Advanced Databases

Phase 3 – Index and Optimizations of Queries

Project: VitaDiet

Martin Zdraveski - 231550

Denis Skuka - 231128

Ahmet Turan Yanar – 231506

View1: User profile

1. The primary filter for the vw_user_profile view will be by the user's id. It can also be filtered by email, role, and active_diet.

2. This view will be used when a user opens their profile page in the app, displaying their personal information, current role, and active diet plan. Performance is important here since this view is loaded every time a user accesses their profile.

3. The initial view execution time is 5s 336ms.

```
[2026-04-30 17:09:49] advdb_202526l_prj_vitadiet.public> SELECT * FROM vw_user_profile
                                                         WHERE id = 9231
[2026-04-30 17:09:54] 15 rows retrieved starting from 1 in 5 s 336 ms (execution: 4 s 306 ms,
```

This is not an acceptable time which is why we use indexes.

4. The slowest scans are the sequential scans

Operation	Params	Rows	Total Cost	Startup Cost
Select				
Nested Loops (Nested Loop)		102	209342.62	2000.87
Nested Loops (Nested Loop)		2	19768.06	1000.58
Index Scan	table: User; index: User_pkey;	1	8.44	0.42
Unknown (Gather)		2	19759.6	1000.15
Nested Loops (Nested Loop)		1	18759.4	0.15
Full Scan (Seq Scan)	table: user_roles;	1	18751.22	0.0
Unknown (Memoize)		1	8.17	0.15
Index Scan	table: roles; index: roles_pkey;	1	8.16	0.14
Temporary (Materialize)		51	189573.41	1000.29
Unknown (Gather)		51	189573.15	1000.29
Nested Loops (Nested Loop)		21	188568.05	0.29
Full Scan (Seq Scan)	table: user_diets;	21	188420.0	0.0
Index Scan	table: diet; index: diet_pkey;	1	7.05	0.29

and they can be improved with indexes. The time spent performing insert and update operations is reduced.

```
[2026-04-30 19:15:22] advdb_202526l_prj_vitadiet.public> INSERT INTO "User" (username,email, password, height, weight, gender, created_at)
                                                         VALUES ('Marko','testuser1@gmail.com', 'password123', 175, 70, 'male', NOW())
[2026-04-30 19:15:23] 1 row affected in 243 ms
```

```
[2026-04-30 19:16:33] advdb_202526l_prj_vitadiet.public> UPDATE "User" SET height = 180 WHERE email = 'testuser1@gmail.com'
[2026-04-30 19:16:33] 1 row affected in 431 ms
```

```
CREATE INDEX idx_user_email ON "User"(id);
CREATE INDEX idx_user_roles_userid ON User_roles(user_id);
CREATE INDEX idx_user_diets_userid ON User_Diets(user_id);
CREATE INDEX idx_user_diets_dietid ON User_Diets(Diet_id);
```

5. The time taken to execute the query with indexes is 1.052ms, which is an acceptable time.

```
Planning:
  Buffers: shared hit=24
Planning Time: 0.723 ms
Execution Time: 0.329 ms
```

Operation	Params	Rows	Total Cost	Startup Cost
Select				
Nested Loops (Nested Loop)		102	402.36	1.73
Nested Loops (Nested Loo		2	32.08	1.01
Nested Loops (Nested L		2	19.79	0.85
Index Scan	table: User; index: idx_user_id;	1	8.44	0.42
Index Scan	table: user_roles; index: idx_user_roles_userid;	2	11.33	0.43
Unknown (Memoize)		1	8.17	0.15
Index Scan	table: roles; index: roles_pkey;	1	8.16	0.14
Temporary (Materialize)		51	369.14	0.72
Nested Loops (Nested L		51	368.88	0.72
Index Scan	table: user_diets; index: idx_user_diets_userid;	51	9.33	0.43
Index Scan	table: diet; index: diet_pkey;	1	7.05	0.29

