JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Hike Planner

Course Section: CS605.641.83
Summer, 2018

Prepared by

Anthony Gray

08/10/2018

Database Design Project Document

Table of Contents

Introduction

Backpackers need to carry enough gear to be safe and comfortable, but also need to carry food and water. The food needs to be lightweight and easy to prepare, but it also has to be nutritious enough to make up for the energy spent carrying it. It might also all have to fit into a bear can, depending on local regulations. Additionally, its only considered safe for an individual to carry 20% to 30% of their body weight on their back. The result is an optimization problem where a backpacker attempts to carry as much high-value food as they can, and thereby stay in the wilderness for as long as possible without sacrificing necessary gear.
The purpose of the Hike Planner application is to provide a calculator that contains information about an individual, specific pieces of gear, certain types of food and different locations. It can be used to generate packing lists, along with statistics like total weight and calorie content of a meal plan. Additional requirements can be displayed to the user, such as gear regulations that affect food choices.

1.1. Scope and Purpose of Document

This document contains information on the purpose of the Hike Planner application, as well as the details and rationales of several design choices. It is important to note that the application has many possible points of expansion. At present, it is intended as an application for personal use, and will therefore not feature multiuser functionality or a Web implementation. It also assumes some degree of backpacking experience of the user and is not set up to prevent actions by the user, or to automatically take any action. Notes for particular entity types are included as simple strings displayed to the user, and it is up to the user to notice, remember and respond to them. Should the application be extended, these safeguards are the first likely avenues.

1.2. Project Objective

The objective of this project was to implement a small personal database application that could potentially become the core of a larger application. The design and implementation of a database that can capture the relevant details of planning a backpacking list is the most important part of such an application. Information regarding the food, gear, and locations involved in a backpacking trip are necessary, as well as the ability to organize that information into useful forms such as packing or shopping lists.

2. System Requirements

The requirements to run this application are fairly minimal. At the moment, it is intended for individual use, so it can work on most personal computers. The database is implemented with MySQL 8.0.12, and the interface application is a console-based prompt system implemented with Python 3.6.5. Python is lightweight enough to operate on all but the most obsolete computers, while MySQL uses amounts of resources scaling directly with the amount of data stored. As a small personal application, this won't be a lot, and if it is ever extended to a web application, it will only require a working web browser. However, the database was implemented with the help of MySQL Workbench 8.0.12, an open-source RDBMS. Even only with personal use, the application benefits from MySQL Workbench, which does have stricter requirements.

2.1 Hardware Requirements

MySQL Workbench requires at least a 3GHz single processor or a 2GHz dual core processor. 4 GB of RAM, a GPU supporting OpenGL 1.5, and a display resolution of 1280x1024 are also required. A dual/quad core processor, 6 GB of RAM, and 1920x1200 display resolution are recommended. Obviously, these are minimum requirements, as the users operating system must still be able to run.

2.2 Software Requirements

The application was implemented on a machine running Windows 10. However, MySQL works on a wide range of operating systems. A complete list can be found here. In brief, and recent Windows or Mac OS system should work, as will some of the more common Linux distributions such as Ubuntu. MySQL Workbench is helpful, but not required to run the application, and a Python based interface is included as well. Should the application ever be migrated to the web, a web browser will be necessary, most likely Google Chrome.

2.3 Functional Requirements

The database needs to be able to store information about food, hiking gear, and locations. It needs to be able to organize food information into meals and meals into meal plans, gear information into packing lists, and assemble location data into useable itineraries. Additionally, it needs to be able to answer queries such as the most used food item or food with the most protein content.

2.4 Database Requirements

The database is implemented with MySQL 8.0.12, and the interface application is a console-based prompt system implemented with Python 3.6.5.

3. Database Design Description

There are four basic types of entity involved in this project, Hikers, Food, Gear, and Locations. The Hiker is straightforward, and only exists to calculate a total weight limit based on its body weight. This is achieved with a simple 25% calculation. More sophisticated methods exist, but they involve complex formulas, much more information, and a means to measure experience. This would be extremely difficult to implement accurately, and the 25% method is a good estimate that serves its purpose within the context of the application.

The second, and most important entity is food. Nutritional information about different types of food is stored and can arranged into meals. Meals can be assigned to days, and days arranged into meal plans that can in turn generate shopping lists. Facts like a meal plan's total weight, or the total calorie content of a planned day can also be retrieved.

Gear is the third entity type and is intended to include all the non-food items that a backpacker carries with them. The main purpose of the gear is to subtract from a hiker's weight capacity to provide a more accurate limit on the weight of the food that can be carried. To this end, gear can be assembled into packing lists with associated total weights. Additionally, gear has types, such as tents, stoves, and so forth. Certain types of gear can necessary, such as a stove for cooking pasta, or subject to restriction based on location. The application will not enforce any of these limitations, as following them is up to the judgement of the hiker. Functionality along these lines could be added in the future if the application is extended.

The final entity type is location. Locations are used to build itineraries and can be associated with features like trailheads and ranger stations. They also exist in regions like national parks, which may have gear restrictions. Of all the entities, locations have the most potential for expansion as actual GPS location data could be implemented. However, there are already applications in existence that contain trail data, and though they are used primarily for navigation, they also play a role in planning. Considering the scope of this application, real-world location data will be omitted, as it can be accessed via other applications. However, this application can be effectively paired with those focused on navigation by the user.

3.1 Design Rationale

This ER design is set up to handle relatively small amounts of interconnected data. To that end, many of the tables only have a handful of columns and are set up to join to the others easily. In fact, only six of the sixteen tables in the database lack foreign keys from any other table, and several contain one or less native variables, being comprised mostly of foreign keys. The extensive use of joins is acceptable due to the fact that the application expects to handle fairly shallow tables. Most hikers own less than 100 unique pieces of gear, and entities like gear_type are essentially finite. There are only so many types of hiking gear. The location portion of the database could be a bit more extensive, but most hikers plan trips near their homes in the same parks or wilderness areas. This database functionality is not intended to stand alone regardless, as it is included to inform food and gear choice before being paired with a third-party application for actual navigational purposes. The food section has the most potential for depth, as there are a great number of variations on the same kind of food product from many different brands, but again these options are limited by an individual user's geography. Essentially, the applications main goal is to compare data, and the performance costs of many join operations are not expected to be noticeable.

3.2 E/R Model

In general, this design is highly normalized, with the notable exception of gear_restrictions. In this case, a table containing the possible restrictions could have been implemented, but instead the input space for the restriction variable is limited to four choices via the ENUMERATE function. This choice was made because there were only four options, and yet another join was not necessary. On the other hand, gear_type began as a part of the gear entity and was removed when the enumerated option list grew too long to easily keep track of. Additionally, while there are only four possible restrictions a location can place on a type of gear, the types themselves are difficult to keep track of, as technologies change, and new products are released to market. This ER model also makes extensive use of automatically incrementing primary keys. Again, this is to increase the ease of performing join operations and to keep each entry as flexible as possible in its relations with other entities. There are a few notable exceptions, however, where other keys or combinations of keys are used to enforce uniqueness in N:M relations. For example, a region such as a national park has exactly one rule or lack of a rule regarding any particular type of gear, so paired foreign keys are used as a primary key. This prevents the same rule from being added to the same location twice, and a similar strategy is used to ensure that a piece of gear will only have one rule associated with it for a given location, and that it will only appear once on any given packing list.

3.2.1 Entities

Food – Food contains information about a specific food product, including the name, brand and whether it needs to be cooked. Additionally, nutritional information, including weight, calories and protein content per serving are included, as well as the number of servings per package. This information allows the user to calculate and sort based on these factors.

Meal – Meals are an entity to contain an organize food. A meal is named and consists of one or more food types in any quantity.

Day – Days are an entity to organize meals. Each day can associate up to four Meals one each for breakfast, lunch, dinner and snacks.

Meal Plan – A meal plan is a series of Days planned for a trip. It has a name and can associate with any number of Days. It can also be assigned to an Itinerary.

Hiker – The hiker represents the human carrying the backpack. There are several variables, but the only ones utilized at present are name and weight. The name is to indicate the owner of an itinerary or packing list, and the weight is to calculate a maximum carrying capacity. Budget and age could be useful in the future, as age can be used to adjust the carrying capacity along with weight. If the application ever gains pricing information for gear and food, budget could be useful as well, but for now these variables are not used.

Packing List – A packing list is owned by a Hiker and has its own name corresponding to a trip or objective. Gear items can be associated with it.

Gear – Gear represents all the non-food items in a backpack. Like food, they also have a name brand and weight, and are assembled into packing lists to calculate a base weight. Additionally, Gear items have a Gear Type as well, which can be used to fill out a packing checklist or check to see if a packing list is legal in a certain area.

Gear Restrictions – Gear Restrictions refer to the rules regulating the use of certain types of gear. For example, a bear can is required, while commercial drones are banned in many National Parks.

Gear Type – Gear Type refers to the function a piece of gear performs, such as providing shelter or cooking food. Gear Restrictions apply to Gear Types, rather than specific Gear items.

