 kept + 1,384 removed by the filter).

After Index Creation:

- Switched to a Bitmap Heap Scan with a Bitmap Index Scan, reducing the cost to 273 and lowering the execution time to 0.743 ms.
- Both the Bitmap Heap Scan and Bitmap Index Scan processed 11,624 rows.

Reason for the Observed Behavior:

- **Before Index**: The Sequential Scan processed all rows, resulting in higher costs and execution times as the filter (`amount < 2`) was applied across the entire dataset.
- **After Index**: The Bitmap operations leveraged the newly created index (`idx14`) to efficiently locate rows matching the filter condition. This significantly reduced processing costs and runtime.

Query 15

Physical Plan Before:

	QUERY PLAN text
1	Seq Scan on weight (cost=0.00..287.61 rows=1215 width=50) (actual time=0.009..0.736 rows=1215 loops=1)
2	Filter: (amount > '2'::double precision)
3	Rows Removed by Filter: 11794
4	Planning Time: 0.163 ms
5	Execution Time: 0.779 ms

Highest Cost: The `Seq Scan` at line 1 ,`cost=288`
Slowest Runtime:The `Seq Scan` at line 1,`actual time=0.736ms`
Largest Number of Rows:The `Seq Scan` at line 1, rows scanned :11794+1215=13009 rows

Physical Plan After:
CREATE INDEX idx15 ON weight(amount);
Index: `amount` on `weight` (B-Tree).
Justification: Optimizes filtering for `amount > 2`.

	QUERY PLAN text
1	Bitmap Heap Scan on weight (cost=25.70..165.89 rows=1215 width=50) (actual time=0.061..0.241 rows=1215 loops=1)
2	Recheck Cond: (amount > '2'::double precision)
3	Heap Blocks: exact=86
4	-> Bitmap Index Scan on idx15 (cost=0.00..25.40 rows=1215 width=0) (actual time=0.051..0.051 rows=1215 loops=1)
5	Index Cond: (amount > '2'::double precision)
6	Planning Time: 0.255 ms
7	Execution Time: 0.282 ms

Highest Cost: The `Bitmap Heap Scan` at line 1 ,`cost=140`
Slowest Runtime:The `Bitmap Heap Scan` at line 1,`actual time=0.19`
Largest Number of Rows:The `Bitmap index Scan` at line 4 and `Bitmap Heap Scan`, rows=1215

Before Index Creation:

- Used Sequential Scan on `weight`, with a high cost of 288 and an execution time of 0.736 ms.
- The Sequential Scan initially processed 13,009 rows (1,215 kept + 11,794 removed by the filter).

After Index Creation:

- Switched to a Bitmap Heap Scan with a Bitmap Index Scan, reducing the cost to 140 and lowering the execution time to 0.19 ms.
- Both the Bitmap Heap Scan and Bitmap Index Scan processed 1,215 rows.

Reason for the Observed Behavior

- **Before Index**: The Sequential Scan processed all rows, resulting in higher costs and execution times as the filter (`amount > 2`) was applied across the entire dataset.
- **After Index**: The Bitmap operations leveraged the newly created index (`idx15`) to efficiently locate rows matching the filter condition. This significantly reduced processing costs and runtime.

Query 16

Physical Plan Before:

	QUERY PLAN text
1	HashAggregate (cost=296.73..307.56 rows=867 width=29) (actual time=0.917..0.948 rows=140 loops=1)
2	Group Key: amount, msre_desc
3	Batches: 1 Memory Usage: 73kB
4	-> Seq Scan on weight (cost=0.00..287.61 rows=1215 width=21) (actual time=0.007..0.684 rows=1215 loops=1)
5	Filter: (amount > '2'::double precision)
6	Rows Removed by Filter: 11794
7	Planning Time: 0.116 ms
8	Execution Time: 0.985 ms

Highest Cost: The `Seq Scan on weight` at line 4 ,`cost=288`
Slowest Runtime:The `Seq Scan on weight` at line 4,`actual time=0.684`
Largest Number of Rows:The `Seq Scan on weight` at line 4, rows scanned :11794+1215=13009 rows
Physical Plan After:
CREATE INDEX idx16 ON weight(amount, msre_desc);
Index: Composite index on `amount` and `msre_desc` (B-Tree).
Justification: Facilitates grouping by `amount` and `msre_desc`.

	QUERY PLAN text
1	GroupAggregate (cost=0.29..69.50 rows=867 width=29) (actual time=0.015..0.204 rows=140 loops=1)
2	Group Key: amount, msre_desc
3	-> Index Only Scan using idx16 on weight (cost=0.29..49.55 rows=1215 width=21) (actual time=0.010..0.092 rows=1215 loops=1)
4	Index Cond: (amount > '2'::double precision)
5	Heap Fetches: 0
6	Planning Time: 0.091 ms
7	Execution Time: 0.228 ms

Highest Cost: The `Index only Scan` at line 3,`cost=49.5`
Slowest Runtime:The `GroupAggregate` at line 1,`actual time=0.112ms`
Largest Number of Rows:The `Index only Scan` at line 3, rows=1215

Before Index Creation:

- Used Sequential Scan on `weight`, with a high cost of 288 and an execution time of 0.684 ms.
- The Sequential Scan initially processed 13,009 rows (1,215 kept + 11,794 removed by the filter).

