Project Iteration 4

Junyu Mou

**Adding attributed to DBMS Physical ERD**

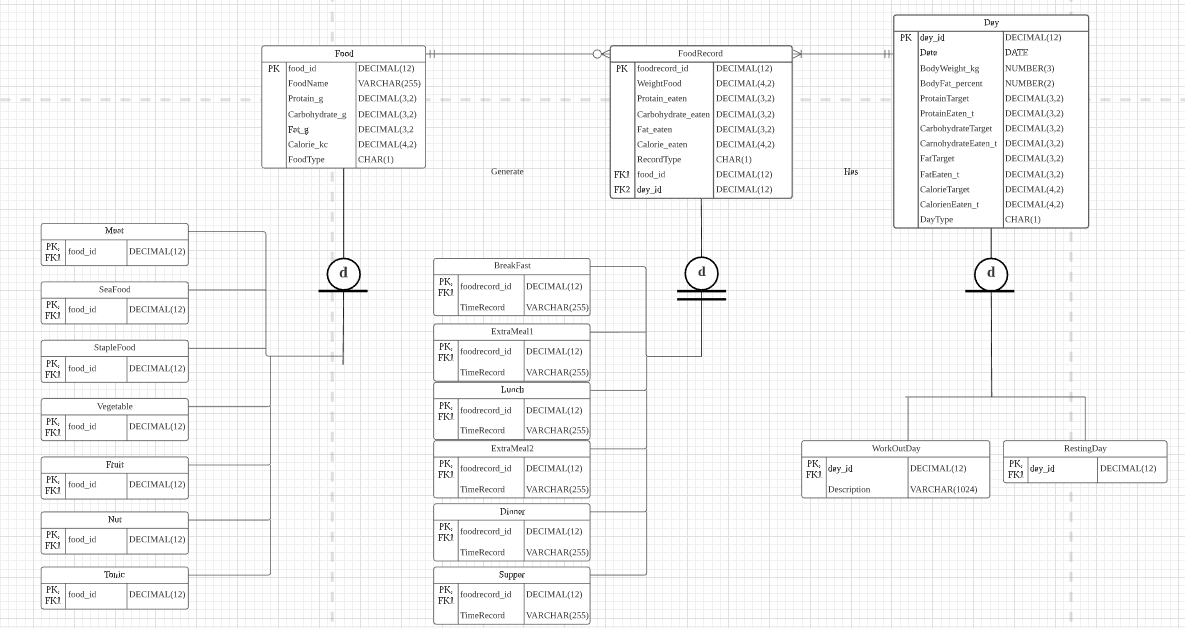
1. Driving questions:

|  |  |
| --- | --- |
| Questions | Reasoning |
| What field do other similar applications and databases store? | This database is made for tracking users’ daily nutrition, this database should have function of food library, recording every food eaten and record daily nutrition. Based on my experience with app named Peppermint nutritionist (which is an app acts like food library), each food needs a name, nutrition facts for unit weight, GI or GL value for each food per unit weight and diet recommendation.  Apple’s build-in health app is a similar application which can record daily nutrition. It can make a retailed record including many kinds of nutrition. It can show the nutrition in a linear group in scale of day, week or month. |
| What fields are obvious for my entities? | For my database, I would like to have the similar food nutrition library just like Peppermint nutritionist. So, my database will record food name and nutrition facts. But I will not include GI and GL values and diet recommendation. The food record in my database is for users to find food easier, but not to get knowledge.  Like ios’s health app, my database will record nutrition for everyday but just for protein, fat, carbohydrate and calorie. My database will record amount of nutrition based on the true weight of the food, and in order to achieve that, my database will need to record the true weight for food eaten by user. |
| What fields are unique for my database? | My database will have daily nutrition target for being compared with daily nutrition consumption by user. The user can check whether he or she finish the day’s target by comparing the nutrition target with nutrition assumption to modify his or her every schedule. |
| What would be presented on a user interface that uses my database? | The user will have the place to enter the food name and weight for food record. Also, the user would be required to enter personal information such as body weight, body fat, and information such as date. |