Itinerary – Itineraries are loose outline of a planned trip. They have an owning Hiker, a start and end Location, and a length in days. They can also associate with a Meal Plan, presumably of equal length in days, though this is not required.

Location – A Location represents a specific waypoint that could potentially be modeled by GPS information should the application be extended significantly. For now, it has a name, contains a Feature and is a part of a Region.

Feature – A Feature can be any physical place of note and has a relationship with Location like the relationship between Gear and Gear Type. A feature could be a trailhead, or spring, parking lot, or viewpoint. It could also be a town, post office or road. In all cases, Locations can reference the entry in Feature for a classification and to avoid data duplication, as many trailheads and roads exist.

Region – Regions refer to a large geographic area that includes many locations, such as a National Park or National Forest. Each Region has a name and owner and can be associated with any number of Gear Restrictions.

3.2.2 Relationships

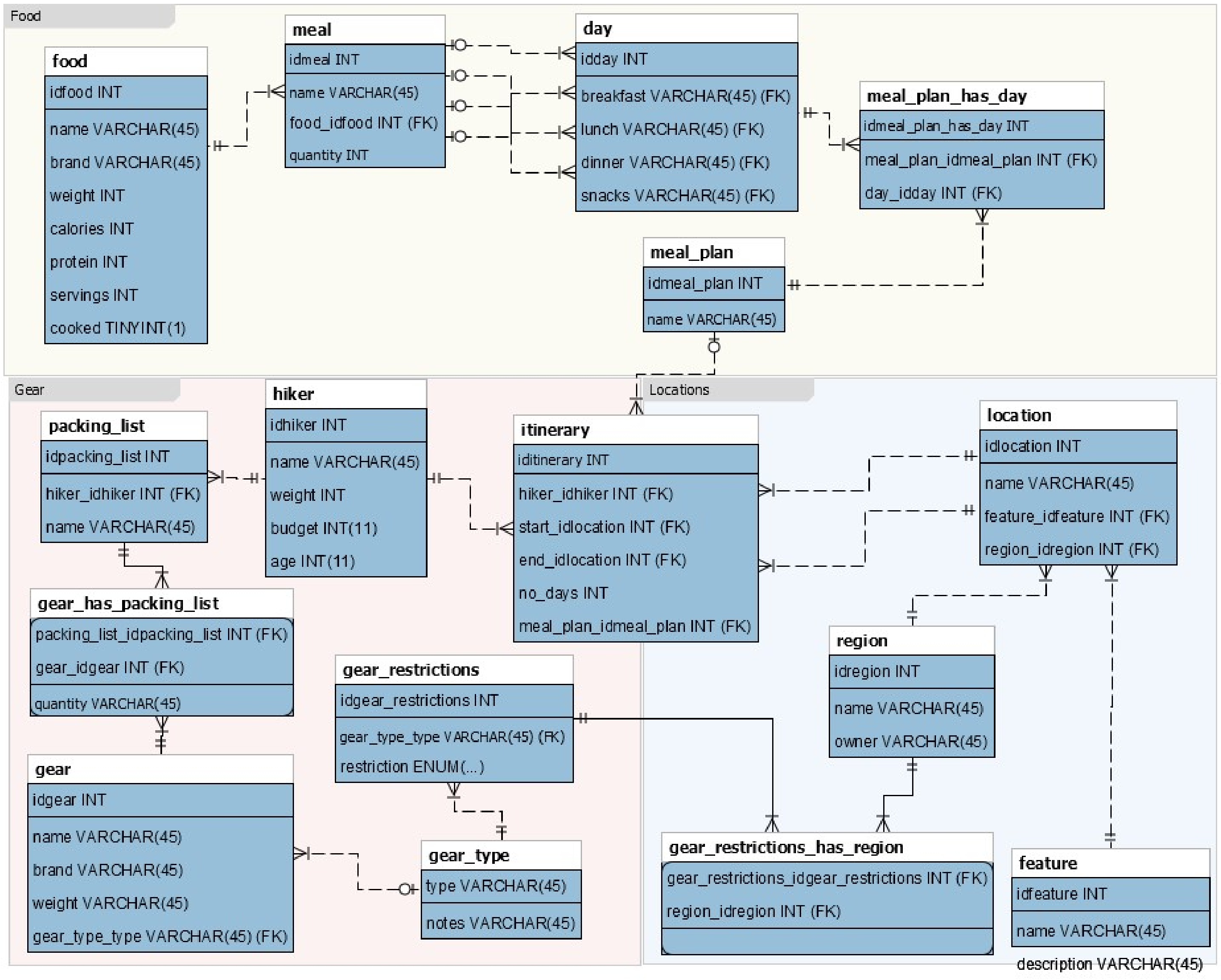
Meal Plan to Day – Many Days can be assigned to any number of Meal Plans. The same day could appear in different Meal Plans, or even multiple times in the same one. As Days are essentially containers for Meals and by extension Food, they provide a convenient way to plan and store a day's worth of food. They can then be assembled into as many Plans as needed. Note that the order of Days in a Meal Plan does not matter. The hiker will have the food in their pack and can eat it in any order. The Days in a Meal Plan are for calculating total calories and weight, rather than explicitly laying out a menu.

Meal Plan to Meal – Ultimately replaced Meal Plan to Day

Gear to Packing List – Gear can be assigned to a Packing List in much the same way Days can be assigned to a Meal Plan. The difference is that only one of each Gear item can be assigned to a given Packing List. This is largely because it is very unusual for a backpacker to bring more than they need, and the restriction makes the implementation simpler. There is a quantity variable that can be updated if needed, but it defaults to one.

Gear Restrictions to Region – It is unusual to find a Region with no gear regulations, and most have several. Additionally, many of the regulations are the same, such as requiring bear cannisters to camp in certain parks or recommending that hikers carry water purification systems to avoid illness. This is a straightforward many-to-many relationship where each Region can have any number of Restrictions, while each Restriction can apply to several Regions. Each Region is associated with exactly one of each Restriction, as there is no need for repetition.

3.2.3 E/R Diagram



3.3 Relational Model

3.3.1 Data Dictionary

Field	Description	Type	Null	Key	Default	Extra
idfood	Food key	int(11)	NO	PRI		auto_increment
name	Name of food	varchar(45)	NO			
brand	Brand of food	varchar(45)	YES		“Unkown”	
weight	food(g/serving)	int(11)	NO			
calories	Calories/serving	int(11)	NO			
protein	Protein g/serving	int(11)	NO			
servings	Servings per package	int(11)	NO			
cooked	Boolean	tinyint(1)	NO		0	
idmeal	Meal key	int(11)	NO	PRI		auto_increment
name	Meal name	varchar(45)	NO	MUL		

food_idfood	Food foreign key	int(11)	NO	MUL		
quantity	Servings per meal	int(11)	NO			
idday	Day key	int(11)	NO	PRI		auto_increment
breakfast	Meal foreign key	varchar(45)	YES	MUL	0	
lunch	Meal foreign key	varchar(45)	YES	MUL	0	
dinner	Meal foreign key	varchar(45)	YES	MUL	0	
snacks	Meal foreign key	varchar(45)	YES	MUL	0	
idmeal_plan_has_day	Meal FK	int(11)	NO	PRI		auto_increment
meal_plan_idmeal_plan	Meal plan FK	int(11)	NO	MUL		
day_idday	Day FK	int(11)	NO	MUL		
idmeal_plan	Meal plan key	int(11)	NO	PRI		auto_increment
name	Meal plan name	varchar(45)	NO	UNI		
idhiker	Hiker key	int(11)	NO	PRI		auto_increment
name	Hiker name	varchar(45)	NO			
weight	Hiker weight (kg)	int(11)	NO			
budget	Hiker budget	int(11)	YES		0	
age	Hiker age	int(11)	YES		0	
idpacking_list	Packing list FK	int(11)	NO	PRI		auto_increment
hiker_idhiker	Hiker FK	int(11)	YES	MUL	0	
name	Packing list name	varchar(45)	NO			
packing_list_idpacking_list	Packing list FK	int(11)	NO	PRI		
gear_idgear	Gear FK	int(11)	NO	PRI	0	
quantity	Number of items	varchar(45)	NO		1	
idgear	Gear key	int(11)	NO	PRI		auto_increment
name	Gear name	varchar(45)	NO			
brand	Gear brand	varchar(45)	YES		“Unkown”	
weight	Gear weight (g)	varchar(45)	NO			
gear_type_type	Gear type FK	varchar(45)	YES	MUL	"Luxury"	
type	Type key	varchar(45)	NO	PRI		
notes	Type description	varchar(45)	NO			
idgear_restrictions	Restriction key	int(11)	NO	PRI		auto_increment
gear_type_type	Gear type FK	varchar(45)	NO	MUL		
restriction	Type of restriction, chosen from 4 options	enum('Required','Prohibited','Reccomended','Discouraged')	NO			
gear_restrictions_idgear_restrictions	Restriction FK	int(11)	NO	PRI		
region_idregion	Region FK	int(11)	NO	PRI		
idlocation	Location key	int(11)	NO	PRI		auto_increment
name	Location name	varchar(45)	NO			auto_increment
feature_idfeature	Feature FK	int(11)	YES	MUL	0	
region_idregion	Region FK	int(11)	YES	MUL	0	
idfeature	Feature key	int(11)	NO	PRI		auto_increment
name	Feature name	varchar(45)	NO	UNI		

description	Feature notes	varchar(45)	NO			
iditinerary	Itinerary key	int(11)	NO	PRI		auto_increment
hiker_idhiker	Hiker FK	int(11)	YES	MUL	0	
start_idlocation	Start location FK	int(11)	YES	MUL	0	
end_idlocation	End location FK	int(11)	YES	MUL	0	
no_days	Days of trip	int(11)	NO			
meal_plan_idmeal_plan	Meal plan FK	int(11)	YES	MUL	0	
idregion	Region key	int(11)	NO	PRI		auto_increment
name	Name of region	varchar(45)	NO	UNI		
owner	Owner of region	varchar(45)	NO			