After Index Creation:

- Switched to an Index Only Scan, reducing the cost to 49.5 and lowering the execution time to 0.112 ms.
- The Index Only Scan processed 1,215 rows.

Reason for the Observed Behavior:

- **Before Index**: The Sequential Scan processed all rows, resulting in higher costs and execution times because the filter (`amount > 2`) and grouping were applied across the entire dataset.
- **After Index**: The Index Only Scan utilized the newly created index (`idx16`) to efficiently locate rows matching the filter and group by keys (`amount, msre_desc`), significantly reducing processing costs and runtime.

Query 17

Physical Plan Before:

	QUERY PLAN text
1	Group (cost=11852.36..11982.11 rows=1765 width=58) (actual time=48.905..51.019 rows=1 loops=1)
2	Group Key: fd.long_desc, nd.num_studies
3	InitPlan 1 (returns $1)
4	-> Finalize Aggregate (cost=5737.47..5737.48 rows=1 width=4) (actual time=29.914..30.057 rows=1 loops=1)
5	-> Gather (cost=5737.36..5737.47 rows=1 width=4) (actual time=28.741..30.050 rows=2 loops=1)
6	Workers Planned: 1
7	Workers Launched: 1
8	-> Partial Aggregate (cost=4737.36..4737.37 rows=1 width=4) (actual time=17.670..17.671 rows=1 loops=2)
9	-> Parallel Seq Scan on nut_data (cost=0.00..4364.09 rows=149309 width=4) (actual time=0.005..8.630 rows=126912 loops=2)
10	-> Gather Merge (cost=6114.88..6239.44 rows=1038 width=58) (actual time=48.904..50.873 rows=1 loops=1)
11	Workers Planned: 1
12	Params Evaluated: $1
13	Workers Launched: 1
14	-> Group (cost=5114.87..5122.66 rows=1038 width=58) (actual time=9.515..9.517 rows=0 loops=2)
15	Group Key: fd.long_desc, nd.num_studies
16	-> Sort (cost=5114.87..5117.47 rows=1038 width=58) (actual time=9.513..9.514 rows=0 loops=2)
17	Sort Key: fd.long_desc
18	Sort Method: quicksort Memory: 25kB
19	Worker 0: Sort Method: quicksort Memory: 25kB
20	-> Hash Join (cost=322.79..5062.87 rows=1038 width=58) (actual time=4.834..9.476 rows=0 loops=2)
21	Hash Cond: (nd.ndb_no = fd.ndb_no)
22	-> Parallel Seq Scan on nut_data nd (cost=0.00..4737.36 rows=1038 width=10) (actual time=4.101..8.741 rows=0 loops=2)
23	Filter: (num_studies = $1)
24	Rows Removed by Filter: 126912
25	-> Hash (cost=233.46..233.46 rows=7146 width=60) (actual time=1.444..1.445 rows=7146 loops=1)
26	Buckets: 8192 Batches: 1 Memory Usage: 708kB
27	-> Seq Scan on food_des fd (cost=0.00..233.46 rows=7146 width=60) (actual time=0.021..0.606 rows=7146 loops=1)
28	Planning Time: 0.260 ms
29	Execution Time: 51.106 ms

Highest Cost: The Group at line 1, `cost=12,000`

Slowest Runtime:The Group at line 1, `actual time=51ms`

Largest Number of Rows:The `parallel seq scan on nut_data` at line 9, `28,060` rows

Physical Plan After:

CREATE INDEX idx17 ON nut_data(num_studies);

Index: `num_studies` on `nut_data` (B-Tree).

Justification: Optimizes grouping and filtering on `num_studies`.