6. The time elapsed in performing insert and update operations after indexing is:

```
[2026-04-30 19:34:24] advdb_202526l_prj_vitadiet.public> INSERT INTO "User" (username,email, password, height, weight, gender, created_at)
                                                         VALUES ('Darko','testuser2@gmail.com', 'password123', 175, 70, 'male', NOW())
[2026-04-30 19:34:26] 1 row affected in 1 s 511 ms
```

```
[2026-04-30 19:36:59] advdb_202526l_prj_vitadiet.public> UPDATE "User" SET height = 180 WHERE email = 'testuser2@gmail.com'
[2026-04-30 19:36:59] 1 row affected in 395 ms
```

View2: Daily food summary

1. The primary filter for the view vw_daily_food_summary is with its id (id of the user)
2. This view is used when a user looks at their daily food summary, displaying the number of kcal, carbs, protein and fats they've consumed in the day. Performance is important here since this view is loaded every time a user accesses that page.

3. The initial view execution time is 3m 44s.

```
[2026-04-30 19:45:40] advdb_202526l_prj_vitadiet.public> SELECT * FROM vw_daily_food_summary
                                                         WHERE UserId = 8765
[2026-04-30 19:49:24] 0 rows retrieved in 3 m 44 s 170 ms (execution: 3 m 43 s 680 ms, fetching: 490 ms)
```

This is not an acceptable time which is why we use indexes. 4.

The slowest scans are the sequential scans of foodlogs

Operation	Params	Rows	Total Cost	Startup Cost
Select				
Aggregate		15	232771.25	232767.22
Unknown (Gather Merge)		20	232770.36	232767.22
Aggregate		10	231768.03	231767.2
Sort		10	231767.23	231767.2
Hash Join		10	231767.03	222049.32
Hash Join		10	231764.79	222047.13
Full Scan (Seq Scan)	table: food_nutrients;	242931	7288.31	0.0
Transformation (Hash)		6	222047.06	222047.06
Full Scan (Seq Scan)	table: foodlogs;	6	222047.06	0.0
Transformation (Hash)		53	1.53	1.53
Full Scan (Seq Scan)	table: nutrient;	53	1.53	0.0

and they can be improved with indexes. The time spent performing insert and update operations is reduced.

```
[2026-05-01 09:16:45] advdb_202526l_prj_vitadiet> INSERT INTO foodlogs (quantity, unit, meal_type, datetime, userid, foodid)
                                                  VALUES (12.50, 'g', 'snack', '2026-05-01 09:14:00', 1, 101)
[2026-05-01 09:16:57] 1 row affected in 12 s 623 ms
```

```
[2026-05-01 09:27:01] advdb_202526l_prj_vitadiet.public> UPDATE foodlogs
                                                         SET quantity = 25.00,
                                                             unit = 'g'
                                                         WHERE id = 12

[2026-05-01 09:27:01] completed in 298 ms
```

```
CREATE INDEX idx_foodlogs_userid ON FoodLogs(UserId);
CREATE INDEX idx_foodlogs_foodid ON FoodLogs(FoodId);
CREATE INDEX idx_foodnutrients_foodid ON Food_nutrients(FoodId);
CREATE INDEX idx_foodnutrients_nutrientid ON Food_nutrients(NutrientId);
CREATE INDEX idx_foodlogs_user_date_lookup ON foodlogs (userid, datetime);
```

5. The time taken to execute the query with indexes is 460ms, which is an acceptable time.

```
Planning Time: 133.020 ms
Execution Time: 327.834 ms
```

Operation	Params	Rows	Total Cost	Startup Cost
Select				
Aggregate		15	208.51	206.73
Sort		24	206.79	206.73
Hash Join		24	206.18	3.05
Nested Loops (Nes		24	203.87	0.86
Index Scan	table: foodlogs; index: idx_foodlogs_userid;	15	64.7	0.43
Index Scan	table: food_nutrients; index: idx_foodnutrients_foodid;	31	8.97	0.42
Transformation (Hash)		53	1.53	1.53
Full Scan (Seq S	table: nutrient;	53	1.53	0.0

6. The time elapsed in performing insert and update operations after indexing is:

```
[2026-05-01 09:48:58] advdb_202526l_prj_vitadiet.public> INSERT INTO foodlogs (quantity, unit, meal_type, datetime, userid, foodid)
                                                         VALUES (12.50, 'g', 'snack', '2026-05-01 09:14:20', 1, 101)
[2026-05-01 09:48:58] 1 row affected in 125 ms
```

```
[2026-05-01 09:50:12] advdb_202526l_prj_vitadiet.public> UPDATE foodlogs
                                                         SET quantity = 22.00,
                                                             unit = 'g'
                                                         WHERE id = 12
[2026-05-01 09:50:12] completed in 50 ms
```