1. Attributes

|  |  |  |  |
| --- | --- | --- | --- |
| TABLE | ATTRIBUTE | DATATYPE | REASONING |
| Food | FoodName | VARCHAR (255) | Every food has a name which acts like the identifier for the product when people are looking it up. I allow for up to 255 characters just in case of something with long name. |
| Food | Carbohydrate\_g | DECIMAL (3,2) | For this database, I’m trying to save nutrition facts for every 100g of food. There is hardly any food can have more than 99.99g carbohydrate per 100g. For this attribute I allow 3-digits just in case of something can have 100g carbohydrate per 100g. |
| Food | Protain\_g | DECIMAL (3,2) | For this database, I’m trying to save nutrition facts for every 100g of food. There is hardly any food can have more than 99.99g protain per 100g. For this attribute I allow 3-digits just in case of something can have 100g protain per 100g. |
| Food | Fat\_g | DECIMAL (3,2) | For this database, I’m trying to save nutrition facts for every 100g of food. There is hardly any food can have more than 99.99g fat per 100g. For this attribute I allow 3-digits just in case of something can have 100g fat per 100g. |
| Food | Calorie\_kc | DECIMAL (4,2) | Compare to carbohydrate, protein and fat, calorie can be larger. For example, 100g certain brand chololate can have 821 kilocalories. Even calorie can be a large number, 3-digit would be enough, I allow 4-digit just in case there are food exists has extreme large calorie |
| Food | FoodType | CHAR (1) | In the prior iteration, I indicated that there can 7 types of foods. This attribute is the subtype discriminator to indicate which it is. |
| FoodRecord | WeightFood | DECIMAL (4,2) | This is the real weight food. For example, if a user eats 250g steak for lunch, the real weight for the steak is 250g. This attribute records the value of the food which can large. I allow 4-digits for this attribute. |
| FoodRecord | Protain\_eaten | DECIMAL (3,2) | This attribute records protein eaten by user based on the weight of the food. I allow 3-digits in grams which is enough. |
| FoodRecord | Fat\_eaten | DECIMAL (3,2) | This attribute records fat eaten by user based on the weight of the food. I allow 3-digits in grams which is enough. |
| FoodRecord | Carbohydrate\_eaten | DECIMAL (3,2) | This attribute records carbohydrate eaten by user based on the weight of the food. I allow 3-digits in grams which is enough. |
| FoodRecord | Calorie\_eaten | DECIMAL (4,2) | This attribute records calorie eaten by user based on the weight of the food. Compare to nutrition like protein, calorie can be larger value, so I allow 4-digits in kilocalorie which will be enough. |
| FoodRecord | RecordType | CHAR (1) | In the prior iteration, I indicated that the all records are divided into 6-time intervals, such as breakfast, extra meal 1, lunch etc. This attribute is the subtype discriminator to indicate which it is. |
| Day | date\_day | DATE | This is the date for food records. |
| Day | BodyWeight\_kg | NUMBER (3) | This is the body weight in kg for the day. I allow 3-digits, no one would be heavier than 999 kg anyway. |
| Day | BodyFat\_percent | NUMBER (2) | The body fat is a percentage number represented in integer. For example, normal people have 20(%) body fat can be recorded as 20 body fat in this database. 2-digits number is enough, it is impossible for a person to have a 3-digits body fat. |
| Day | ProtainTarget | DECIMAL (3,2) | Protein target is the protein goal for the day. I allowed 3-digits number for this attribute. |
| Day | ProtainEaten\_t | DECIMAL (3,2) | This attribute records the total protein eaten for the day. I allow 3-digits for this attribute. |
| Day | CarbohydrateTarget | DECIMAL (3,2) | Protein target is the carbohydrate goal for the day. I allowed 3-digits number for this attribute. |
| Day | CarbohydrateEaten\_t | DECIMAL (3,2) | This attribute records the total carbohydrate eaten for the day. I allow 3-digits for this attribute. |
| Day | FatTarget | DECIMAL (3,2) | Protein target is the fat goal for the day. I allowed 3-digits number for this attribute. |
| Day | FatEaten\_t | DECIMAL (3,2) | This attribute records the total fat eaten for the day. I allow 3-digits for this attribute. |
| Day | CalorieTarget | DECIMAL (4,2) | Protein target is the calorie goal for the day. Even calorie can be larger than nutrition such as protein, 4-digits is still enough. |
| Day | CalorieEaten\_t | DECIMAL (4,2) | This attribute records the total calorie eaten for the day. I allow 4-digits for this attribute. |
| Day | DayType | CHAR (1) | In the prior iteration, I indicated that the daily records are divided into 2 types, work out day and resting day. This attribute is the subtype discriminator to indicate which it is. |
| WorkOutDay | Description | VARCHAR (1024) | Users may want to add description for work out day. For example, one day’s exercise can affect food consumption. I allow 1024 characters so people can type in something long. |
| BreakFast;  ExtraMeal1;  Lunch;  ExtraMeal2;  Dinner;  Supper | TimeRecord | VARCHAR (255) | For these tables, I would like to add a Time record attribute which allows the user to record time for each meal. I make it VARCHAR, because I think users may need a relatively loose standard for time records for each meal. The user doesn’t have to record time like 6:00, they can record the time such as early in the morning or very late. I allow 255 characters for this attribute just in case the long record. |