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	HashAggregate (cost=3004.58..3022.23 rows=1765 width=58) (actual time=1.481..1.486 rows=1 loops=1)
2	Group Key: fd.long_desc, nd.num_studies
3	Batches: 1 Memory Usage: 73kB
4	InitPlan 2 (returns $1)
5	-> Result (cost=0.44..0.45 rows=1 width=4) (actual time=0.008..0.009 rows=1 loops=1)
6	InitPlan 1 (returns $0)
7	-> Limit (cost=0.42..0.44 rows=1 width=4) (actual time=0.007..0.007 rows=1 loops=1)
8	-> Index Only Scan Backward using idx17 on nut_data (cost=0.42..555.85 rows=26482 width=4) (actual time=0.006..0.006 rows=1 loops=1)
9	Index Cond: (num_studies IS NOT NULL)
10	Heap Fetches: 0
11	-> Hash Join (cost=344.88..2995.30 rows=1765 width=58) (actual time=1.473..1.476 rows=1 loops=1)
12	Hash Cond: (nd.ndb_no = fd.ndb_no)
13	-> Bitmap Heap Scan on nut_data nd (cost=22.10..2667.89 rows=1765 width=10) (actual time=0.015..0.017 rows=1 loops=1)
14	Recheck Cond: (num_studies = $1)
15	Heap Blocks: exact=1
16	-> Bitmap Index Scan on idx17 (cost=0.00..21.66 rows=1765 width=0) (actual time=0.012..0.012 rows=1 loops=1)
17	Index Cond: (num_studies = $1)
18	-> Hash (cost=233.46..233.46 rows=7146 width=60) (actual time=1.448..1.448 rows=7146 loops=1)
19	Buckets: 8192 Batches: 1 Memory Usage: 708kB
20	-> Seq Scan on food_des fd (cost=0.00..233.46 rows=7146 width=60) (actual time=0.004..0.607 rows=7146 loops=1)
21	Planning Time: 0.216 ms
22	Execution Time: 1.529 ms

Highest Cost: The bitmap heap scan at line 13, `cost=2,650`
Slowest Runtime:The hash at line 18, `actual time=0.841ms`
Largest Number of Rows:The seq scan at line 20 and hash at line 18, `7146 rows`

Before Index Creation:

- Used **Group** operation with a **Parallel Sequential Scan**, resulting in a **high cost (12,000)** and **execution time (51 ms)**.
- The **Parallel Sequential Scan** on `nut_data` processed the **largest number of rows (28,060)**.

After Index Creation:

- Switched to a **Bitmap Heap Scan** with an index on `nut_data(num_studies)`, reducing the **cost (2,650)** significantly and improving **runtime (0.841 ms)**.
- The **Hash** and **Sequential Scan** processed the **largest number of rows (7146)**, focused by the index.

Reason for the Observed Behavior:

- **Before Index**: The Group and Parallel Scan processed a large volume of rows, leading to high cost and slow runtime due to the absence of filtering through an index.
- **After Index**: The index on `num_studies` allowed efficient filtering, significantly reducing the number of rows scanned and improving query performance.

Query 18

Physical Plan Before:

	QUERY PLAN text
1	Hash Join (cost=1.54..257.07 rows=7146 width=175) (actual time=0.022..1.938 rows=7146 loops=1)
2	Hash Cond: (fod.fdgrp_cd = fd.fdgrp_cd)
3	-> Seq Scan on food_des fod (cost=0.00..233.46 rows=7146 width=151) (actual time=0.007..0.405 rows=7146 loops=1)
4	-> Hash (cost=1.24..1.24 rows=24 width=24) (actual time=0.008..0.009 rows=24 loops=1)
5	Buckets: 1024 Batches: 1 Memory Usage: 10kB
6	-> Seq Scan on fd_group fd (cost=0.00..1.24 rows=24 width=24) (actual time=0.002..0.003 rows=24 loops=1)
7	Planning Time: 0.164 ms
8	Execution Time: 2.095 ms

Highest Cost: The Seq scan on food_des at line 3, `cost=233`

Slowest Runtime:The hash join at line 1, `timing=1.52 ms`

Largest Number of Rows: The Seq scan on food_des at line 3 and The hash join at line 1, rows=7146

Physical Plan After:

Turned off seq scan

CREATE INDEX idx18 ON food_des(fdgrp_cd);

Index: `fdgrp_cd` on `food_des` (B-Tree).

Justification: Accelerates joins on `fdgrp_cd`.

	QUERY PLAN text
1	Hash Join (cost=13.08..455.32 rows=7146 width=175) (actual time=0.021..2.130 rows=7146 loops=1)
2	Hash Cond: (fod.fdgrp_cd = fd.fdgrp_cd)
3	-> Index Scan using idx18 on food_des fod (cost=0.28..420.45 rows=7146 width=151) (actual time=0.008..0.719 rows=7146 loops=1)
4	-> Hash (cost=12.50..12.50 rows=24 width=24) (actual time=0.007..0.008 rows=24 loops=1)
5	Buckets: 1024 Batches: 1 Memory Usage: 10kB
6	-> Index Scan using fd_group_pkey on fd_group fd (cost=0.14..12.50 rows=24 width=24) (actual time=0.001..0.004 rows=24 loops=1)
7	Planning Time: 0.119 ms
8	Execution Time: 2.280 ms

Highest Cost: The index scan on food_des at line 3, `cost=420`

Slowest Runtime:The hash join at line 1, `timing=1.4 ms`

Largest Number of Rows: The Seq scan on food_des at line 3 and The hash join at line 1, rows=7146

Before Index Creation:

- Used **Sequential Scan** on `food_des`, with a **cost of 233** and a **Hash Join runtime of 1.52 ms**.
- The **Sequential Scan** and **Hash Join** both processed **7146 rows**, representing the entire table.