View3: Exercise calories burned per user

1. The primary filter for the view vw_user_calories_burned is with the UserId or with its log date.

2. The primary purpose of this view is to aggregate data and to build a user's dashboard.

3. The initial view execution time is 12m 42s.

```
[2026-05-01 10:35:05] advdb_202526l_prj_vitadiet.public> SELECT * FROM vw_user_calories_burned
                                                         WHERE UserId = 1
                                                         ORDER BY log_date DESC
[2026-05-01 10:47:48] 39 rows retrieved starting from 1 in 12 m 42 s 609 ms (execution: 12 m 40 s 909 ms, fetching: 1 s 700 ms)
```

This is not an acceptable time which is why we use indexes.

4. The slowest scan is the sequential scan for activitylogs

Operation	Params	Rows	Total Cost	Startup Cost
⇦ Select				
Sort		36	510400.83	510400.74
Aggregate		36	510399.81	510395.02
Unknown (Gather Merge)		30	510398.9	510395.02
Aggregate		15	509395.41	509395.0
Sort		15	509395.03	509395.0
Full Scan (Seq Scan)	table: activitylogs;	15	509394.7	0.0

and they can be improved with indexes. The time spent performing insert and update operations is reduced.

```
[2026-05-01 10:58:32] advdb_202526l_prj_vitadiet.public> INSERT INTO activitylogs (datetime, userid, duration_minutes, calories_burned, exerciseid)
                                                         VALUES ('2026-05-01 09:40:00',1,10,50.55,10)
[2026-05-01 10:58:37] 1 row affected in 4 s 674 ms
```

```
[2026-05-01 11:01:01] advdb_202526l_prj_vitadiet.public> UPDATE activitylogs
                                                         SET calories_burned = 45.12
                                                         WHERE id = 30
[2026-05-01 11:01:03] 1 row affected in 1 s 839 ms
```

```
CREATE INDEX idx_activitylogs_userid ON ActivityLogs (UserId);
```

5. The time taken to execute the query with indexes is 362ms, which is an acceptable time.

```
Planning Time: 0.205 ms
Execution Time: 362.058 ms
```

Operation	Params	Rows	Total Cost	Startup Cost
Select				
Sort		23	98.61	98.56
Aggregate		23	98.04	97.4
Sort		23	97.46	97.4
Index Scan	table: activitylogs; index: idx_activitylogs_userid;	23	96.88	0.44

6. The time elapsed in performing insert and update operations after indexing is:

```
[2026-05-01 13:57:37] advdb_202526l_prj_vitadiet.public> INSERT INTO activitylogs (datetime, userid, duration_minutes, calories_burned, exerciseid
                                                         VALUES ('2026-05-01 13:55:00',1,10,50.55,10)
[2026-05-01 13:57:38] 1 row affected in 118 ms
```

```
[2026-05-01 13:59:00] advdb_202526l_prj_vitadiet.public> UPDATE activitylogs
                                                         set calories_burned = 32.24
                                                         where id = 32
[2026-05-01 13:59:00] 1 row affected in 132 ms
```

View4: Food Nutritional Info

1. The primary filter for the view vw_food_nutrition is the food_id. This allows the system to pinpoint a specific ingredient or product and retrieve its complete nutritional profile across multiple linked tables without scanning the entire database.

2. The primary purpose of this view is to normalize and consolidate food metadata. It flattens the relationship between food items, their categories, and their specific nutrient values (like protein or vitamins) into a single, readable format. This is essential for building a "Food Details" page where a user needs to see all nutritional facts for a specific item in one place.
3. The initial view execution time is 9s 769ms.

```
[2026-05-01 17:22:00] advdb_202526l_prj_vitadiet.public> SELECT * FROM vw_food_nutrition
                                                         WHERE food_id = 12345
[2026-05-01 17:22:09] 48 rows retrieved starting from 1 in 9 s 769 ms (execution: 7 s 104 ms, fetching: 2 s 665 ms)
```

This is not an acceptable time which is why we use indexes.