3.3.2 Integrity Rules

The majority of the columns in the database have the NOT NULL constraint applied. The fields that do no have NOT NULL applied to them are usually foreign key fields or less commonly used variables. A default value is specified in these cases, usually 0 for the foreign key fields.
Any reference field also uses the FOREIGN KEY tag specifically referencing a key field of another entity. The foreign key constraint helps to maintain the relationship between fields by ensuring that the fields are equivalent and taking automatic action if one side of the relationship is disrupted. There are a few cascade triggers, such as between Food and Meal, but in the majority of cases no action is taken if a foreign key is deleted.
The controller interface also plays a role in maintaining data integrity. The inputs are validated for the proper type, and in order to perform an UPDATE operation, information for all columns in an entry must be provided.

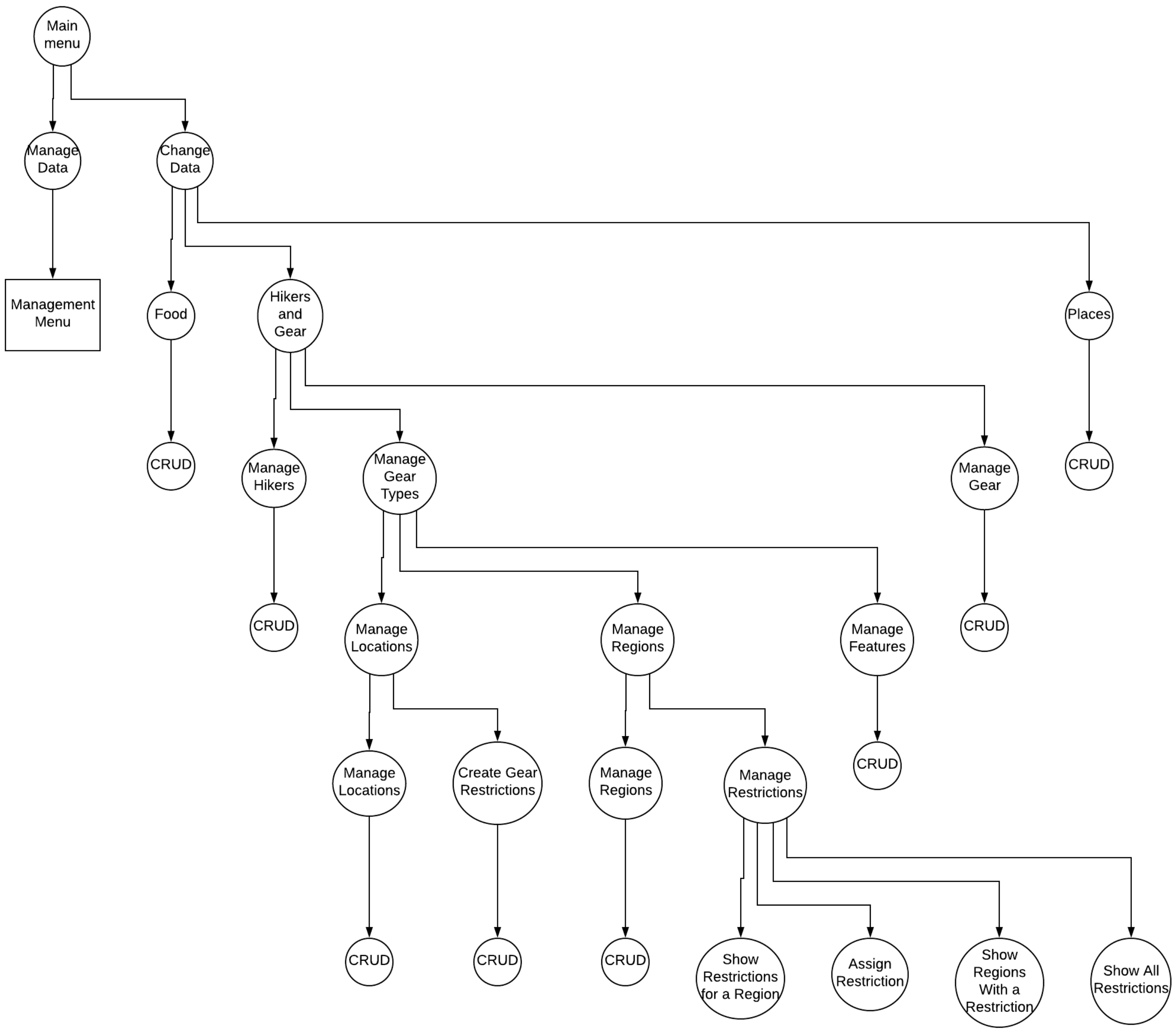
3.3.3 Operational Rules

A user has the ability to insert, modify and remove information from every table in the database. They can insert and manipulate an unlimited number of Food items, Gear items, and Hikers, and assemble them into as many Meal Plans and Packing Lists as they choose. They can also use Features and Regions to generate as many Locations to use in their Itineraries as they need. The specific queries available to a user are limited by the controller interface, but these queries allow a great deal of freedom in terms of manipulating the database.
That said, there are restrictions in place to eliminate duplicate entries in many tables. Meal, Day, Hiker, Gear, and Gear Restrictions all have a Unique Index in place to prevent duplicate rows. Meal Plan, Region, Gear Type and Feature are restricted to unique values in the name column to avoid multiple entries for the same thing. Finally, Gear_Has_Packing_List, Gear_Restrictions and Gear_Restrictions_Has_Region all use a composite key comprised of the foreign key from each entity to ensure that each relationship is represented by exactly one row.