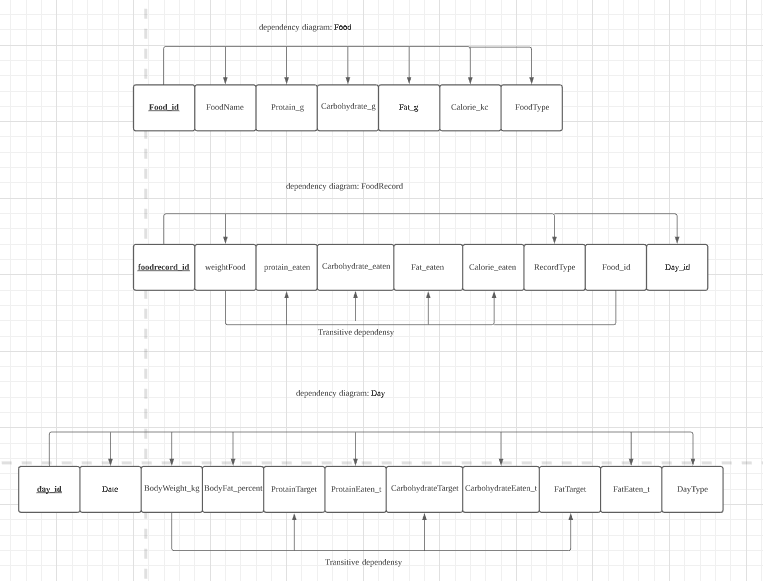
For this part, I tried to add sufficient attributes for the database for achieving detailed design. While I’m adding the attributes, I feel there will be data redundancy in my database especially in Food record entity. So, the changes will be made for the next section, the database needs to be normalized. After the attributes section is done, the ERD with the attributes will be made.



**Normalizing DBMS Physical ERDs:**

1. First normal form(1NF)

The first normal form is about table format. It should be guaranteed that no repeating groups and the primary should be identified. For my database there are no repeating groups, which means there will be no multivalued attributes. For each table, all primary keys are valid, for example, all primary keys from entities will have unique value. In Order to make the normalizing process clear, I made dependency diagram for each table:

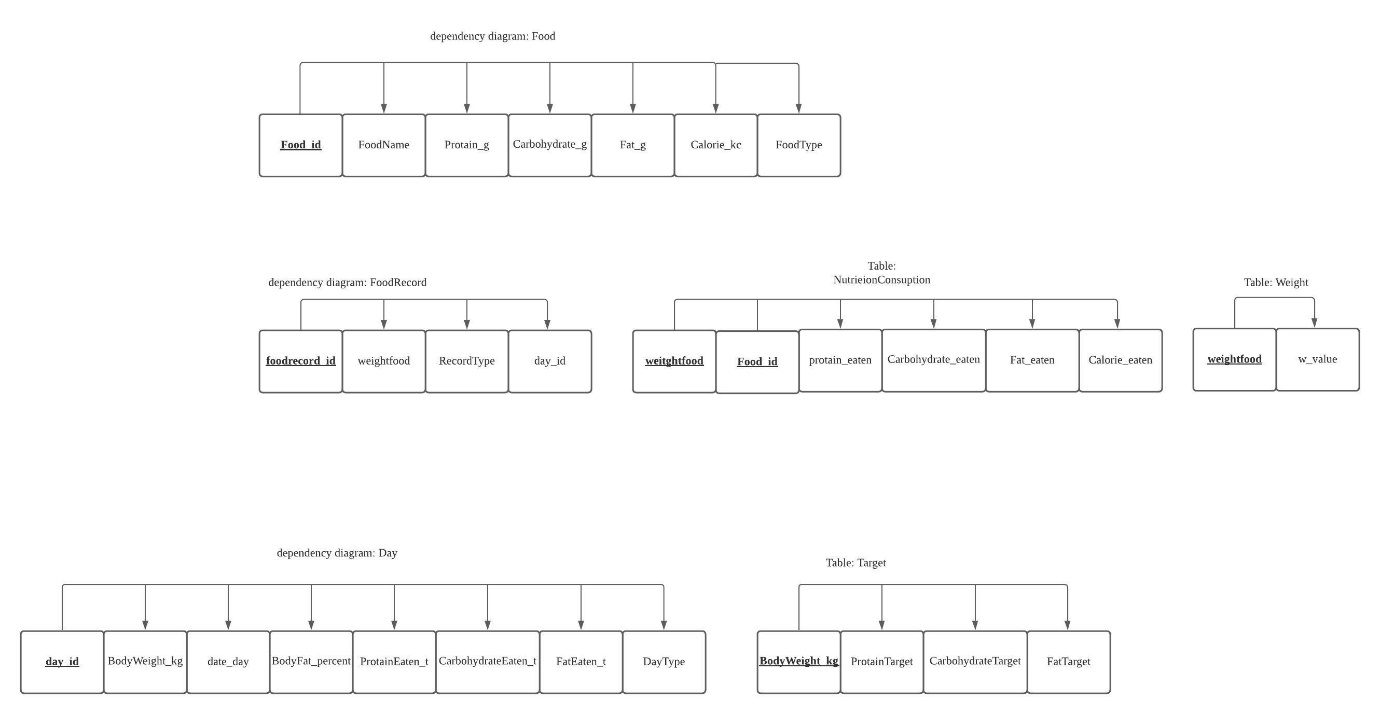


1. Second normal form(2NF)

The second normal form can eliminate partial dependencies further based on the result of 1NF. A partial dependency exists when there is a functional dependence in which the determinant is only part of the primary key. As long as there is no partial dependency for all of tables, this step will not make any changes.