4. The foodnutrients table already has indexes from previous views.

Operation	Params	Rows	Total Cost	Startup Cost
Select				
Hash Join		30	28.27	3.33
Nested Loops (Nested		30	26.0	1.14
Nested Loops (Nest		1	16.75	0.71
Index Scan	table: food; index: food_pkey;	1	8.44	0.42
Index Scan	table: category; index: category_pkey;	1	8.3	0.29
Index Scan	table: food_nutrients; index: idx_foodnutrients_foodid;	30	8.95	0.42
Transformation (Hash)		53	1.53	1.53
Full Scan (Seq Scan)	table: nutrient;	53	1.53	0.0

We will add a category index to reduce cost.

```
[2026-05-01 17:35:07] advdb_202526l_prj_vitadiet.public> UPDATE Food
                                                         SET category_id = (SELECT category_id FROM Category LIMIT 1)
                                                         WHERE id = 12345
[2026-05-01 17:35:09] 1 row affected in 1 s 921 ms
```

```
[2026-05-01 17:36:34] advdb_202526l_prj_vitadiet.public> INSERT INTO Food (name, calories_per_100g, category_id)
                                                           VALUES ('Test Food', 100, 1)
[2026-05-01 17:36:36] 1 row affected in 1 s 286 ms
```

```
CREATE INDEX idx_food_categoryid ON Food(category_id);
```

5. The time taken to execute the query with indexes is 74ms, which is an acceptable time.

```
Planning Time: 0.612 ms
Execution Time: 74.299 ms
```

Operation	Params	Rows	Total Cost	Startup Cost
Select				
Hash Join		30	28.27	3.33
Nested Loops (Nested		30	26.0	1.14
Nested Loops (Nest		1	16.75	0.71
Index Scan	table: food; index: food_pkey;	1	8.44	0.42
Index Scan	table: category; index: category_pkey;	1	8.3	0.29
Index Scan	table: food_nutrients; index: idx_foodnutrients_foodid;	30	8.95	0.42
Transformation (Hash)		53	1.53	1.53
Full Scan (Seq Scan)	table: nutrient;	53	1.53	0.0

6. The time elapsed in performing insert and update operations after indexing is:

```
[2026-05-01 17:50:33] advdb_202526l_prj_vitadiet.public> UPDATE Food
                                                         SET category_id = (SELECT category_id FROM Category LIMIT 1)
                                                         WHERE id = 12345
[2026-05-01 17:50:33] 1 row affected in 326 ms
```

```
[2026-05-01 17:51:42] advdb_202526l_prj_vitadiet.public> INSERT INTO Food (name, calories_per_100g, category_id)
                                                           VALUES ('Test Food 2', 100, 1)
[2026-05-01 17:51:45] 1 row affected in 2 s 706 ms
```

View5: User achievements progress

1. The primary filter for the view vw_user_achievements is the user_id. This allows the system to fetch the specific trophy room or milestone history for a single user without scanning the global list of all achievements earned by the entire user base.

2. The primary purpose of this view is to monitor progress and gamification. It flattens the relationship between the raw user data, the specific achievements they have unlocked, and the metadata (like the target value or period) of those achievements. This view is intended for the User Profile or Dashboard, allowing users to see their accomplishments sorted by the most recent "win".

3. The initial view execution time is 2m 36s.

```
[2026-05-02 13:00:14] advdb_202526l_prj_vitadiet.public> SELECT * FROM vw_user_achievements
                                                         WHERE user_id = 3216
                                                         ORDER BY dateEarned DESC
[2026-05-02 13:02:51] 14 rows retrieved starting from 1 in 2 m 36 s 671 ms (execution: 2 m 35 s 723 ms, fetching: 948 ms)
```

This is not an acceptable time which is why we use indexes.