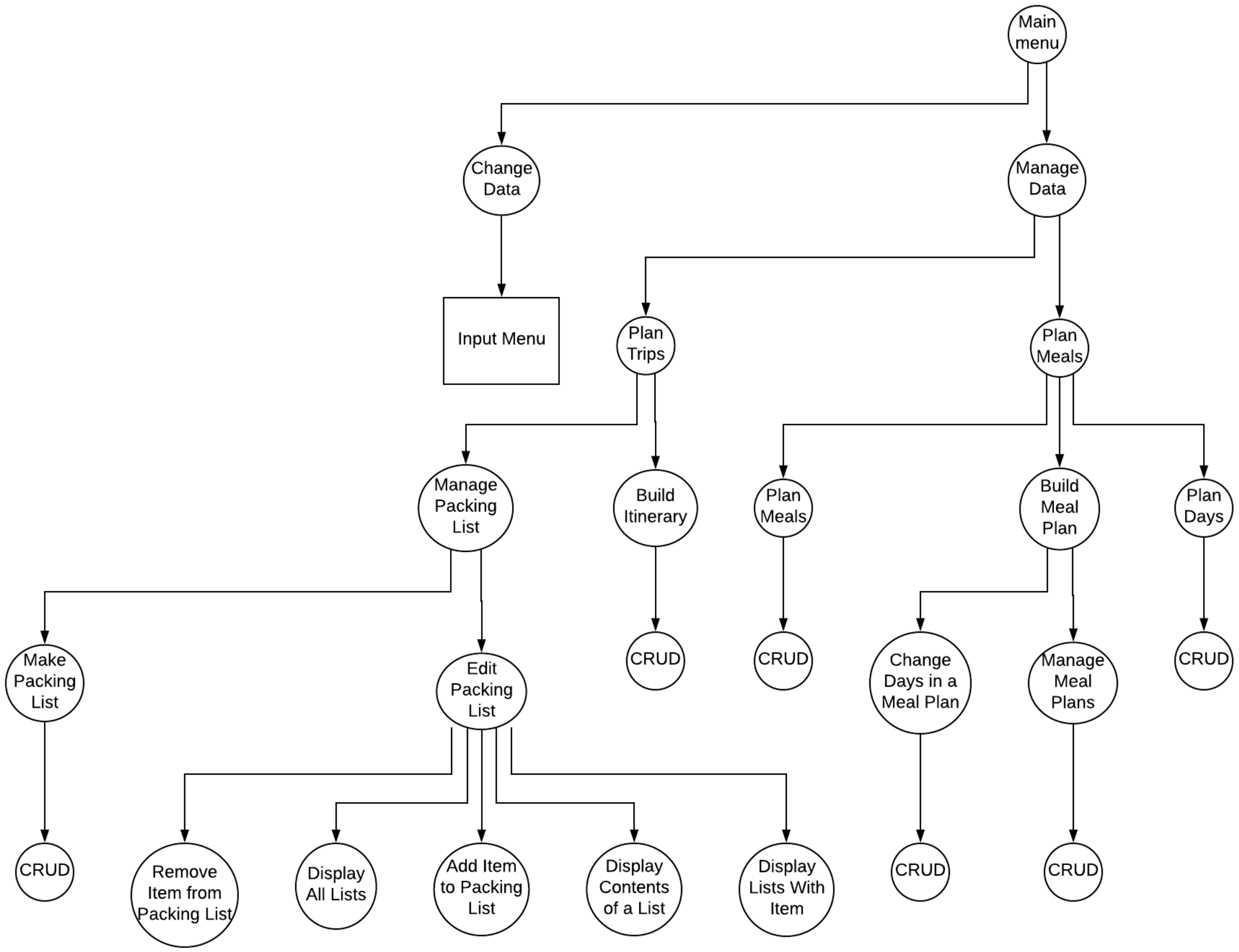
3.3.4 Operations

This application can perform three general classes of operation: Data storage, data assignment, and analytical queries. Data storage is the simplest of the three, and involves the insertion, selection, updating and deletion of the simple entity types like Food, Gear and Feature. Data assignment is the assembly of the simple entities into more complex ones, such Food into Meal Plan or Region and Feature into Location. Data assignment also uses INSERT, UPDATE, and DELETE, but usually after a SELECT operation to identify the key values of the rows involved. Analytical queries are purely SELECT operations that assemble the data contained in the database. For example, they can be used to identify the Food with the highest protein to weight ratio, or all of the Regions that do not ban alcohol stoves.
Each operation class, storage, assignment and analytical query is contained in its own sub-menu in the controller interface. The storage operations are contained in the Change Data Menu, the assignment operations in the Manage Data Menu, and the analytical queries in the Special Queries Menu. The graphs of each menu are

shown below. A node labeled CRUD represents a sub-menu with the Create, Retrieve, Update and Destroy operations for the relevant entity type.



Change Menu:



Mange Menu:

The Special Queries menu is not graphed, as it does not have a hierarchy. It is simply a list of options for each query.

3.4 Security

The security requirements for this application are minimal. No sensitive data of any kind is stored, as all information is sourced from publicly available documents provided by food and gear manufacturers or park administration. Hiker data is limited to a name and approximate weight.
Two user types were implemented with MySQL Workbench, the root user with privileges to modify the schema, and a hiker user who can only insert, update, select, and delete data. If the application is extended to a web interface, a login function will be required, and a system put in place to prevent users from modifying each other's lists, but for a personal tool this isn't necessary. It should be noted that the application as it currently stands allows the user to modify the information of multiple hikers. This is intentional, as it is common for backpackers to plan joint packing strategies and split gear between multiple individuals. The ability for a user to record information for hikers other than themselves and assign other hikers to different lists aids in the planning of more complex routes.
Additional security is provided by the controller interface. At no point does the controller program offer the user the opportunity to submit their own SQL queries. Instead, it uses a series of nested menus and prompts to allow

the user to select which pre-written queries they would like to submit. If the query requires variables from user input, the MySQL Connecter Python package is used to parametrize the inputs and protect against SQL injection. Additionally, the program contains routines to verify that the user input is of the appropriate data type corresponding to the column in the query.

3.5 Database Backup and Recovery

This application utilized the backup functionality provided by MySQL Workbench. MySQL Workbench can automatically generate a script containing the database schema, the information it contains, or both. These scripts were created whenever significant refactoring was in progress as a safeguard.
A web application would implement scheduled backups with a third-party application to handle the greater amount of data.

3.6 Using Database Design or CASE Tool

This database for this application was implemented using MySQL Workbench 8.0. The controller application was implemented in Python using Sublime Text 3.1.1.

3.7 Other Possible E/R Relationships

Gear Type initially began as a column in Gear and Gear Restrictions was a column in Region. Both were refactored into their own entities during normalization. Restriction Type could have been its own entity, and probably should in order to completely normalize the database to 3NF. However, as stated above, doing so would complicate the queries involving restrictions, considering that the table would contain at most four Restriction Types.
Meals were also originally assigned directly to Meal Plans, but it was quickly realized that repeating the same meal occurred often, especially considering that the purpose of the application is to help a user optimize their meal planning. The solution to the problem was the Days entity, but Days was poorly designed and introduced an even worse issue in that it prevented joins between Meals and Meal Plans. It was later removed, as the lack of duplication could be solved by inputting more data, while the inability to join was a much greater problem.
Itineraries could potentially use a container entity as well, but it is not strictly necessary. A Trip entity could contain Itineraries and arrange them based on their start and end points into a longer itinerary, each with it's associated Packing List and Meal Plan. This could be useful to model longer trips where changes in equipment might occur, or where the hiker needs to resupply at different points. At the moment, this can be approximated by managing the number of Itineraries associate with a particular hiker. If a Hiker is only associated with itineraries for a single trip, selecting those Itineraries associated with that Hiker performs the same function. This does require the deletion of old itineraries, so the Trip entity will definitely be implemented should the application be extended and need to handle larger amounts of information or input from multiple users.
Gear Restrictions and Region could have been implemented with composite keys, rather than an incrementing primary key. This would help to enforce the uniqueness of each entry in those tables but would make joining them more complex. Instead, a composite key was used in the joining table gear_restrictions_has_region. It is therefore possible to have duplicate entries while still maintaining the unique nature of their relationships.

4. Implementation Description

The implementation requirements for the database are to store data regarding Food, Gear, and Locations, and to allow easy manipulation of that data. The requirements for the controller interface, implemented in Python, were to store the SQL statements used to interact with the database and to organize and present them to the user with a menu system. The controller is also responsible for displaying the data recovered from the database in a readable table.

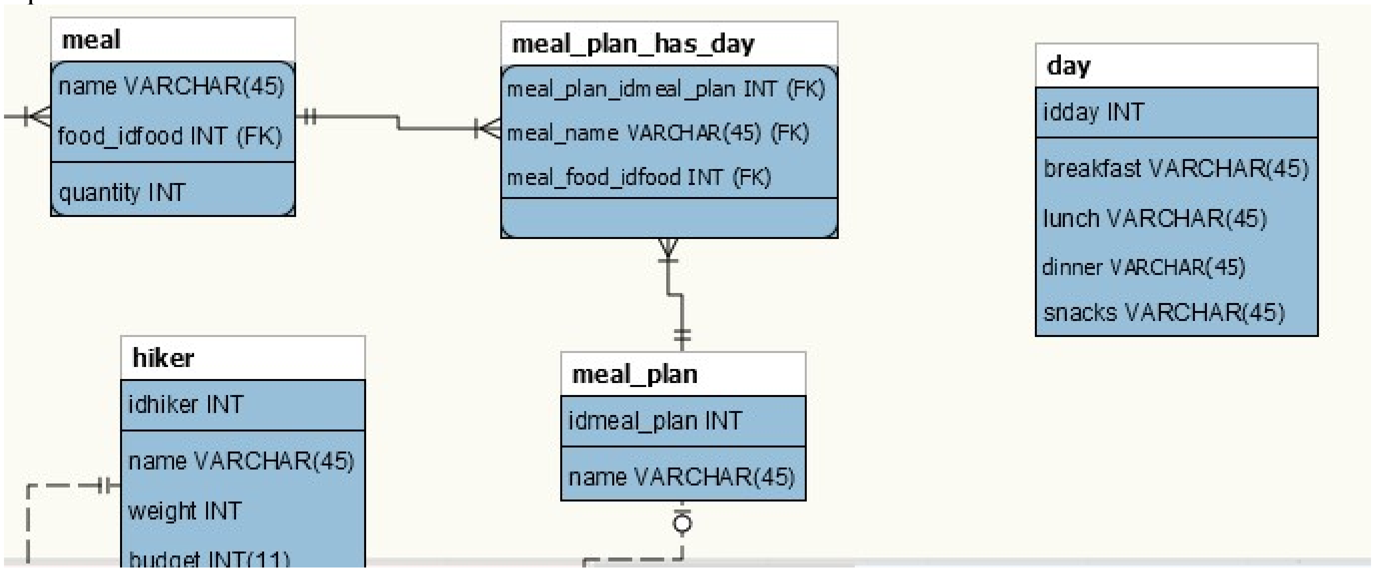
4.1 Design Changes

The Day was implemented to store groups of Meals and ultimately assign them to Meal Plans with fewer steps. This turned out to be a huge mistake, as it made the process of retrieving the food data associated with a meal plan effectively impossible. The entity was removed as an intermediate in the relationship between Meals and Meal Plans, and a more conventional N:M setup replaced it.

