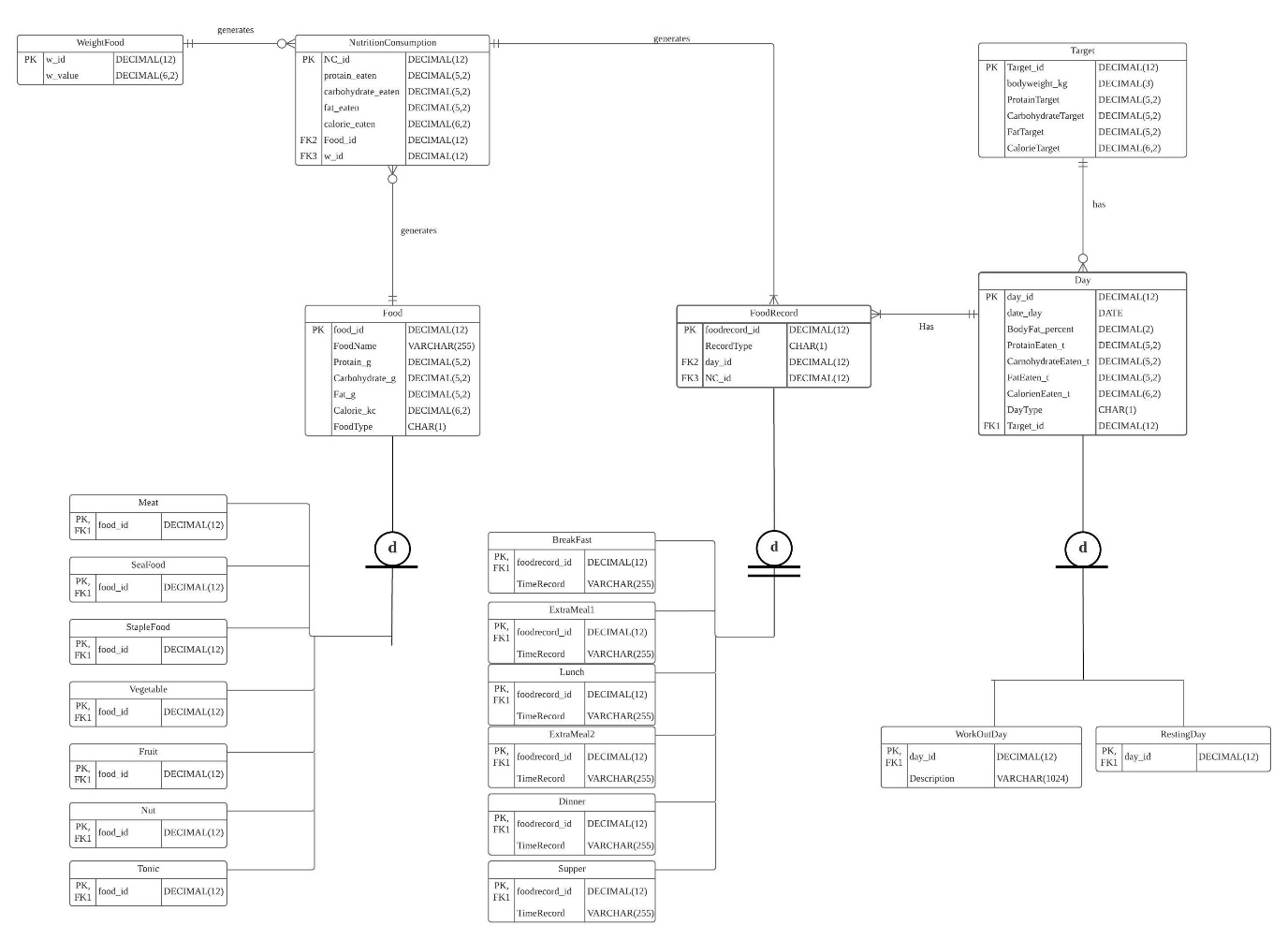
**Correcting mistakes before continue:**

After I tried to insert the data myself, I notice that the longest number for nutrition facts is not enough. So, I changed the longest digits allowed for attributes relating to nutrition facts and weight:



**Transaction driven, reusable stored procedures:**

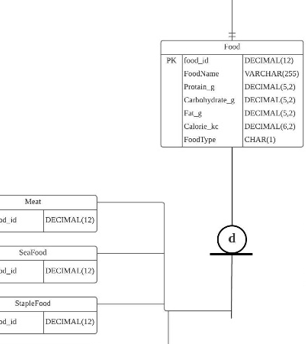
For this part I will choose three use cases to implement transactions.