4. The gather merge slows down the search.

Operation	Params	Rows	Total Cost	Startup Cost
Select				
Unknown (Gather Merge)		44	190429.61	190424.47
Sort		22	189424.51	189424.45
Hash Join		22	189423.96	12.68
Nested Loops (Nested l		22	189411.65	0.42
Full Scan (Seq Scan)	table: user_achievements;	22	189225.7	0.0
Index Scan	table: User; index: idx_user_id;	1	8.44	0.42
Transformation (Hash)		100	11.0	11.0
Full Scan (Seq Scan)	table: achievements;	100	11.0	0.0

We will add indexes to replace it with a standard sort operation.

```
[2026-05-02 13:10:01] advdb_202526l_prj_vitadiet.public> INSERT INTO User_achievements (user_id, achievement_id, dateEarned)
                                                         VALUES (1, 23, CURRENT_TIMESTAMP)
[2026-05-02 13:10:03] 1 row affected in 1 s 186 ms
```

```
[2026-05-02 13:10:48] advdb_202526l_prj_vitadiet.public> UPDATE User_achievements
                                                         SET dateEarned = '2026-05-01 10:00:00'
                                                         WHERE user_id = 1 AND achievement_id = 23
[2026-05-02 13:10:52] 3 rows affected in 3 s 34 ms
```

```
CREATE INDEX idx_userachievements_userid ON User_achievements(user_id);
CREATE INDEX idx_userachievements_achievementid ON User_achievements(achievement_id);
```

5. The time taken to execute the query with indexes is 24.831ms, which is an acceptable time.

```
Planning Time: 0.362 ms
Execution Time: 24.469 ms
```

Operation	Params	Rows	Total Cost	Startup Cost
Select				
Sort		52	32.32	32.19
Hash Join		52	30.71	13.11
Nested Loops (Nes		52	18.31	0.86
Index Scan	table: User; index: idx_user_id;	1	8.44	0.42
Index Scan	table: user_achievements; index: idx_userachievements_userid;	52	9.35	0.43
Transformation (Hash		100	11.0	11.0
Full Scan (Seq S	table: achievements;	100	11.0	0.0

6. The time elapsed in performing insert and update operations after indexing is:

```
[2026-05-02 13:20:02] advdb_202526l_prj_vitadiet.public> INSERT INTO User_achievements (user_id, achievement_id, dateEarned)
                                                         VALUES (1, 24, CURRENT_TIMESTAMP)
[2026-05-02 13:20:02] 1 row affected in 32 ms
```

```
[2026-05-02 13:19:23] advdb_202526l_prj_vitadiet.public> UPDATE User_achievements
                                                         SET dateEarned = '2026-05-01 11:00:00'
                                                         WHERE user_id = 1 AND achievement_id = 23
[2026-05-02 13:19:23] 3 rows affected in 50 ms
```

View6: User streak summary

1. The primary filter for this view is the user_id. This allows the application to pull the specific consistency metrics for an individual user such as their daily login or workout streak without the database needing to evaluate the streak data for the entire population of the vitadiet platform.

2. The primary purpose of this view is engagement tracking and behavioral analysis. It simplifies the relationship between the user, their active streaks, and the specific rules of those streaks (like metric_type) into a single flattened object. This is essential for building "consistency" widgets on a dashboard that motivate users to maintain their health habits.

3. The initial view execution time is 41s 800ms.

```
[2026-05-02 15:10:07] advdb_202526l_prj_vitadiet.public> SELECT * FROM vw_user_streaks
                                                         WHERE user_id = 7654
                                                         ORDER BY last_updated DESC
[2026-05-02 15:10:49] 15 rows retrieved starting from 1 in 41 s 800 ms (execution: 40 s 576 ms, fetching: 1 s 224 ms)
```

This is not an acceptable time which is why we use indexes.

4. The slowest scan is the sequential scan of streak

Operation	Params	Rows	Total Cost	Startup Cost
⇦ Select				
Unknown (Gather Merge)		32	183682.38	183678.64
Sort		16	182678.66	182678.62
Hash Join		16	182678.3	11.78
Nested Loops (Nested l		16	182666.91	0.42
Full Scan (Seq Scan)	table: streak;	16	182531.67	0.0
Index Scan	table: User; index: idx_user_id;	1	8.44	0.42
Transformation (Hash)		60	10.6	10.6
Full Scan (Seq Scan)	table: streak_type;	60	10.6	0.0