Updated Data Dictionary:

meal_plan_idmeal_plan	int(11)	NO	PRI	meal_plan_idmeal_plan
meal_name	varchar(45)	NO	PRI	meal_name
meal_food_idfood	int(11)	NO	PRI	meal_food_idfood
meal_plan_idmeal_plan	int(11)	NO	PRI	meal_plan_idmeal_plan

Updated ERD

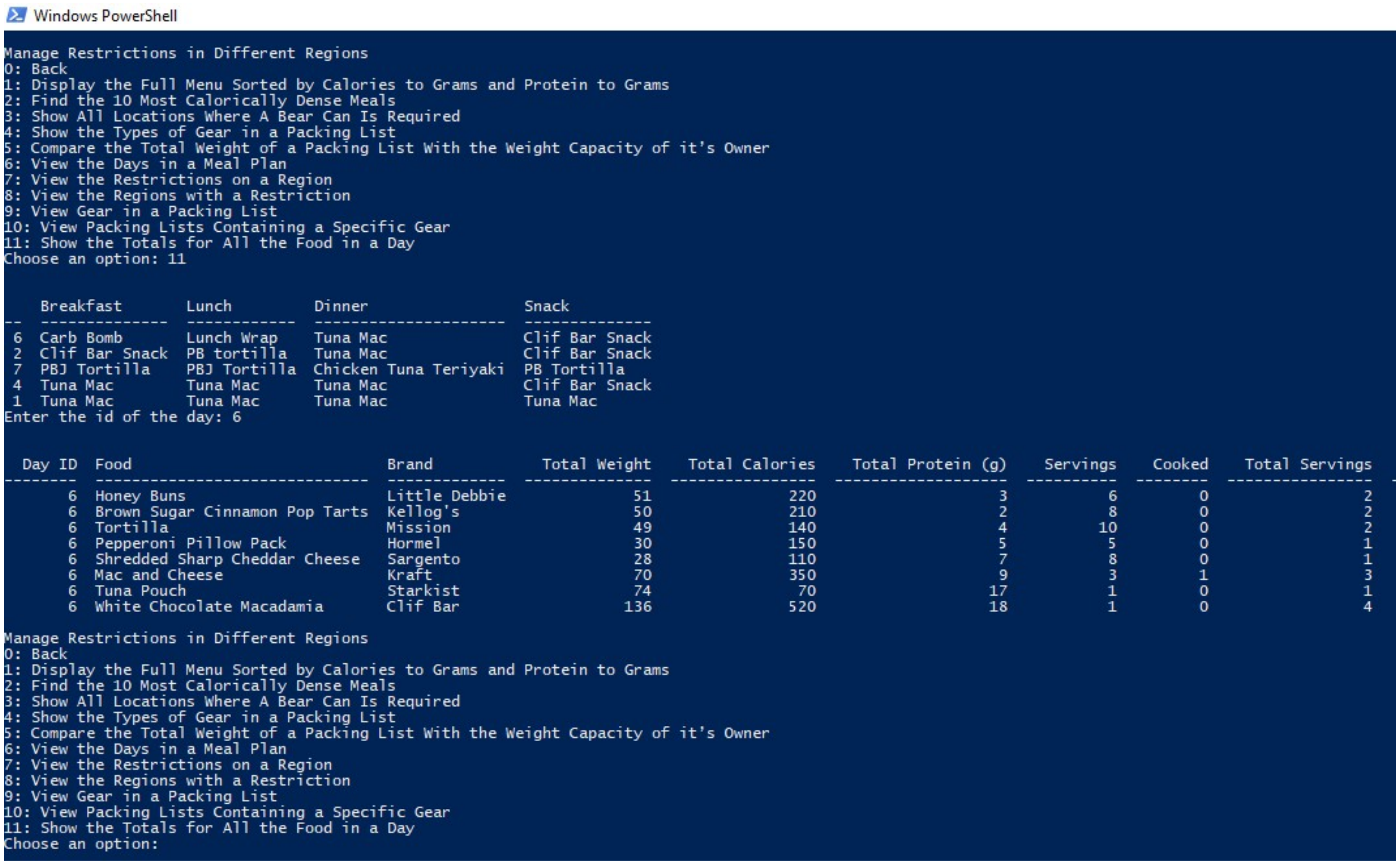


4.2 Advanced Features

There are no stored procedures, triggers or functions stored within the database. All SQL statements are stored in the controller interface as tuples of a single string. These tuples are used as arguments for the methods that interface directly with the database.

The only procedure within the database itself that could be considered a trigger is the cascade function taken by several tables when a foreign key is deleted. However, most tables do not have this behavior enabled in order to limit the amount of data that is lost with a single deletion.

A screenshot of the controller interface sending a query based on a choice by the user, as well as the tabulated results.



```
Manage Restrictions in Different Regions
0: Back
1: Display the Full Menu Sorted by Calories to Grams and Protein to Grams
2: Find the 10 Most Calorically Dense Meals
3: Show All Locations Where A Bear Can Is Required
4: Show the Types of Gear in a Packing List
5: Compare the Total Weight of a Packing List With the Weight Capacity of it's Owner
6: View the Days in a Meal Plan
7: View the Restrictions on a Region
8: View the Regions with a Restriction
9: View Gear in a Packing List
10: View Packing Lists Containing a Specific Gear
11: Show the Totals for All the Food in a Day
Choose an option: 11


    Breakfast       Lunch         Dinner                Snack
--  -------------   ------------  --------------------  --------------
 6  Carb Bomb       Lunch Wrap    Tuna Mac              Clif Bar Snack
 2  Clif Bar Snack  PB tortilla   Tuna Mac              Clif Bar Snack
 7  PBJ Tortilla    PBJ Tortilla  Chicken Tuna Teriyaki PB Tortilla
 4  Tuna Mac        Tuna Mac      Tuna Mac              Clif Bar Snack
 1  Tuna Mac        Tuna Mac      Tuna Mac              Tuna Mac
Enter the id of the day: 6


  Day ID  Food                          Brand           Total Weight   Total Calories   Total Protein (g)   Servings   Cooked   Total Servings
--------  ----------------------------  -------------   -------------  ---------------  ------------------  ---------  -------  ---------------
       6  Honey Buns                    Little Debbie             51              220                  3          6        0                2
       6  Brown Sugar Cinnamon Pop Tarts Kellog's                 50              210                  2          8        0                2
       6  Tortilla                      Mission                   49              140                  4         10        0                2
       6  Pepperoni Pillow Pack         Hormel                    30              150                  5          5        0                1
       6  Shredded Sharp Cheddar Cheese Sargento                  28              110                  7          8        0                1
       6  Mac and Cheese                Kraft                     70              350                  9          3        1                3
       6  Tuna Pouch                    Starkist                  74               70                 17          1        0                1
       6  White Chocolate Macadamia     Clif Bar                 136              520                 18          1        0                4

Manage Restrictions in Different Regions
0: Back
1: Display the Full Menu Sorted by Calories to Grams and Protein to Grams
2: Find the 10 Most Calorically Dense Meals
3: Show All Locations Where A Bear Can Is Required
4: Show the Types of Gear in a Packing List
5: Compare the Total Weight of a Packing List With the Weight Capacity of it's Owner
6: View the Days in a Meal Plan
7: View the Restrictions on a Region
8: View the Regions with a Restriction
9: View Gear in a Packing List
10: View Packing Lists Containing a Specific Gear
11: Show the Totals for All the Food in a Day
Choose an option:
```

4.3 Queries

The queries listed below are coded into the controller interface under the Special Queries menu option. Each is designed to show a potential use case for the Hike Planner application. There are many other queries coded into the controller interface, but these are simple data manipulation queries like INSERT or UPDATE and are not shown here.