After Index Creation:

- Switched to an **Index Scan** on `food_des`, increasing the **cost to 420** but slightly improving the **Hash Join runtime to 1.4 ms**.
- The **Index Scan** and **Hash Join** still processed the **same 7146 rows**.

Reason for the Observed Behavior:

- **Before Index**: The Sequential Scan processed all rows, leading to lower cost but less optimized filtering for the join operation.
- **After Index**: The Index Scan provided more precise data retrieval for the join but added overhead, increasing the cost slightly while reducing runtime marginally. The total rows processed remained unchanged.

Query 19

Physical Plan Before:

	QUERY PLAN text
1	Hash Join (cost=324.33..7184.21 rows=253825 width=267) (actual time=1.186..123.346 rows=253825 loops=1)
2	Hash Cond: (fod.fdgrp_cd = fd.fdgrp_cd)
3	-> Hash Join (cost=322.79..6398.73 rows=253825 width=243) (actual time=1.169..80.790 rows=253825 loops=1)
4	Hash Cond: (nd.ndb_no = fod.ndb_no)
5	-> Seq Scan on nut_data nd (cost=0.00..5409.25 rows=253825 width=92) (actual time=0.002..11.621 rows=253825 loops=1)
6	-> Hash (cost=233.46..233.46 rows=7146 width=151) (actual time=1.153..1.156 rows=7146 loops=1)
7	Buckets: 8192 Batches: 1 Memory Usage: 1364kB
8	-> Seq Scan on food_des fod (cost=0.00..233.46 rows=7146 width=151) (actual time=0.003..0.368 rows=7146 loops=1)
9	-> Hash (cost=1.24..1.24 rows=24 width=24) (actual time=0.011..0.012 rows=24 loops=1)
10	Buckets: 1024 Batches: 1 Memory Usage: 10kB
11	-> Seq Scan on fd_group fd (cost=0.00..1.24 rows=24 width=24) (actual time=0.005..0.006 rows=24 loops=1)
12	Planning Time: 0.254 ms
13	Execution Time: 128.311 ms

Highest Cost: The Seq scan on nut_data at line 5, `cost=5140`

Slowest Runtime:The hash join at line 3, `timing=68 ms`

Largest Number of Rows: The Seq scan on nut_data at line 5 and hash joins at line 1 & 3, rows=253,825

Physical Plan After:

Turned off seq scan

CREATE INDEX idx19 ON nut_data(ndb_no);

Index: `ndb_no` on `nut_data` (B-Tree).

Justification: Speeds up joins on `ndb_no`.

	QUERY PLAN text
1	Hash Join (cost=462.89..9514.90 rows=253825 width=267) (actual time=1.255..139.080 rows=253825 loops=1)
2	Hash Cond: (fod.fdgrp_cd = fd.fdgrp_cd)
3	-> Hash Join (cost=450.09..8718.16 rows=253825 width=243) (actual time=1.242..95.978 rows=253825 loops=1)
4	Hash Cond: (nd.ndb_no = fod.ndb_no)
5	-> Index Scan using idx19 on nut_data nd (cost=0.29..7601.67 rows=253825 width=92) (actual time=0.004..25.473 rows=253825 loops=1)
6	-> Hash (cost=360.47..360.47 rows=7146 width=151) (actual time=1.228..1.230 rows=7146 loops=1)
7	Buckets: 8192 Batches: 1 Memory Usage: 1364kB
8	-> Index Scan using food_des_pkey on food_des fod (cost=0.28..360.47 rows=7146 width=151) (actual time=0.003..0.652 rows=7146 loops=1)
9	-> Hash (cost=12.50..12.50 rows=24 width=24) (actual time=0.009..0.009 rows=24 loops=1)
10	Buckets: 1024 Batches: 1 Memory Usage: 10kB
11	-> Index Scan using fd_group_pkey on fd_group fd (cost=0.14..12.50 rows=24 width=24) (actual time=0.002..0.005 rows=24 loops=1)
12	Planning Time: 0.230 ms
13	Execution Time: 144.446 ms

Highest Cost: The index scan on nut_data at line 5, `cost=7600`

Slowest Runtime:The hash join at line 3, `timing=69.3 ms`

Largest Number of Rows: The index scan on nut_data at line 5 and hash joins at line 1 & 3, rows=253,825

Before Index Creation:

- Used **Sequential Scan** on `nut_data`, resulting in a **high cost (5140)** and a **Hash Join runtime of 68 ms**.
- The **Sequential Scan** and **Hash Joins** processed the **largest number of rows (253,825)**.