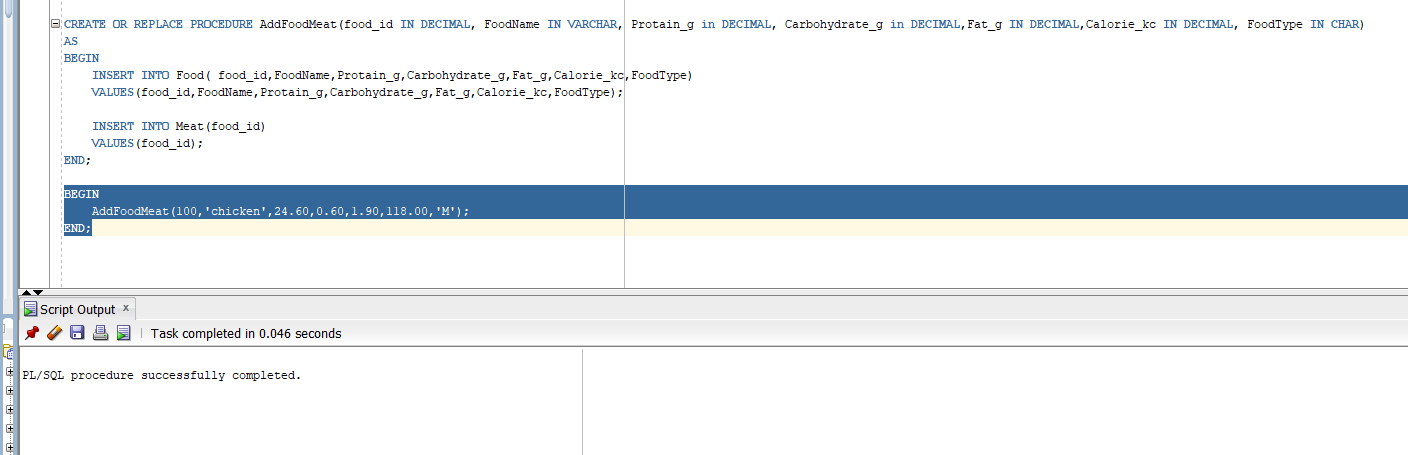
1. A food is a meat, a seafood, a staple food, a vegetable, a fruit, a nut, a tonic or a none of these
2. Each Weight may products many nutrition consumption record; each nutrition consumption record only has one Weight.
3. Each Target may be had by many Day; each Day only has one Target.

The first use case base on the first structural rule:

1. The user input the food name and nutrition facts for that food
2. The database will store information about the food.
3. Further, the foods will be classified into groups.

For this use case, I will implement transactions that record food information and put the information into meat subtype.