4.3.1 Display the Full Menu Sorted by Calories to Grams and Protein to Grams

```
SELECT name, brand, weight, calories, protein, servings, cooked,
calories/weight AS "cal_per_g", protein/weight AS "prot_per_g"
FROM food ORDER BY cal_per_g DESC, prot_per_g DESC;
```

4.3.2 Find the 10 Most Calorically Dense Meals

```
SELECT meal.name, sum(food.calories * meal.quantity) AS cal_total
FROM meal JOIN food ON food_idfood = idfood
GROUP BY meal.name ORDER BY cal_total DESC LIMIT 10;
```

4.3.3 Show All Locations Where A Bear Can Is Required

```
SELECT location.name, region.name, gear_type_type, restriction FROM location
JOIN region ON location.region_idregion = region.idregion
JOIN gear_restrictions_has_region ON gear_restrictions_has_region.region_idregion= region.idregion
JOIN gear_restrictions
  ON gear_restrictions_has_region.gear_restrictions_idgear_restrictions =
  gear_restrictions.idgear_restrictions
WHERE gear_type_type = "Bear Can" AND restriction = "Required"
```

4.3.4 Show the Types of Gear in a Packing List

```
SELECT gear_type_type, gear.name, brand, quantity, quantity*weight AS total_weight
FROM packing_list
JOIN gear_has_packing_list
  ON packing_list.idpacking_list = gear_has_packing_list.packing_list_idpacking_list
JOIN gear ON gear.idgear = gear_has_packing_list.gear_idgear
WHERE packing_list.idpacking_list = 7
```

4.3.5 Compare the Total Weight of a Packing List With the Weight Capacity of it's Owner

```
SELECT packing_list.name, sum(quantity* gear.weight) AS total_weight, hiker.name,
ROUND((hiker.weight /4 * 1000),0) AS carry_capacity
FROM packing_list
JOIN gear_has_packing_list ON packing_list.idpacking_list =
gear_has_packing_list.packing_list_idpacking_list
JOIN gear ON gear.idgear = gear_has_packing_list.gear_idgear
JOIN hiker ON packing_list.hiker_idhiker = hiker.idhiker
WHERE packing_list.idpacking_list = 7
GROUP BY packing_list.name
```

4.3.6 View the Stats of a Meal Plan

```
SELECT meal_plan.name, SUM(food.weight), SUM(food.calories), SUM(food.protein)
FROM meal_plan
JOIN meal_plan_has_day ON meal_plan.idmeal_plan = meal_plan_idmeal_plan
JOIN food ON meal_food_idfood = food.idfood
WHERE idmeal_plan = INPUT
```

4.3.7 GROUP BY meal_plan.name**View the Restrictions on a Region**

```
SELECT name, gear_type_type, restriction FROM gear_restrictions_has_region
JOIN gear_restrictions ON gear_restrictions_idgear_restrictions = gear_restrictions.idgear_restrictions
JOIN region ON region_idregion = region.idregion
WHERE idregion = INPUT
```

4.3.8 View the Regions with a Restriction

```
SELECT name, gear_type_type, restriction FROM gear_restrictions_has_region
JOIN gear_restrictions ON gear_restrictions_idgear_restrictions = gear_restrictions.idgear_restrictions
JOIN region ON region_idregion = region.idregion
WHERE idgear_restrictions = INPUT
```

4.3.9 View Gear in a Packing List

```
SELECT gear.name, quantity FROM gear_has_packing_list
JOIN packing_list ON packing_list_idpacking_list = idpacking_list
JOIN gear ON gear_idgear = idgear
WHERE gear_has_packing_list.packing_list_idpacking_list = INPUT
```

4.3.10 View Packing Lists Containing a Specific Gear

```
SELECT packing_list.name, hiker.name, quantity FROM gear_has_packing_list
JOIN packing_list ON packing_list_idpacking_list = idpacking_list
JOIN gear ON gear_idgear = idgear
JOIN hiker ON packing_list.hiker_idhiker = idhiker
```

```
WHERE gear_has_packing_list.gear_idgear = INPUT
```

4.3.11 Show the Totals for All the Food in a Day

```
SELECT idday, name, brand, SUM(weight), SUM(calories), SUM(protein), servings, cooked,
SUM(quantity), CEIL(SUM(quantity)/servings)  FROM
(
          SELECT  idday, food.name, food.brand, food.weight, food.calories, food.protein, food.servings,
          food.cooked, meal.quantity
          FROM day
          JOIN meal ON day.breakfast = meal.name
          JOIN food ON meal.food_idfood = food.idfood
          where idday = INPUT

  UNION ALL

          SELECT  idday, food.name, food.brand, food.weight, food.calories, food.protein, food.servings,
          food.cooked, meal.quantity
          FROM day
          JOIN meal ON day.lunch = meal.name
          JOIN food ON meal.food_idfood = food.idfood
          where idday = INPUT

  UNION ALL

          SELECT  idday, food.name, food.brand, food.weight, food.calories, food.protein, food.servings,
          food.cooked, meal.quantity
          FROM day
          JOIN meal ON day.dinner = meal.name
          JOIN food ON meal.food_idfood = food.idfood
          where idday = INPUT

  UNION ALL

          SELECT  idday, food.name, food.brand, food.weight, food.calories, food.protein, food.servings,
          food.cooked, meal.quantity
          FROM day
          JOIN meal ON day.snacks = meal.name
          JOIN food ON meal.food_idfood = food.idfood
          where idday = INPUT
) z
GROUP BY name
```

5. CRUD Matrix

	E1	E2	E3	E4	E6	E7	E8	E9	E10	E11	E12	E13	E14
F1	CRUD												
F2		CRUD											
F3			CRUD										

F4				CRUD									
F5					CRUD								
F6						CRUD							
F7							CRUD						
F8								CRUD					
F9									CRUD				
F10										CRUD			
F11											CRUD		
F12												CRUD	
F13													CRUD
F14											U		
F15								R					
F16								R			R		
F17								R			R		
F18											U		
F19				U									
F20			R										
F21		R											
F22				U									
F23						U							
F24						U							
F25						R							
F26						R	R						
F27						R	R						

5.1 List of Entity Types

E1: Food
E2: Meal
E3: Day
E4: Meal Plan
E6: Hiker
E7: Packing List
E8: Gear
E9: Gear Restriction
E10: Gear Type
E11: Itinerary
E12: Region
E13: Location
E14: Feature

5.2 List of Functions

F1: Create/Read/Update/Destroy a Food
F2: Create/Read/Update/Destroy a Meal
F3: Create/Read/Update/Destroy a Day

F4: Create/Read/Update/Destroy a Meal Plan
F5: Create/Read/Update/Destroy a Hiker
F6: Create/Read/Update/Destroy a Packing List
F7: Create/Read/Update/Destroy a Gear
F8: Create/Read/Update/Destroy a Gear Restriction
F9: Create/Read/Update/Destroy a Gear Type
F10: Create/Read/Update/Destroy an Itinerary
F11: Create/Read/Update/Destroy a Region
F12: Create/Read/Update/Destroy a Location
F13: Create/Read/Update/Destroy a Feature
F14: Assign a Restriction to a Region
F15: Show All Restrictions
F16: Show All Restrictions in a Region
F17: Show All Regions with a Restriction
F18: Remove a Restriction from a Region
F19: Assign Days to a Plan*
F20: Display all Days*
F21: Show all Meals in a Plan
F22: Remove a Day from a Plan*
F23: Remove a Gear from a Packing List
F24: Display all Lists
F25: Display contents of a List
F26: Display all Lists containing a specific Gear
F27: Add a Gear to a Packing List

*Day methods have been replaced with ones that directly affect the relationship between Meals and Meal Plans. The functions remain the same.

6. Concluding Remarks

The Hike Planner application functions almost exactly as intended, with one exception. The Day entity was poorly implemented, and the desired reports involving it could not be generated. A list of everything to be tested was made at the start of the implementation process, but the joining Days to Meals was missed. By the time the error was identified, there was only time to partially resolve the issue. Day was removed from the relationship between Meal and Meal Plan, which allowed functional joins. However, it caused a new issue in that duplicate meals cannot be assigned to meal plans. This is a relatively minor issue that can be resolved by the user adding more data but is still not ideal. It is, however, better than not being able to join tables within the database. This leads to the first lesson learned: Test Absolutely Everything, then test it again. Test features before they are linked to other features. The poor implementation of Days could have been identified earlier with more systematic testing.

Another lesson is to stay focused on the objective. It can be easy to get lost in the minutiae of the data or get distracted with an idea for a new feature. However, it is important to constantly work toward the objective especially when the project involves a subject of great personal interest. Perfect optimization and cool features are less important than a working prototype.

Regarding my own personal strengths and weaknesses, I think my biggest weakness is that I am an inexperienced developer. My workflow and debugging strategies can be improved, as well as my familiarity with the tools and resources available to me. I think when I have more experience, easily avoidable mistakes like forgetting to test a table implementation will be easier to avoid. On the other hand, my biggest strength is defining the requirements of the application. I believe I successfully created a flexible design that can be extended in several ways with added functionality for the Food, Gear and Location data. All the reports I intended to generate were successfully implemented, and the Hike Planner application, while unpolished, has already helped me plan a short trip.