After Index Creation:

- Switched to an **Index Scan** on `nut_data`, increasing the **cost to 7600** but slightly increasing the **Hash Join runtime to 69.3 ms**.
- The **Index Scan** and **Hash Joins** still processed the **same 253,825 rows**.

Reason for the Observed Behavior:

- **Before Index**: The Sequential Scan accessed all rows in `nut_data`, leading to a lower cost but slower runtime for joins.
- **After Index**: The Index Scan introduced overhead for data retrieval, increasing the cost. However, the runtime remained almost the same since the same number of rows were processed during the join operation.

Query 20

Physical Plan Before:

	QUERY PLAN text
1	HashSetOp Intersect (cost=0.00..549.69 rows=1 width=28) (actual time=1.722..1.724 rows=0 loops=1)
2	-> Append (cost=0.00..548.94 rows=299 width=28) (actual time=1.362..1.713 rows=118 loops=1)
3	-> Subquery Scan on "*SELECT* 1" (cost=0.00..287.62 rows=1 width=10) (actual time=0.528..0.529 rows=0 loops=1)
4	-> Seq Scan on weight (cost=0.00..287.61 rows=1 width=6) (actual time=0.528..0.528 rows=0 loops=1)
5	Filter: (amount > '50'::double precision)
6	Rows Removed by Filter: 13009
7	-> Subquery Scan on "*SELECT* 2" (cost=1.31..259.83 rows=298 width=10) (actual time=0.833..1.173 rows=118 loops=1)
8	-> Hash Join (cost=1.31..256.85 rows=298 width=6) (actual time=0.832..1.159 rows=118 loops=1)
9	Hash Cond: (fod.fdgrp_cd = fd.fdgrp_cd)
10	-> Seq Scan on food_des fod (cost=0.00..233.46 rows=7146 width=11) (actual time=0.003..0.442 rows=7146 loops=1)
11	-> Hash (cost=1.30..1.30 rows=1 width=5) (actual time=0.006..0.007 rows=1 loops=1)
12	Buckets: 1024 Batches: 1 Memory Usage: 9kB
13	-> Seq Scan on fd_group fd (cost=0.00..1.30 rows=1 width=5) (actual time=0.004..0.004 rows=1 loops=1)
14	Filter: (fddrp_desc = 'Snacks'::text)
15	Rows Removed by Filter: 23
16	Planning Time: 0.165 ms
17	Execution Time: 1.746 ms

Highest Cost: The Seq scan on weight at line 4, `cost=288`

Slowest Runtime:The hash join at line 8, `timing=0.71 ms`

Largest Number of Rows: The Seq scan on weight at line 4 scanned:13009+0=13009 rows

Physical Plan After:

CREATE INDEX idx_weight_amount ON weight(amount) where amount >50;

Index: Partial index on `amount` (B-Tree).

Justification: Improves filtering for `amount > 50`.

	QUERY PLAN text
1	HashSetOp Intersect (cost=0.12..270.22 rows=1 width=28) (actual time=0.779..0.781 rows=0 loops=1)
2	-> Append (cost=0.12..269.47 rows=299 width=28) (actual time=0.600..0.775 rows=118 loops=1)
3	-> Subquery Scan on "*SELECT* 1" (cost=0.12..8.15 rows=1 width=10) (actual time=0.004..0.004 rows=0 loops=1)
4	-> Index Scan using idx20 on weight (cost=0.12..8.14 rows=1 width=6) (actual time=0.003..0.003 rows=0 loops=1)
5	-> Subquery Scan on "*SELECT* 2" (cost=1.31..259.83 rows=298 width=10) (actual time=0.596..0.765 rows=118 loops=1)
6	-> Hash Join (cost=1.31..256.85 rows=298 width=6) (actual time=0.595..0.759 rows=118 loops=1)
7	Hash Cond: (fod.fdgrp_cd = fd.fdgrp_cd)
8	-> Seq Scan on food_des fod (cost=0.00..233.46 rows=7146 width=11) (actual time=0.006..0.317 rows=7146 loops=1)
9	-> Hash (cost=1.30..1.30 rows=1 width=5) (actual time=0.007..0.007 rows=1 loops=1)
10	Buckets: 1024 Batches: 1 Memory Usage: 9kB
11	-> Seq Scan on fd_group fd (cost=0.00..1.30 rows=1 width=5) (actual time=0.004..0.005 rows=1 loops=1)
12	Filter: (fddrp_desc = 'Snacks'::text)
13	Rows Removed by Filter: 23
14	Planning Time: 0.199 ms
15	Execution Time: 0.808 ms

Highest Cost: The Seq scan on food_des at line 8, `cost=233`

Slowest Runtime:The hash join at line 6, `timing=0.435 ms`
Largest Number of Rows: The Seq scan on food_des at line 8, rows=7146

Before Index Creation:

- Used **Sequential Scan** on `weight`, with a **cost of 288** and a **Hash Join runtime of 0.71 ms**.
- The **Sequential Scan on `weight`** processed the **largest number of rows (13,009)** before filtering.