1. Third normal form(3NF)

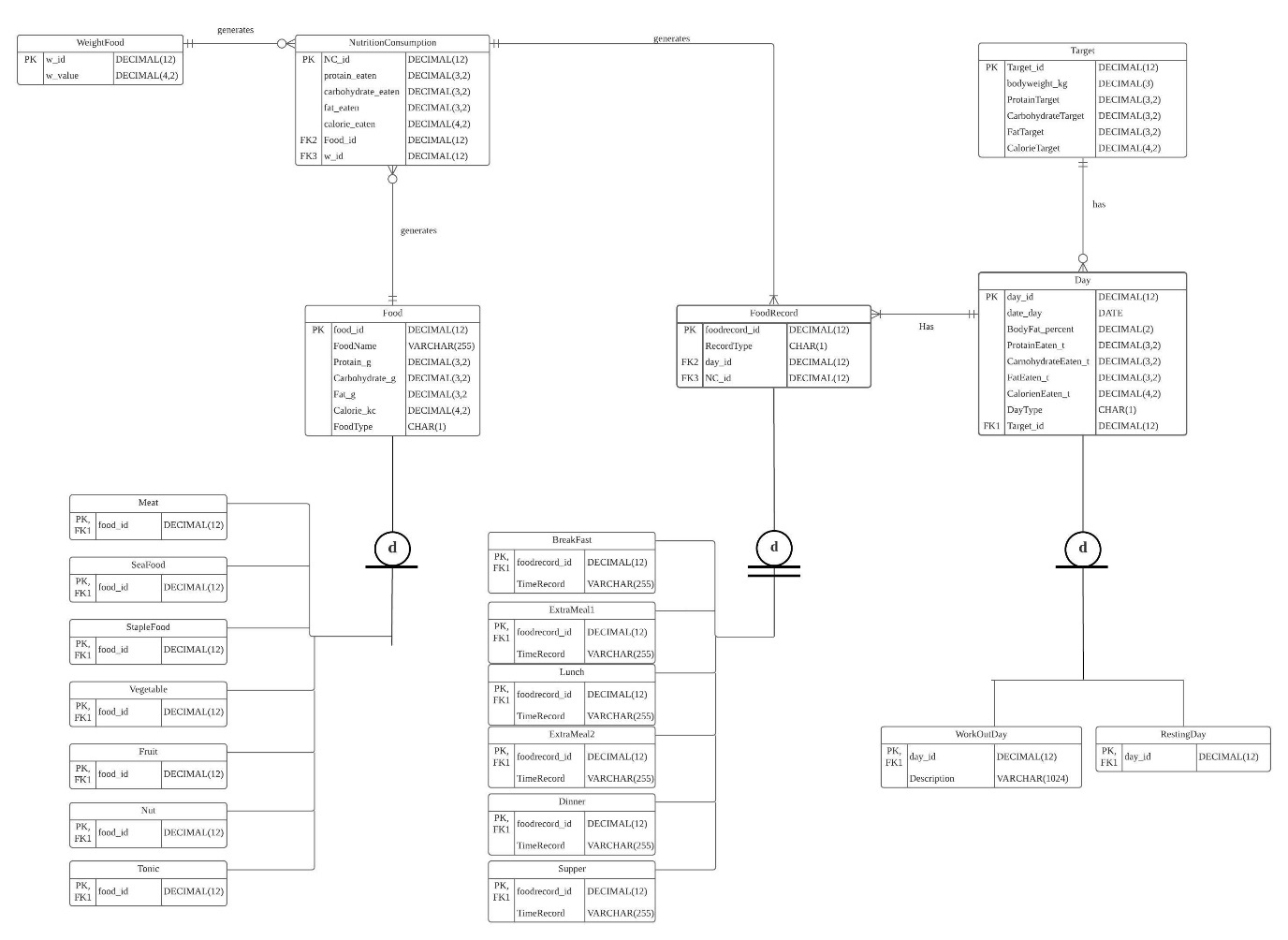
The third normal form eliminates transitive dependencies further based on the result of 2NF. A transitive dependency exists when there are functional dependencies such that X->Y, Y-> Z, and X is the primary key. For my database so far, there are transitive dependencies appear in table Day and food record. The tables will be modified to eliminate transitive dependencies.



For the result after 3NF, the original entities are modified to eliminate transitive dependency.

1. Boyce-Codd normal form (BCNF)

BCNF is done when the determinant in the table is a candidate key which is a unique type of super key without useless attribute. That means the database is already BCNF.



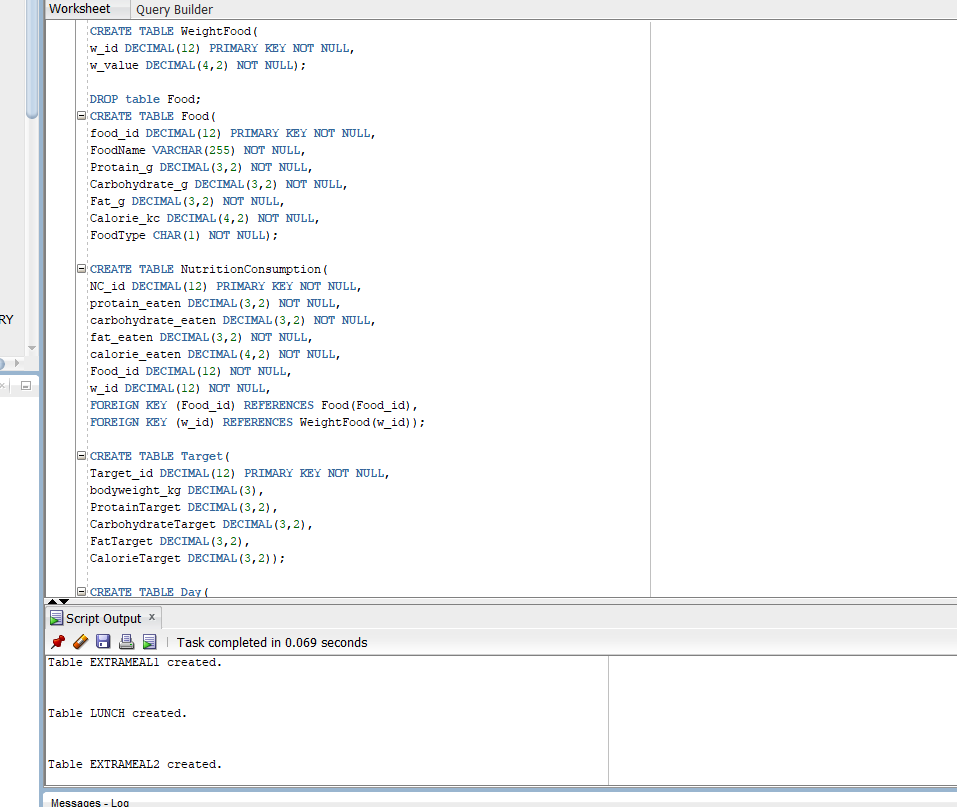
For the normalization process, I modified and changed some of the features in old version. Three new entities are added to make the project reasonable. By adding WeightFood entities user doesn’t have to input the weight of food every time. This entity may be confusing in the real-life cause not so many people care about the weight of the food before they eat, they don’t consider about eating a specific amount of food neither. But for people who has training goal, knowing weight before eat is essential. For example, I can arrange my diet like: 100g meet for lunch, I can switch from beef to chicken without changing the weight of food because nutrition facts for beef and chicken is similar. I add a NutritionConsumption entities which has the nutrition facts based on weight of food and food. I obtain this entity by splitting the old food record entity. For the old version, if the food and weight is the same, there will be many same rows generated in FoodRecord entity and that can cause data redundancy. I added a Target entity to reduce redundancy for Day entity. Target entity has nutrition target which are only depend on body weight. Body weight can be changed by weeks, so the old version of Day entity had may rows with same body weight and target which could cause data redundancy.