Appendices

Known Issues:

- Inserting a Meal Plan via the controller app does not work. However, inserting it through MySQL Workbench with the same SQL syntax works fine. A solution was not found.
- Queries can be improved by using a search feature based on a select subquery rather than asking for id values.
- Delete statements should ask the user if they are sure they want to proceed.
- Make meal menu should check to see if meal already exists.
- A basic GUI would make data manipulation much easier.
- Packing lists are not linked to itineraries. It would require another entity, packing_list_has_intinerary that would associate with hiker as well.
- Interfacing with Google Maps would improve the Location data
- More sophisticated packing list checks
- Duplicate Meals cannot be assigned to Meal Plans

Appendix A - DDL, INSERT, SELECT Statements

Appendix A contains the database dump for the application. It provides a seed script that can be imported into MySQL Workbench to generate a working version of the Hike Planner database along with some seed data. The controller interface can be found in the GitHub repository linked in references. Between that Python File, this script, and a copy of MySQL workbench, the complete application should be able to be reconstructed.

```
-- MySQL dump 10.13  Distrib 8.0.12, for Win64 (x86_64)
--
-- Host: 127.0.0.1    Database: mydb
-- ------------------------------------------------------
-- Server version	8.0.11

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
 SET NAMES utf8 ;
/*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;
/*!40103 SET TIME_ZONE='+00:00' */;
/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;

--
-- Table structure for table `day`
--

DROP TABLE IF EXISTS `day`;
```

```
/*!40101 SET @saved_cs_client     = @@character_set_client */;
 SET character_set_client = utf8mb4 ;
CREATE TABLE `day` (
  `idday` int(11) NOT NULL AUTO_INCREMENT,
  `breakfast` varchar(45) DEFAULT '0',
  `lunch` varchar(45) DEFAULT '0',
  `dinner` varchar(45) DEFAULT '0',
  `snacks` varchar(45) DEFAULT '0',
  PRIMARY KEY (`idday`),
  UNIQUE KEY `no_dups` (`breakfast`,`lunch`,`dinner`,`snacks`)
) ENGINE=InnoDB AUTO_INCREMENT=8 DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `day`
--

LOCK TABLES `day` WRITE;
/*!40000 ALTER TABLE `day` DISABLE KEYS */;
INSERT INTO `day` VALUES (6,'Carb Bomb','Lunch Wrap','Tuna Mac','Clif Bar Snack'),(2,'Clif Bar
Snack','PB tortilla','Tuna Mac','Clif Bar Snack'),(7,'PBJ Tortilla','PBJ Tortilla','Chicken Tuna Teriyaki','PB
Tortilla'),(4,'Tuna Mac','Tuna Mac','Tuna Mac','Clif Bar Snack'),(1,'Tuna Mac','Tuna Mac','Tuna Mac','Tuna
Mac');
/*!40000 ALTER TABLE `day` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `feature`
--

DROP TABLE IF EXISTS `feature`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
 SET character_set_client = utf8mb4 ;
CREATE TABLE `feature` (
  `idfeature` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(45) NOT NULL,
  `description` varchar(45) NOT NULL,
  PRIMARY KEY (`idfeature`),
  UNIQUE KEY `name_UNIQUE` (`name`)
) ENGINE=InnoDB AUTO_INCREMENT=8 DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `feature`
--

LOCK TABLES `feature` WRITE;
/*!40000 ALTER TABLE `feature` DISABLE KEYS */;
```

```
INSERT INTO `feature` VALUES (1,'Post Office','Holds resupply shipments'),(2,'Trailhead','Start or end tha
hike here'),(3,'Water source','Crucial source of drinking water'),(4,'Trash Can','Throw away trash
here'),(7,'Ranger Station','Check in with rangers to register or learn');
/*!40000 ALTER TABLE `feature` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `food`
--

DROP TABLE IF EXISTS `food`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
 SET character_set_client = utf8mb4 ;
CREATE TABLE `food` (
  `idfood` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(45) NOT NULL,
  `brand` varchar(45) DEFAULT NULL,
  `weight` int(11) NOT NULL,
  `calories` int(11) NOT NULL,
  `protein` int(11) NOT NULL,
  `servings` int(11) NOT NULL,
  `cooked` tinyint(1) NOT NULL DEFAULT '0',
  PRIMARY KEY (`idfood`)
) ENGINE=InnoDB AUTO_INCREMENT=50 DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `food`
--

LOCK TABLES `food` WRITE;
/*!40000 ALTER TABLE `food` DISABLE KEYS */;
INSERT INTO `food` VALUES (1,'Mac and Cheese','Kraft',70,350,9,3,1),(2,'Tuna
Pouch','Starkist',74,70,17,1,0),(3,'Tortilla','Mission',49,140,4,10,0),(4,'12oz. Peanut
Butter','Jif',33,190,7,10,0),(5,'White Chocolate Macadamia','Clif Bar',68,260,9,1,0),(31,'Maple & Brown Sugar
Instant Oats','Quaker',43,160,4,18,1),(32,'Honey Almond Granola','Bear Naked',30,140,6,10,0),(33,'Classic
Almond Butter','Justin\'s',32,190,7,14,0),(34,'Legend Beef Jerky, Teriyaki','Jack Link',28,70,9,10,0),(35,'Shells
and Real Aged Cheddar','Annie\'s',71,270,10,6,1),(36,'Honey Buns','Little Debbie',51,220,3,6,0),(37,'Brown
Sugar Cinnamon Pop Tarts','Kellog\'s',50,210,2,8,0),(38,'Strawberery Pop
Tarts','Kellog\'s',52,200,2,8,0),(39,'Unfrosted Strawberry Pop Tarts','Kellog\'s',50,210,2,8,0),(40,'Strawberry
Jelly, Squeeze Bottle','Smucker\'s',20,50,0,28,0),(41,'Pepperoni Pillow
Pack','Hormel',30,150,5,5,0),(42,'Shredded Sharp Cheddar Cheese','Sargento',28,110,7,8,0),(43,'Original
Mashed Potatoes','Idahoan',57,180,2,3,1),(44,'Beef Stroganoff With Noodles','Mountain
House',136,650,27,1,1),(45,'Chicken Teriyaki With Rice','Mountain House',115,440,20,1,1),(46,'Pad Thai
Veggie','Backpacker\'s Pantry',226,920,32,1,1),(47,'Fettuccini Alfredo With Chicken','Backpacker\'s
Pantry',212,540,32,1,1),(48,'Alfredo Broccoli Pasta Side','Knorr',126,480,16,1,1),(49,'Herb and Butter Rice
Side','Knorr',162,575,12,1,1);
/*!40000 ALTER TABLE `food` ENABLE KEYS */;
UNLOCK TABLES;
```

```
--
-- Table structure for table `gear`
--

DROP TABLE IF EXISTS `gear`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
 SET character_set_client = utf8mb4 ;
CREATE TABLE `gear` (
  `idgear` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(45) NOT NULL,
  `brand` varchar(45) DEFAULT '"Unknown"',
  `weight` varchar(45) NOT NULL,
  `gear_type_type` varchar(45) DEFAULT '"Luxury"',
  PRIMARY KEY (`idgear`),
  UNIQUE KEY `no_dups` (`name`,`brand`),
  KEY `fk_gear_gear_type1_idx` (`gear_type_type`),
  CONSTRAINT `fk_gear_gear_type` FOREIGN KEY (`gear_type_type`) REFERENCES `gear_type` (`type`)
) ENGINE=InnoDB AUTO_INCREMENT=25 DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `gear`
--

LOCK TABLES `gear` WRITE;
/*!40000 ALTER TABLE `gear` DISABLE KEYS */;
INSERT INTO `gear` VALUES (7,'Atmos 65
AG','Osprey','2041','Backpack'),(8,'Circuit','ULA','1162','Backpack'),(9,'BV500','BearVault','1162','Bear
Can'),(10,'Pocket Rocket 2','MSR','73','Gas Stove'),(12,'Kindle Paperwhite','Amazon','205','Luxury'),(13,'Sonic
Sleeping Bag','NEMO','932','Sleep System'),(14,'Magma 10 Sleeping Bag','REI Co-op','907','Sleep
System'),(15,'Copper Spur HV UL 2','Big Agnes','1360','Shelter'),(16,'Quarter Done 1','REI Co-
op','1303','Shelter'),(17,'Ultramid 2','Hyperlite Mountain Gear','534','Shelter'),(18,'RidgeREst SOLite','Therm-a-
Rest','538','Sleep System'),(19,'NeoAir XLite','Therm-a-Rest','453','Sleep System'),(20,'Squeeze Water
Filter','Sawyer','85','Water Purification'),(21,'Water Treatment - 1 oz','Aquamira','59','Water
Purification'),(22,'Storm Headlamp','Black Diamond','110','Light Source'),(23,'PreCip Rain
Jacket','Marmot','368','Water Purification'),(24,'TruArc 3','Brunton','31','Navigational Aid');
/*!40000 ALTER TABLE `gear` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `gear_has_packing_list`
--

DROP TABLE IF EXISTS `gear_has_packing_list`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
 SET character_set_client = utf8mb4 ;
CREATE TABLE `gear_has_packing_list` (
  `packing_list_idpacking_list` int(11) NOT NULL,