After Index Creation:

- Switched to an **Index Scan** on `weight` with a filtered index on `amount > 50`, reducing the **cost to 233** and improving the **Hash Join runtime to 0.435 ms**.
- The **Sequential Scan on `food_des`** still processed the **largest number of rows (7146)**.

Reason for the Observed Behavior:

- **Before Index**: Sequential Scan on `weight` processed all rows, resulting in higher cost and slower performance for rows with `amount > 50`.
- **After Index**: The filtered index on `weight(amount > 50)` efficiently reduced the rows scanned, lowering the cost and improving runtime. The overall rows processed in the join (`food_des`) remained constant.

Query 21

Physical Plan Before:

	QUERY PLAN text
1	Aggregate (cost=274.94..274.95 rows=1 width=32) (actual time=2.545..2.548 rows=1 loops=1)
2	-> Hash Join (cost=1.54..257.07 rows=7146 width=6) (actual time=0.023..1.562 rows=7146 loops=1)
3	Hash Cond: (fod.fdgrp_cd = fd.fdgrp_cd)
4	-> Seq Scan on food_des fod (cost=0.00..233.46 rows=7146 width=11) (actual time=0.005..0.400 rows=7146 loops=1)
5	-> Hash (cost=1.24..1.24 rows=24 width=5) (actual time=0.009..0.010 rows=24 loops=1)
6	Buckets: 1024 Batches: 1 Memory Usage: 9kB
7	-> Seq Scan on fd_group fd (cost=0.00..1.24 rows=24 width=5) (actual time=0.002..0.004 rows=24 loops=1)
8	Planning Time: 0.153 ms
9	Execution Time: 2.575 ms

Highest Cost: The Seq scan on food_des at line 4, `cost=233`

Slowest Runtime:The hash join at line 2, `timing=1.15 ms`
Largest Number of Rows: The Seq scan on food_des at line 4 and :The hash join at line 2, rows=7146
Physical Plan After:
CREATE INDEX idx21 ON food_des(fdgrp_cd);
Index: `fdgrp_cd` on `food_des` (B-Tree).
Justification: Facilitates filtering and joins on `fdgrp_cd`.
Turned off seq scan

	QUERY PLAN text
1	Aggregate (cost=473.19..473.20 rows=1 width=32) (actual time=2.240..2.242 rows=1 loops=1)
2	-> Hash Join (cost=13.08..455.32 rows=7146 width=6) (actual time=0.032..1.728 rows=7146 loops=1)
3	Hash Cond: (fod.fdgrp_cd = fd.fdgrp_cd)
4	-> Index Scan using idx21 on food_des fod (cost=0.28..420.45 rows=7146 width=11) (actual time=0.011..0.793 rows=7146 loops=1)
5	-> Hash (cost=12.50..12.50 rows=24 width=5) (actual time=0.013..0.013 rows=24 loops=1)
6	Buckets: 1024 Batches: 1 Memory Usage: 9kB
7	-> Index Only Scan using fd_group_pkey on fd_group fd (cost=0.14..12.50 rows=24 width=5) (actual time=0.003..0.007 rows=24 loops=1)
8	Heap Fetches: 24
9	Planning Time: 0.149 ms
10	Execution Time: 2.268 ms

Highest Cost: The index scan on food_des at line 4, `cost=420`
Slowest Runtime:The hash join at line 2, `timing=0.922 ms`
Largest Number of Rows: The index scan on food_des at line 4 and The hash join at line 2, rows=7146

Before Index Creation:

- Used **Sequential Scan** on `food_des`, with a **cost of 233** and a **Hash Join runtime of 1.15 ms**.
- The **Sequential Scan** and **Hash Join** processed the **largest number of rows (7146)**.

After Index Creation:

- Switched to an **Index Scan** on `food_des` with a filtered index on `fdgrp_cd`, increasing the **cost to 420** but improving the **Hash Join runtime to 0.922 ms**.
- The **Index Scan** and **Hash Join** still processed the **same 7146 rows**.

Reason for the Observed Behavior:

- **Before Index**: The Sequential Scan accessed all rows in `food_des`, leading to lower cost but slower data retrieval for the join.
- **After Index**: The Index Scan reduced row access time, improving runtime while increasing cost due to the added overhead of using the index. The rows processed in the join remained constant.