and it can be improved with indexes. The time spent performing insert and update operations is reduced.

```
[2026-05-02 15:17:16] advdb_202526l_prj_vitadiet.public> INSERT INTO Streak (User_id, Streak_type_id, current_count, longest_count, last_updated)
                                                         VALUES (1, 2, 1, 2, CURRENT_TIMESTAMP)
[2026-05-02 15:17:16] 1 row affected in 107 ms
```

```
[2026-05-02 15:17:59] advdb_202526l_prj_vitadiet.public> UPDATE Streak
                                                         SET current_count = current_count + 1,
                                                             last_updated = CURRENT_TIMESTAMP
                                                         WHERE User_id = 1 AND Streak_type_id = 2
[2026-05-02 15:18:01] 4 rows affected in 2 s 167 ms
```

```
CREATE INDEX idx_streak_userid ON Streak(User_id);
CREATE INDEX idx_streak_streaktypeid ON Streak(Streak_type_id);
```

5. The time taken to execute the query with indexes is 90.341ms, which is an acceptable time.

```
Planning Time: 0.336 ms
Execution Time: 90.005 ms
```

Operation	Params	Rows	Total Cost	Startup Cost
Select				
Sort		39	145.21	145.11
Hash Join		39	144.08	12.21
Nested Loops (Nested		39	132.62	0.86
Index Scan	table: User; index: idx_user_id;	1	8.44	0.42
Index Scan	table: streak; index: idx_streak_userid;	39	123.78	0.43
Transformation (Hash)		60	10.6	10.6
Full Scan (Seq Scan	table: streak_type;	60	10.6	0.0

6. The time elapsed in performing insert and update operations after indexing is:

```
[2026-05-02 15:23:48] advdb_202526l_prj_vitadiet.public> INSERT INTO Streak (User_id, Streak_type_id, current_count, longest_count, last_updated)
                                                         VALUES (2, 2, 1, 2, CURRENT_TIMESTAMP)
[2026-05-02 15:23:48] 1 row affected in 171 ms
```

```
[2026-05-02 15:24:27] advdb_202526l_prj_vitadiet.public> UPDATE Streak
                                                         SET current_count = current_count + 1,
                                                             last_updated = CURRENT_TIMESTAMP
                                                         WHERE User_id = 2 AND Streak_type_id = 2
[2026-05-02 15:24:27] 4 rows affected in 30 ms
```

View7: Body Measurement History

1. The primary filter for this view is the user_id. Since our project involves 20 million rows, filtering by a specific user allows the database to ignore the massive volume of global data and focus solely on the physical progress of a single individual.

2. The primary purpose of this view is health and physical progress tracking. It serves as the data source for generating trend charts (like weight loss or body fat percentage over time) on the user's dashboard. By ordering the results by measured_at DESC, the view ensures that the most recent data is always at the top for immediate review.

3. The initial view execution time is 1m 43s.

```
[2026-05-02 17:17:29] advdb_202526l_prj_vitadiet.public> SELECT * FROM vw_body_measurements
                                                         WHERE user_id = 12345
                                                         ORDER BY measured_at DESC
[2026-05-02 17:19:12] 10 rows retrieved starting from 1 in 1 m 43 s 330 ms (execution: 1 m 42 s 820 ms, fetching: 510 ms)
```

This is not an acceptable time which is why we use indexes.

4. The slowest scan is the sequential scan of bodymeasurements

Operation	Params	Rows	Total Cost	Startup Cost
Select				
Unknown (Gather Merge)		8	146576.14	146575.21
Sort		4	145575.19	145575.18
Nested Loops (Nested L		4	145575.14	0.42
Full Scan (Seq Scan)	table: bodymeasurements;	4	145541.33	0.0
Index Scan	table: User; index: idx_user_id;	1	8.44	0.42

and it can be improved with indexes. The time spent performing insert and update operations is reduced.

```
[2026-05-02 17:23:24] advdb_202526l_prj_vitadiet.public> INSERT INTO BodyMeasurements (User_id, weight, waist, body_fat, dateTime)
                                                         VALUES (1, 82.5, 90.0, 18.5, CURRENT_TIMESTAMP)
[2026-05-02 17:23:26] 1 row affected in 1 s 571 ms
```

```
[2026-05-02 17:23:58] advdb_202526l_prj_vitadiet.public> UPDATE BodyMeasurements
                                                         SET weight = 81.5,
                                                             body_fat = 18.0
                                                         WHERE User_id = 1 AND dateTime = '2026-05-01 08:30:00'
[2026-05-02 17:24:00] completed in 1 s 437 ms
```

```
CREATE INDEX idx_bodymeasurements_userid ON BodyMeasurements(User_id);
CREATE INDEX idx_bodymeasurements_datetime ON BodyMeasurements(dateTime);
```