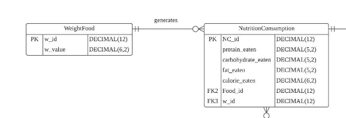




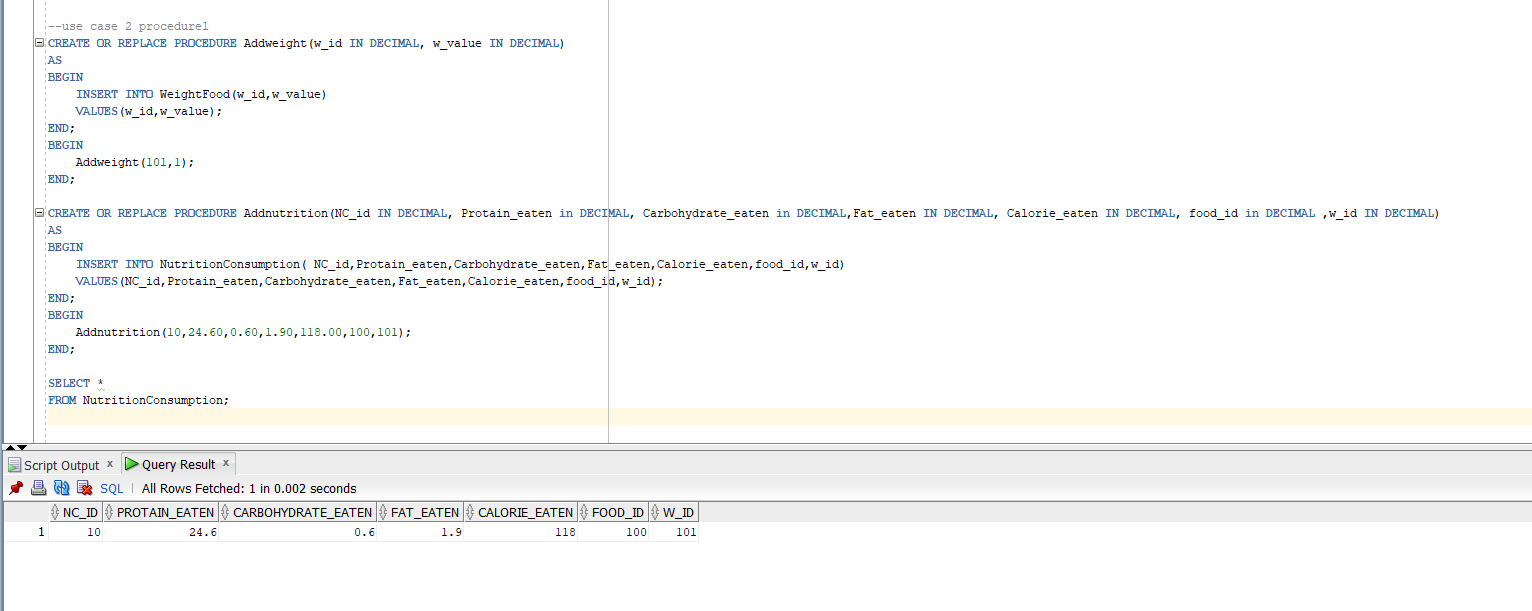
I named the procedure “AddFoodMeat”, and gave it parameters that correspond to the Food and Meat tables. And I hardcode the character “M” for food type. I insert the real food nutrition fact from internet.

The second use case base on the second structural rule:

1. The user enters the wight of the food
2. The wight will be stored in database for next time use
3. The amount of nutrition eaten will be entered into NutritionConsumption



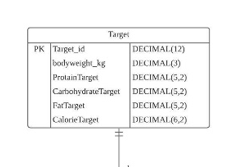
For this user case I insert a row for NutritionConsumption entity, before that, I entered a row into WeightFood entity.