Query 22

Physical Plan Before:

	QUERY PLAN text
1	HashAggregate (cost=310.67..382.13 rows=7146 width=38) (actual time=3.707..4.723 rows=7146 loops=1)
2	Group Key: fod.ndb_no
3	Batches: 1 Memory Usage: 1425kB
4	-> Hash Join (cost=1.54..257.07 rows=7146 width=22) (actual time=0.021..2.111 rows=7146 loops=1)
5	Hash Cond: (fod.fdgrp_cd = fd.fdgrp_cd)
6	-> Seq Scan on food_des fod (cost=0.00..233.46 rows=7146 width=27) (actual time=0.006..0.449 rows=7146 loops=1)
7	-> Hash (cost=1.24..1.24 rows=24 width=5) (actual time=0.009..0.010 rows=24 loops=1)
8	Buckets: 1024 Batches: 1 Memory Usage: 9kB
9	-> Seq Scan on fd_group fd (cost=0.00..1.24 rows=24 width=5) (actual time=0.002..0.004 rows=24 loops=1)
10	Planning Time: 0.193 ms
11	Execution Time: 4.970 ms

Highest Cost: The Seq scan on food_des at line 6, `cost=233`

Slowest Runtime:The HashAggregate at line 1, `timing=2.61 ms`

Largest Number of Rows: The Seq scan on food_des at line 6 and The hash join at line 4, and the The HashAggregate at line 1, rows=7146

Physical Plan After:

CREATE INDEX idx22 ON food_des(ndb_no);

Index: `ndb_no` on `food_des` (B-Tree).

Justification: Optimizes grouping by `ndb_no`.

Turned off seq scan

	QUERY PLAN text
1	HashAggregate (cost=448.94..520.40 rows=7146 width=38) (actual time=3.700..5.058 rows=7146 loops=1)
2	Group Key: fod.ndb_no
3	Batches: 1 Memory Usage: 1425kB
4	-> Hash Join (cost=13.08..395.34 rows=7146 width=22) (actual time=0.022..2.223 rows=7146 loops=1)
5	Hash Cond: (fod.fdgrp_cd = fd.fdgrp_cd)
6	-> Index Scan using idx22 on food_des fod (cost=0.28..360.47 rows=7146 width=27) (actual time=0.007..0.879 rows=7146 loops=1)
7	-> Hash (cost=12.50..12.50 rows=24 width=5) (actual time=0.010..0.011 rows=24 loops=1)
8	Buckets: 1024 Batches: 1 Memory Usage: 9kB
9	-> Index Only Scan using fd_group_pkey on fd_group fd (cost=0.14..12.50 rows=24 width=5) (actual time=0.003..0.006 rows=24 loops=1)
10	Heap Fetches: 24
11	Planning Time: 0.148 ms
12	Execution Time: 5.344 ms

Highest Cost: The Index scan on food_des at line 6, `cost=360`

Slowest Runtime:The HashAggregate at line 1, `timing=2.84 ms`

Largest Number of Rows: The index scan on food_des at line 6 and The hash join at line 4, and the The HashAggregate at line 1, rows=7146

Before Index Creation:

- Used **Sequential Scan** on `food_des`, with a **cost of 233** and a **HashAggregate runtime of 2.61 ms**.
- The **Sequential Scan**, **Hash Join**, and **HashAggregate** all processed the **largest number of rows (7146)**.

After Index Creation:

- Switched to an **Index Scan** on `food_des`, increasing the **cost to 360** and slightly increasing the **HashAggregate runtime to 2.84 ms**.
- The **Index Scan**, **Hash Join**, and **HashAggregate** still processed the **same 7146 rows**.

Reason for the Observed Behavior:

- **Before Index**: Sequential Scan accessed all rows in `food_des`, resulting in a lower cost but slower runtime for the aggregate operation.
- **After Index**: The Index Scan reduced row access time but introduced additional overhead, increasing the cost and slightly impacting runtime. The rows processed in the query remained unchanged.

Query 23

Physical Plan Before:

	QUERY PLAN text
1	Aggregate (cost=489.32..489.33 rows=1 width=16) (actual time=4.987..4.991 rows=1 loops=1)
2	-> HashAggregate (cost=310.67..382.13 rows=7146 width=38) (actual time=3.616..4.694 rows=7146 loops=1)
3	Group Key: fod.ndb_no
4	Batches: 1 Memory Usage: 1425kB
5	-> Hash Join (cost=1.54..257.07 rows=7146 width=22) (actual time=0.041..2.011 rows=7146 loops=1)
6	Hash Cond: (fod.fdgrp_cd = fd.fdgrp_cd)
7	-> Seq Scan on food_des fod (cost=0.00..233.46 rows=7146 width=27) (actual time=0.016..0.428 rows=7146 loops=1)
8	-> Hash (cost=1.24..1.24 rows=24 width=5) (actual time=0.015..0.017 rows=24 loops=1)
9	Buckets: 1024 Batches: 1 Memory Usage: 9kB
10	-> Seq Scan on fd_group fd (cost=0.00..1.24 rows=24 width=5) (actual time=0.006..0.008 rows=24 loops=1)
11	Planning Time: 0.226 ms
12	Execution Time: 5.107 ms

Highest Cost: The Seq scan on food_des at line 7, `cost=233`

Slowest Runtime:The HashAggregate at line 2, `timing=2.68 ms`

Largest Number of Rows: The Seq scan on food_des at line 7 and The hash join at line 5, and the The HashAggregate at line 2, rows=7146

Physical Plan After:

CREATE INDEX idx23 ON food_des(ndb_no);

Index: `ndb_no` on `food_des` (B-Tree).