After the normalization process, the structural database rules are changed as well:

1. Each Food may products many nutrition consumption record; each nutrition consumption record only has one food.
2. Each Weight may products many nutrition consumption record; each nutrition consumption record only has one Weight.
3. Each nutrition consumption record generates one or more food record; each food record has only one nutrition consumption record.
4. Each day has one or more FoodRecords; each FoodRecord is for one day.
5. Each Target may be had by many Day; each Day only has one Target.
6. A food is a meat, a seafood, a staple food, a vegetable, a fruit, a nut, a tonic or a none of these
7. A day is a work out day, a resting day or none of these
8. A food record is a breakfast record, an extra meal 1 record, a lunch record, an extra meal 2 record, a dinner record or a supper record.

For my database, I have 21 entities which more than the minimum requirement for the project.

**Creating tables from a DBMS physical ERD:**

****

This screen shots just show part of the SQL code, all 21 tables are successfully created with proper attributes and constrains.

**Indexing the database:**

1. All primary keys are automatically indexed in modern DBMS:

|  |  |
| --- | --- |
| Primary Key column | Description |
| WeightFood.w\_id | Primary key for table WeightFood. |
| NutritionConsumption.NC\_id | Primary key for table NutritionConsumption |
| Food.food\_id | Primary key for table Food |
| FoodRecord.foodrecord\_id | Primary key for table FoodRecord |
| Target.Target\_id | Primary key for table Target |
| Day.day\_id | Primary key for table Day |
| Specialization tables: foodrecord\_id | Primary key for Specialization tables: BreakFast, ExtraMeal1, Lunch, ExtraMeal2, Dinner and Supper |
| Specialization tables: food\_id | Primary key for Specialization tables: Meat, SeaFood, StapleFood, Vegetable, Fruit, Nut and tonic |
| Specialization tables: day\_id | Primary key for Specialization tables: WorkOutDay and Resting Day |