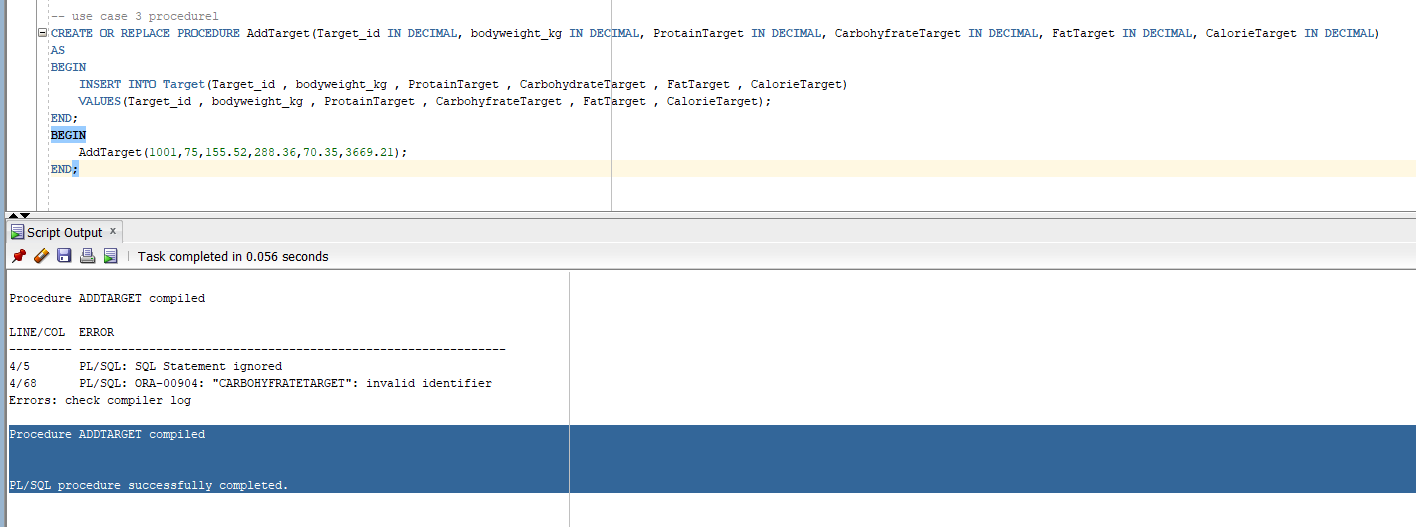


In this case, I used two procedurals, the first one called “Addweight” which add w\_in and w\_value into the weightFood entity. I entered value 1 for w\_value to indicate that the w\_value is in 100g. And the second procedural called “Addnutrition” insert record into NutritionConsumption entity. The nutrition record is generated by 100g chicken breast. I print the NutritionConsumption entity after the procedural to show that the insert was successful.

The third use case is related to Target entity:

1. The user enters the day’s body weight
2. Nutrition target will be calculated and recorded into the database.
3. The targets will be used to compare with the nutrition consumption



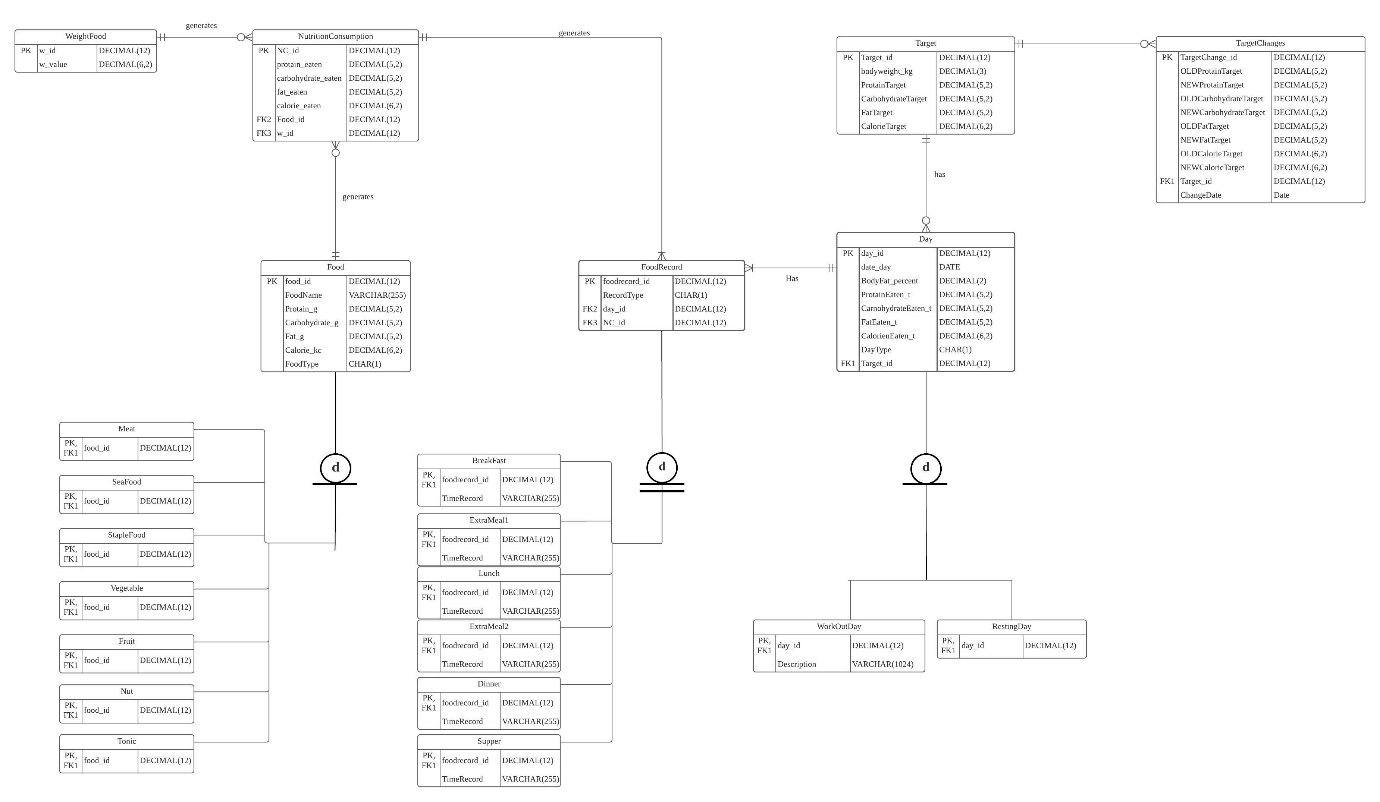


For this use case, I make a procedural called “addTarget” which can insert records into Target entity. I made up all values which are very close to a real situation.