5. The time taken to execute the query with indexes is 383ms, which is an acceptable time.

```
Planning Time: 0.239 ms
Execution Time: 383.737 ms
```

Operation	Params	Rows	Total Cost	Startup Cost
⇦ Select				
Sort		10	52.95	52.92
Nested Loops (Nes		10	52.76	0.86
Index Scan	table: User; index: idx_user_id;	1	8.44	0.42
Index Scan	table: bodymeasurements; index: idx_bodymeasurements_userid;	10	44.21	0.43

6. The time elapsed in performing insert and update operations after indexing is:

```
[2026-05-02 17:31:00] advdb_202526l_prj_vitadiet.public> INSERT INTO BodyMeasurements (User_id, weight, waist, body_fat, dateTime)
                                                         VALUES (1, 82.5, 90.0, 18.5, CURRENT_TIMESTAMP)
[2026-05-02 17:31:03] 1 row affected in 3 s 26 ms
```

```
[2026-05-02 17:31:48] advdb_202526l_prj_vitadiet.public> UPDATE BodyMeasurements
                                                         SET weight = 81.5,
                                                             body_fat = 18.0
                                                         WHERE User_id = 2 AND dateTime = '2026-05-01 08:30:00'
[2026-05-02 17:31:48] completed in 36 ms
```

View8: Diet Plan Details

1. The primary filter for this view is the diet_id. Because a single diet plan can include multiple foods and exercise targets, filtering by the specific ID allows the database to aggregate all related requirements nutritional and physical into a single structured plan without scanning every diet stored in the system.

2. The primary purpose of this view is comprehensive lifestyle planning. It acts as a central hub that joins dietary restrictions (calories/macros) with specific food items and supplemental exercise routines. This view is used to generate the "Daily Plan" or "Assigned Diet" screen for users, providing them with a holistic view of their targets in one place.

3. The initial view execution time is 8s 921ms.

```
[2026-05-02 19:29:15] advdb_202526l_prj_vitadiet.public> SELECT * FROM vw_diet_plan
                                                         WHERE diet_id = 4567
[2026-05-02 19:29:24] 27 rows retrieved starting from 1 in 8 s 921 ms (execution: 7 s 652 ms, fetching: 1 s 269 ms)
```

This is not an acceptable time which is why we use indexes.

4. The slowest scan is the sequential scan of diet_foods

Operation	Params	Rows	Total Cost	Startup Cost
Select				
Nested Loops (Nested Loop)		10	2875.14	0.99
Nested Loops (Nested Lo		2	821.43	0.57
Nested Loops (Nested		2	804.83	0.29
Index Scan	table: diet; index: diet_pkey;	1	8.3	0.29
Full Scan (Seq Scan	table: diet_exercise_targets;	2	796.5	0.0
Index Scan	table: exercise; index: exercise_pkey;	1	8.3	0.28
Temporary (Materialize)		5	2053.6	0.42
Nested Loops (Nested		5	2053.58	0.42
Full Scan (Seq Scan	table: diet_foods;	5	2011.38	0.0
Index Scan	table: food; index: food_pkey;	1	8.44	0.42

and it can be improved with indexes. The time spent performing insert and update operations is reduced.

```
[2026-05-02 19:39:43] advdb_202526l_prj_vitadiet.public> INSERT INTO Diet_foods (DietId, FoodId, amount, unit)
                                                         VALUES (5, 120, 200, 'grams')
[2026-05-02 19:39:45] 1 row affected in 1 s 185 ms
```

```
[2026-05-02 19:40:33] advdb_202526l_prj_vitadiet.public> UPDATE Diet
                                                         SET protein_target = 180,
                                                             calorie_target = 2200
                                                         WHERE id = 5
[2026-05-02 19:40:35] 1 row affected in 1 s 823 ms
```

```
CREATE INDEX idx_dietfoods_dietid ON Diet_foods(DietId);
CREATE INDEX idx_dietfoods_foodid ON Diet_foods(FoodId);
CREATE INDEX idx_dietexercise_dietid ON Diet_Exercise_Targets(DietId);
CREATE INDEX idx_dietexercise_exerciseid ON Diet_Exercise_Targets(ExerciseId);
```