1. Adding foreign key indexes:

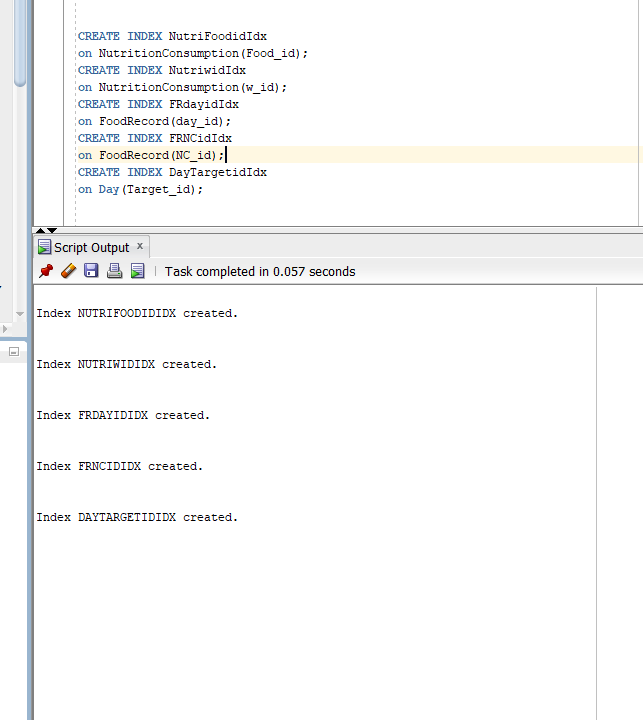
|  |  |  |
| --- | --- | --- |
| Column | Unique? | Description |
| NutritionConsumption.Food\_id | Non-unique | This foreign key in the NutritionConsumption table references the Food table. The index is non-unique since many nutrition consumption records can be generated by same food. |
| NutritionConsumption.w\_id | Non\_unique | This foreign key in the NutritionConsumption table references the WeightFood table. The index is non-unique since many nutrition consumption records can be generated by same weight of food. |
| FoodRecord.day\_id | Non-unique | This foreign key in the FoodRecord table references the Day table. The index is non-unique since many food records can be in one day. |
| FoodRecord.NC\_id | Non-unique | This foreign key in the FoodRecord table references the NutritionConsumption table. The index is non-unique since many food records can be generated by same nutrition consumption record. |
| Day.Target\_id | Non-unique | This foreign key in the Day table references the Target table. The index is non-unique since many days can have the same target. |

1. Query driven indexes

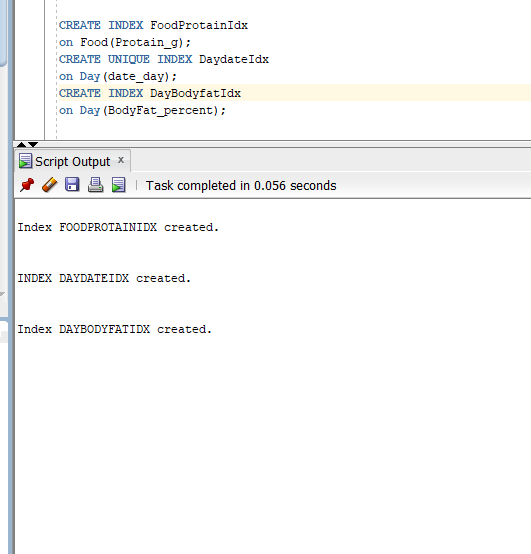
|  |  |  |
| --- | --- | --- |
| Column | Unique? | Description |
| Food.protain\_g | Non-unique | It is reasonable to index protein for each food, because protein is the most important factor for user to choose food. The user can find the food with high protein proportion. For example, the user wants to find foods with protein more than 25(g) per unit weigh. I made this non-unique just for special case, even though it is hardly possible for more than two kinds of food have the same protein content. |
| Day.date\_day | Unique | It is reasonable that the date of the day will be a limiting column in queries, because reports and analysts will commonly want to limit analysis by date range. The user may want to inspect the records in a particular year, month or week. This is unique, because each day can only have one date. |
| Day.BodyFat\_percent | Non-unique | The user may wonder about how the diet influence the body fat, for example, the user wants to know what he or she eat for days the body fat is just 10(%). For my database, I allow body fat to be null, I think there won’t be so many null values, so it is still valuable to index this column. This would be non\_unique because the user can have the same body fat for everal days. |

**Creating indexes in SQL:**

1. Indexes for foreign keys:



1. Query driven indexes:



**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 spent most of time on normalization process. Before finished the normalization, I felt that I was not very familiar with normalization process, although I read and understand the text book. Reading and knowing is far different from doing. After this iteration I feel more confident for normalization process.

After this iteration is done, I find that before the normalization process the database design is kind of messed up, even I thought the design was good before this iteration. It is kind of creepy to change the original design (not just modify), I change the logical a bit which could be time consuming.

The implementation part is exciting. I hate errors when I’m coding, so I tried to make every step correct. There is a mistake during the table creation, but I figure it out.