**Maintain history tables with triggers:**

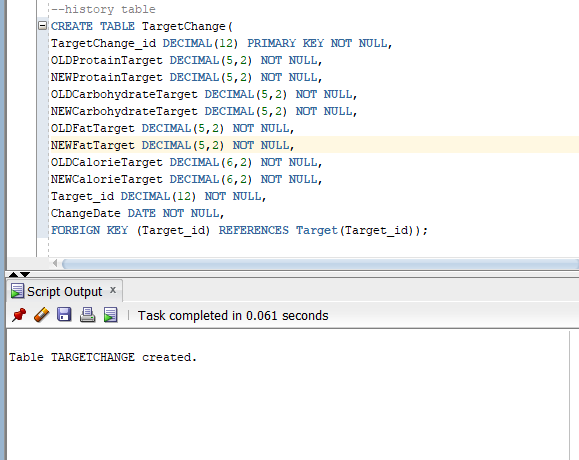
I will maintain a history table for Target entity. I will track nutrition target changes, which would be useful to track. I mentioned earlier that the nutrition target will be decided depends on the body weight, in other words, the nutrition targets are calculated by bodyweight, but the body weight is not the only factor influencing the nutrition targets. The nutrition targets are calculated by body weight multiply by amount of nutrition needed per unit bodyweight. For example, a 75 kg person has a protein target equal to 225g because he needs 3g protein per kg of body weight. Each individual can have unique nutrition targets with same body weight. Also, people can change their targets if their body weight has not changed for long time (which means they need more nutrition). However, people cannot change their targets whatever they want, there is a secure range for the target, for example 2.5g to 6g per unit bodyweight. It will product health risks if people take less than 2.5g or more than 6g protein per unit bodyweight, that’s the reason why the target is also decided by body weight. For my database, it is useful to record the target history changes, because users can use this information to modify their nutrition targets. The first step is to create the history table with old nutrition targets and new nutrition targets, a foreign key to Target table, and date. The new structural database rule would be: each target can have many target changes; each target change is for one target.

The new DBMS physical ERD would be:

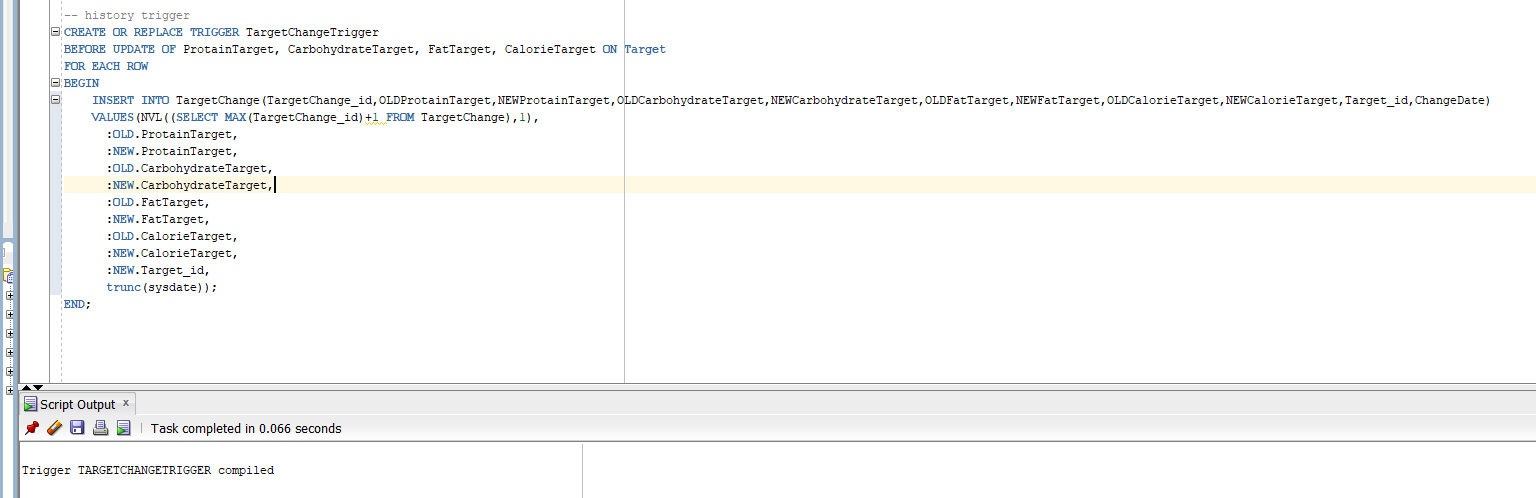


|  |  |
| --- | --- |
| Attribute | Description |
| TargetChange\_id | This is the primary key of the history table, it is DECIMAL (12) to allow for many values. |
| OLDProtainTarget | The protein target before the change. This datatype mirrors the protein target datatype in the Target entity. |
| NEWProtainTarget | Theprotain target after the change. This datatype mirrors the protein terget datatype in the Target entity. |
| OLDCarbohydrateTarget | The carbohydrate target before the change. This datatype mirrors the carbohydrate target datatype in the Target entity. |
| NEWCarbohydrateTarget | The carbohydrate target after the change. This datatype mirrors the carbohydrate target datatype in the Target entity. |
| OLDFatTarget | The fat target before the change. This datatype mirrors the fattarget datatype in the Target entity. |
| NEWFatTarget | The fat target after the change. This datatype mirrors the fat target datatype in the Target entity. |
| OLDCalorieTarget | The calorie target before the change. This datatype mirrors the calorie target datatype in the Target entity. |
| NEWCalorieTarget | The calorie target after the change. This datatype mirrors the calorie target datatype in the Target entity. |
| Target\_id | This is the foreign key to the target table, a reference to the target that had changed in nutrition target. |
| ChangeDate | This is the date the price change occurred, with a DATE dattype. |

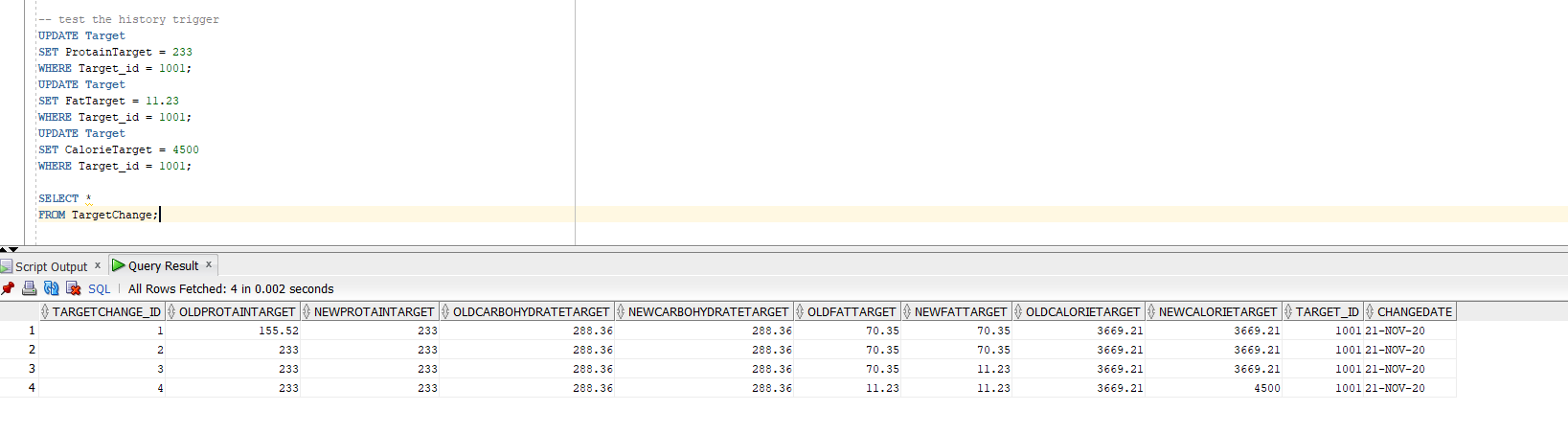
Create the TargetChange table by CREATE commend:



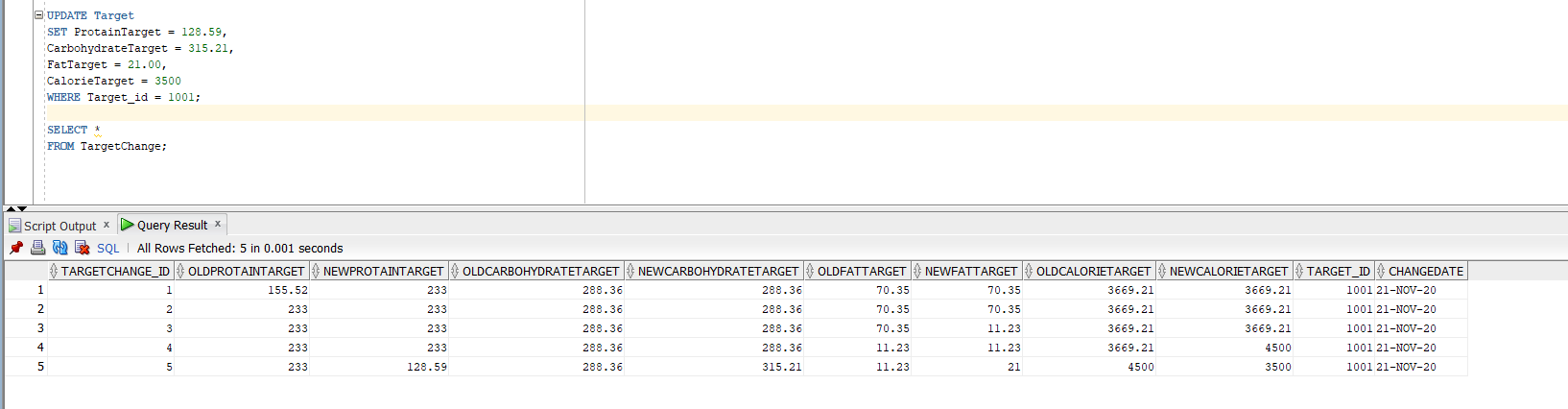
Create the trigger in SQL:



In order to test the trigger, I input three UPDATE commends, one for changing ProtainTarget, one for FatTarget and one for CalorieTarget:



From the query result, I can say that the trigger is working properly. The history table will make the primary ID itself every time it has a new record. If I change anything, the changes will be showed in the query result, if no change is made, the values will be the same, and the new value will be as same as the old value.

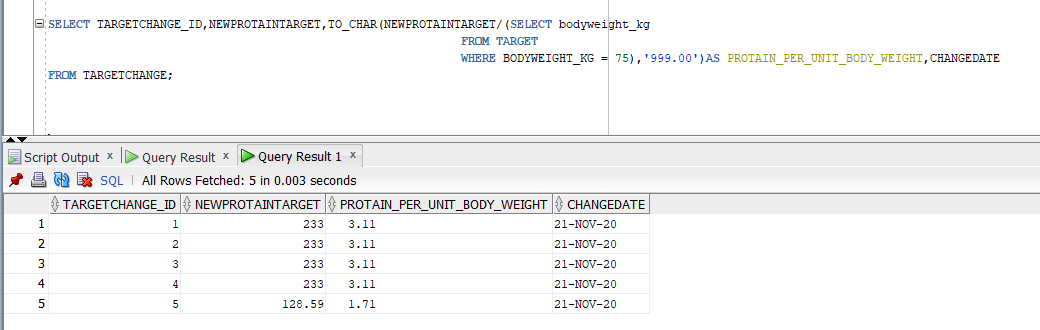


Also, I tried to update all columns together (which is more likely in real life), the history trigger works well.

**Organization-Driven Queries:**

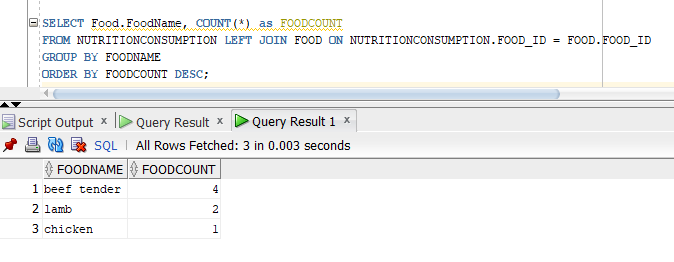
For each query, I will insert sufficient rows for testing the queries. I will try to add as many rows as possible to express my queries.

1. How the protein per unit body weight changed to be heavier than 75kg? This is a useful problem. Let’s assume that a user wants to gain weight, and he or she has stayed being 75kg for longtime. So, the reasonable action is to raise the daily protein consumption. But the user is a cautious person who doesn’t want to make aggressive increases in daily protein. So, it is useful if the database can tell the user how much protein the user has increased for every change the user made on daily protein target in the past. And I want the protein per unit body weight to be printed because this value is more straight forward, and the protein per body wight is the value changed by the user to modify daily protein target.