Justification: Enhances filtering and grouping on `ndb_no`.

Turned off seq scan

	QUERY PLAN text
1	Aggregate (cost=627.59..627.60 rows=1 width=16) (actual time=5.608..5.610 rows=1 loops=1)
2	-> HashAggregate (cost=448.94..520.40 rows=7146 width=38) (actual time=4.400..5.336 rows=7146 loops=1)
3	Group Key: fod.ndb_no
4	Batches: 1 Memory Usage: 1425kB
5	-> Hash Join (cost=13.08..395.34 rows=7146 width=22) (actual time=0.034..2.714 rows=7146 loops=1)
6	Hash Cond: (fod.fdgrp_cd = fd.fdgrp_cd)
7	-> Index Scan using idx23 on food_des fod (cost=0.28..360.47 rows=7146 width=27) (actual time=0.014..1.088 rows=7146 loops=1)
8	-> Hash (cost=12.50..12.50 rows=24 width=5) (actual time=0.012..0.012 rows=24 loops=1)
9	Buckets: 1024 Batches: 1 Memory Usage: 9kB
10	-> Index Only Scan using fd_group_pkey on fd_group fd (cost=0.14..12.50 rows=24 width=5) (actual time=0.004..0.007 rows=24 loops=1)
11	Heap Fetches: 24
12	Planning Time: 0.261 ms
13	Execution Time: 5.669 ms

Highest Cost: The index scan on food_des at line 7, `cost=360`
Slowest Runtime:The HashAggregate at line 2, `timing=2.62 ms`
Largest Number of Rows: The Seq scan on food_des at line 7 and The hash join at line 5, and the The HashAggregate at line 2, rows=7146

Before Index Creation:

- Used **Sequential Scan** on `food_des`, with a **cost of 233** and a **HashAggregate runtime of 2.68 ms**.
- The **Sequential Scan**, **Hash Join**, and **HashAggregate** processed the **largest number of rows (7146)**.

After Index Creation:

- Switched to an **Index Scan** on `food_des`, increasing the **cost to 360** and slightly improving the **HashAggregate runtime to 2.62 ms**.
- The **Index Scan**, **Hash Join**, and **HashAggregate** still processed the **same 7146 rows**.

Reason for the Observed Behavior:

- **Before Index**: Sequential Scan accessed all rows in `food_des`, resulting in lower cost but slower runtime for the aggregation operation.
- **After Index**: The Index Scan reduced row access time but introduced additional overhead, increasing the cost while marginally improving runtime. The total rows processed during the query remained constant.

Query 24

Physical Plan Before:

	QUERY PLAN text
1	Seq Scan on src_cd (cost=0.00..1.10 rows=10 width=44) (actual time=0.597..0.599 rows=10 loops=1)
2	Planning Time: 1.053 ms
3	Execution Time: 0.612 ms

Highest Cost: The seq scan on src_cd at line 1, `cost=1.1`

Slowest Runtime:The seq scan on src_cd at line 1, `timing=0.599 ms`

Largest Number of Rows: The seq scan on src_cd at line 1, rows =10

Physical Plan After:

CREATE INDEX idx24 ON src_cd(srccd_desc);

Index: `srccd_desc` on `src_cd` (B-Tree).

Justification: Improves filtering on `srccd_desc`.

Turned off seq scan

	QUERY PLAN text
1	Index Only Scan using idx24 on src_cd (cost=0.14..12.29 rows=10 width=44) (actual time=0.048..0.050 rows=10 loops=1)
2	Heap Fetches: 10
3	Planning Time: 0.072 ms
4	Execution Time: 0.061 ms

Highest Cost: The seq scan on src_cd at line 1, `cost=12.3`

Slowest Runtime:The Seq scan on src_cd at line 1, `timing=0.05 ms`

Largest Number of Rows: The seq scan on src_cd at line 1, rows =10

Before Index Creation:

- Used **Sequential Scan** on `src_cd`, with a **cost of 1.1** and an **execution time of 0.599 ms**.
- The **Sequential Scan** processed **10 rows**.

After Index Creation:

- Switched to an **Index Only Scan** on `src_cd`, increasing the **cost to 12.3** but significantly reducing the **execution time to 0.05 ms**.
- The **Index Only Scan** still processed **10 rows**.

Reason for the Observed Behavior:

- **Before Index**: Sequential Scan accessed all rows, resulting in lower cost but slower data retrieval.
- **After Index**: The Index Only Scan allowed faster data retrieval, significantly improving runtime while slightly increasing the cost due to index overhead. The number of rows remained the same.