5. The time taken to execute the query with indexes is 413ms, which is an acceptable time.

```
Planning Time: 0.773 ms
Execution Time: 413.901 ms
```

Operation	Params	Rows	Total Cost	Startup Cost
Select				
Nested Loops (Nested Loop)		10	2875.14	0.99
Nested Loops (Nested Lo		2	821.43	0.57
Nested Loops (Nested		2	804.83	0.29
Index Scan	table: diet; index: diet_pkey;	1	8.3	0.29
Full Scan (Seq Scan	table: diet_exercise_targets;	2	796.5	0.0
Index Scan	table: exercise; index: exercise_pkey;	1	8.3	0.28
Temporary (Materialize)		5	2053.6	0.42
Nested Loops (Nested		5	2053.58	0.42
Full Scan (Seq Scan	table: diet_foods;	5	2011.38	0.0
Index Scan	table: food; index: food_pkey;	1	8.44	0.42

6. The time elapsed in performing insert and update operations after indexing is:

```
[2026-05-02 19:42:35] advdb_202526l_prj_vitadiet.public> INSERT INTO Diet_foods (DietId, FoodId, amount, unit)
                                                         VALUES (5, 120, 200, 'grams')
[2026-05-02 19:42:35] 1 row affected in 241 ms
```

```
[2026-05-02 19:48:14] advdb_202526l_prj_vitadiet.public> UPDATE Diet
                                                         SET protein_target = 189,
                                                             calorie_target = 2200
                                                         WHERE id = 5
[2026-05-02 19:48:15] 1 row affected in 189 ms
```

View9: Exercise Library

1. The primary filter for this view is exercise_id. When a user searches for a specific movement or views an exercise detail page, filtering by the ID allows the database to instantly aggregate the specific muscle groups and activity type/s associated with that entry.

2. The primary purpose of this view is content categorization and educational reference. It serves as the searchable "encyclopedia" for the application, allowing users to understand which muscles are being worked and the intensity (calories per minute) of each exercise.

3. The initial execution time is 446ms. This is an acceptable time for the application.

```
[2026-05-03 10:59:30] advdb_202526l_prj_vitadiet.public> SELECT * FROM vw_exercise_library
                                                         WHERE exercise_id = 1236
[2026-05-03 10:59:30] 1 row retrieved starting from 1 in 446 ms (execution: 64 ms, fetching: 382 ms)
```

4. The slowest scan is the sequential scan of exercise_muscles.

Operation	Params	Rows	Total Cost	Startup Cost
Select				
Aggregate		1	83.19	83.17
Sort		1	83.17	83.17
Nested Loops (Nested Loop		1	83.16	0.58
Nested Loops (Nested Lc		1	74.96	0.43
Nested Loops (Nestec		1	16.57	0.43
Index Scan	table: exercise; index: exercise_pkey;	1	8.3	0.28
Index Scan	table: activity; index: activity_pkey;	1	8.16	0.14
Full Scan (Seq Scan)	table: exercise_muscles;	1	58.38	0.0
Index Scan	table: muscle; index: muscle_pkey;	1	8.17	0.15

The time spent performing insert and update operations are:

```
[2026-05-03 11:06:38] advdb_202526l_prj_vitadiet.public> INSERT INTO Exercise (name, calories_per_minute, activity_id)
                                                         VALUES ('Test Exercise', 7.5, 2)
[2026-05-03 11:06:38] 1 row affected in 116 ms
```

```
[2026-05-03 11:08:00] advdb_202526l_prj_vitadiet.public> UPDATE Exercise
                                                         SET calories_per_minute = 8.0
                                                         WHERE id = 50
[2026-05-03 11:08:01] 1 row affected in 59 ms
```