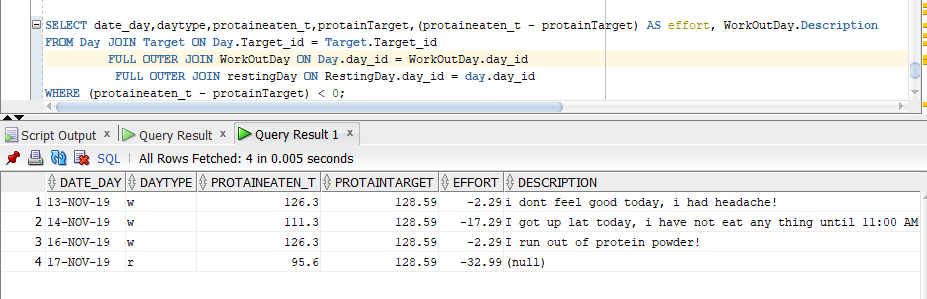
To get the result, I used a subquery, aggregation function and a WHERE clause. By using subquery, I SELECT the desired value from Target table (bodyweight\_kg), I used a WHERE clause to define the value of bodyweight\_kg, and I used an aggregation function to divide NEWPROTAINTARGET from TargetChange history table by bodyweight\_kg. I only SELECT data about protein, because the problem is concentrating on protein Target. From the query result, we can see the protein\_per\_unit\_body\_weight is calculated, when the ProtainTarget change, the protein\_per\_unit\_body\_weight will be changed. And the date of change will be showed.

1. The second question I would like to know that what is the frequency for each food I eat. For this question, I want to know the name of the food I eat for the most of time. Also, I would like to know the name of other food I have eaten and what’s the frequency. Answer of this question helps user to balance the diet. I found it is easy for people who wants to gain weight to eat single food. It is understandable because choosing a new food requires user to search the information for that food and this can increase the potential cost (money and time). But eating single food all the time is not the best choice for health. User can use result for this query to find the frequency of each food appear in nutrition consumption records and modify the diet (switch to similar food).



For this query, I left join NutritionConsumption table and Food table. I need date from both table because the NutrietionConsumption table has food consumption records which provide the frequency of food. Food table provides name of foods which is very important. I used left join because I only want to count food appears in Nutrition Consumption entity, I don’t want to count food in food table but not have been eaten yet. I used count aggregate function to count the number of foods. The count function accompanied with GROUP BY cluse make the SQL count number for each food. I also used the ORDER BY DESC clause to order the result by count of foods. From the query result we can tell that the user eats beef tender for most of time, so, the user may want to eat more lamb or chicken in the future.

1. For the database, I want to know which day the user didn’t finish the protein target and the reason for that based on the description. This question is very important. Although the plan is detailed enough for people to follow, but user still cannot finish the goal sometimes. The reason for that can hardly be gotten from the numerical data in database, that’s why we need to find the passible reasons from description.



For this query, I join 4 tables: Day,Target, WorkOutDay and RestingDay. Because I need date and protein consumption information from Day table, I need target information from Target table, I need description from WorkOutDay Table, and I need information from Resting Day table. I use an aggregation function to get the result from protein eaten minus protein target, if the result is positive or zero then the goal is finished, if the result is negative then the goal is not finished. Then I use a WHERE clause to only print negative result (which means the goal is not finished). The query answers many questions such as, how many days with unfinished goals, the detailed information about each day, how much more protein needed to finish the day etc. From the query result, we can see, there are 4 days with unfinished goal so far. The day with date 17 NOV 2019 is a resting day, so it is understandable why the goal was not finished. For rest of days, there are descriptions indicating the possible reason for unfinished goals.

**Summary and reflection:**

My database is for recording daily diet. Compare nutrition goal and nutrition consumption for users who wants to build muscle. In general, the database records the daily diet, sum the amount of daily nutrition up, records user’s health condition to offer information to user to modify his or her exercise and eating schedule. The database should support a user accessing, and searching information.

For this iteration, I finally implement the database. It feels so good to finish a project from the beginning to the end and implement it. I have learned so much from the beginning of the project. For now, I’m not 100 percent proficient in database design, I hope there will be more opportunities for me to practice this skill.