  `gear_idgear` int(11) NOT NULL DEFAULT '0',
  `quantity` varchar(45) NOT NULL DEFAULT '1',
  PRIMARY KEY (`packing_list_idpacking_list`,`gear_idgear`),
  KEY `fk_gear_has_packing_list_packing_list1_idx` (`packing_list_idpacking_list`),
  KEY `fk_gear_has_packing_list_gear1_idx` (`gear_idgear`),
  CONSTRAINT `fk_gear_has_packing_list_gear1` FOREIGN KEY (`gear_idgear`) REFERENCES `gear`
(`idgear`) ON DELETE CASCADE,
  CONSTRAINT `fk_gear_has_packing_list_packing_list1` FOREIGN KEY (`packing_list_idpacking_list`)
REFERENCES `packing_list` (`idpacking_list`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `gear_has_packing_list`
--

LOCK TABLES `gear_has_packing_list` WRITE;
/*!40000 ALTER TABLE `gear_has_packing_list` DISABLE KEYS */;
INSERT INTO `gear_has_packing_list` VALUES
(1,7,'1'),(1,14,'1'),(1,16,'1'),(1,18,'1'),(1,24,'1'),(4,7,'1'),(4,8,'1'),(7,8,'1'),(7,9,'1'),(7,10,'1'),(7,13,'1'),(7,15,'1'),(7,20
,'1');
/*!40000 ALTER TABLE `gear_has_packing_list` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `gear_restrictions`
--

DROP TABLE IF EXISTS `gear_restrictions`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
 SET character_set_client = utf8mb4 ;
CREATE TABLE `gear_restrictions` (
  `idgear_restrictions` int(11) NOT NULL AUTO_INCREMENT,
  `gear_type_type` varchar(45) NOT NULL,
  `restriction` enum('Required','Prohibited','Reccomended','Discouraged') NOT NULL,
  PRIMARY KEY (`idgear_restrictions`),
  UNIQUE KEY `no_dups` (`gear_type_type`,`restriction`),
  KEY `fk_gear_restrictions_gear_type1_idx` (`gear_type_type`),
  CONSTRAINT `fk_gear_restrictions_gear_type1` FOREIGN KEY (`gear_type_type`) REFERENCES
`gear_type` (`type`)
) ENGINE=InnoDB AUTO_INCREMENT=7 DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `gear_restrictions`
--

LOCK TABLES `gear_restrictions` WRITE;
/*!40000 ALTER TABLE `gear_restrictions` DISABLE KEYS */;
INSERT INTO `gear_restrictions` VALUES (2,'Bear Can','Required'),(6,'Bear
Can','Reccomended'),(3,'Drone','Prohibited'),(1,'Water Container','Reccomended');
/*!40000 ALTER TABLE `gear_restrictions` ENABLE KEYS */;
```

```
UNLOCK TABLES;

--
-- Table structure for table `gear_restrictions_has_region`
--

DROP TABLE IF EXISTS `gear_restrictions_has_region`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
 SET character_set_client = utf8mb4 ;
CREATE TABLE `gear_restrictions_has_region` (
  `gear_restrictions_idgear_restrictions` int(11) NOT NULL,
  `region_idregion` int(11) NOT NULL,
  PRIMARY KEY (`gear_restrictions_idgear_restrictions`,`region_idregion`),
  KEY `fk_gear_restrictions_has_region_region1_idx` (`region_idregion`),
  KEY `fk_gear_restrictions_has_region_gear_restrictions1_idx` (`gear_restrictions_idgear_restrictions`),
  CONSTRAINT `fk_gear_restrictions_has_region_gear_restrictions1` FOREIGN KEY
(`gear_restrictions_idgear_restrictions`) REFERENCES `gear_restrictions` (`idgear_restrictions`) ON DELETE
CASCADE,
  CONSTRAINT `fk_gear_restrictions_has_region_region1` FOREIGN KEY (`region_idregion`)
REFERENCES `region` (`idregion`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `gear_restrictions_has_region`
--

LOCK TABLES `gear_restrictions_has_region` WRITE;
/*!40000 ALTER TABLE `gear_restrictions_has_region` DISABLE KEYS */;
INSERT INTO `gear_restrictions_has_region` VALUES (2,1),(3,1),(2,2),(3,2),(3,4);
/*!40000 ALTER TABLE `gear_restrictions_has_region` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `gear_type`
--

DROP TABLE IF EXISTS `gear_type`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
 SET character_set_client = utf8mb4 ;
CREATE TABLE `gear_type` (
  `type` varchar(45) NOT NULL,
  `notes` varchar(45) NOT NULL,
  PRIMARY KEY (`type`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `gear_type`
--
```

```
LOCK TABLES `gear_type` WRITE;
/*!40000 ALTER TABLE `gear_type` DISABLE KEYS */;
INSERT INTO `gear_type` VALUES ('Alcohol Stove','Lightweight stove that uses denatured
alcohol'),('Backpack','Carries a gear for you'),('Bear Can','Prevents wildlife from accessing food'),('Cannister
Fuel','works with gas stoves'),('Clothing','stay warm and protected'),('Drone','personal flying machine for
photography'),('Firearm','weapon for hunting or defense'),('First Aid','emergency supply'),('Gas Stove','Boils
water from a fuel canister'),('Glass Container','contains water, or other things'),('Light Source','to see at
night'),('Luxury','Nice to have, but does nothing special'),('Navigational Aid','for finding a
route'),('Raincoat','stay dry'),('Shelter','protects from elements'),('Sleep System','stay warm when
sleeping'),('Water Container','Holds a volume of water'),('Water Purification','safely treats water'),('Water
Weight','drinking water, just for weight calculations');
/*!40000 ALTER TABLE `gear_type` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `hiker`
--

DROP TABLE IF EXISTS `hiker`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
 SET character_set_client = utf8mb4 ;
CREATE TABLE `hiker` (
  `idhiker` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(45) NOT NULL,
  `weight` int(11) NOT NULL,
  `budget` int(11) DEFAULT '0',
  `age` int(11) DEFAULT '0',
  PRIMARY KEY (`idhiker`),
  UNIQUE KEY `idhiker_UNIQUE` (`idhiker`),
  UNIQUE KEY `no_dups` (`name`,`weight`)
) ENGINE=InnoDB AUTO_INCREMENT=18 DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `hiker`
--

LOCK TABLES `hiker` WRITE;
/*!40000 ALTER TABLE `hiker` DISABLE KEYS */;
INSERT INTO `hiker` VALUES (1,'Science',86,NULL,NULL),(3,'Athena',72,NULL,NULL),(5,'Loner
Boner',41,NULL,NULL),(6,'Fraid Not',80,NULL,NULL),(7,'Pony Puncher',75,NULL,NULL),(11,'Bill
Bob',20,NULL,NULL);
/*!40000 ALTER TABLE `hiker` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `itinerary`
--
```

```
DROP TABLE IF EXISTS `itinerary`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
 SET character_set_client = utf8mb4 ;
CREATE TABLE `itinerary` (
  `iditinerary` int(11) NOT NULL AUTO_INCREMENT,
  `hiker_idhiker` int(11) DEFAULT '0',
  `start_idlocation` int(11) DEFAULT NULL,
  `end_idlocation` int(11) DEFAULT '0',
  `no_days` int(11) NOT NULL,
  `meal_plan_idmeal_plan` int(11) DEFAULT NULL,
  PRIMARY KEY (`iditinerary`),
  KEY `fk_itinerary_hiker1_idx` (`hiker_idhiker`),
  KEY `fk_itinerary_location2_idx` (`end_idlocation`),
  KEY `fk_itinerary_meal_plan1_idx` (`meal_plan_idmeal_plan`),
  KEY `fk_itinerary_location1` (`start_idlocation`),
  CONSTRAINT `fk_itinerary_hiker1` FOREIGN KEY (`hiker_idhiker`) REFERENCES `hiker` (`idhiker`),
  CONSTRAINT `fk_itinerary_location1` FOREIGN KEY (`start_idlocation`) REFERENCES `location`
(`idlocation`),
  CONSTRAINT `fk_itinerary_location2` FOREIGN KEY (`end_idlocation`) REFERENCES `location`
(`idlocation`),
  CONSTRAINT `fk_itinerary_meal_plan1` FOREIGN KEY (`meal_plan_idmeal_plan`) REFERENCES
`meal_plan` (`idmeal_plan`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `itinerary`
--

LOCK TABLES `itinerary` WRITE;
/*!40000 ALTER TABLE `itinerary` DISABLE KEYS */;
INSERT INTO `itinerary` VALUES (1,1,1,2,3,7),(2,3,6,7,4,8);
/*!40000 ALTER TABLE `itinerary` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `location`
--

DROP TABLE IF EXISTS `location`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
 SET character_set_client = utf8mb4 ;
CREATE TABLE `location` (
  `idlocation` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(45) NOT NULL,
  `feature_idfeature` int(11) DEFAULT '0',
  `region_idregion` int(11) DEFAULT NULL,
  PRIMARY KEY (`idlocation`),
  KEY `fk_location_feature1_idx` (`feature_idfeature`),
  KEY `fk_location_region1_idx` (`region_idregion`),
  CONSTRAINT `fk_location_feature1` FOREIGN KEY (`feature_idfeature`) REFERENCES `feature`
(`idfeature`) ON DELETE SET NULL,
  CONSTRAINT `fk_location_region1` FOREIGN KEY (`region_idregion`) REFERENCES `region`
(`idregion`) ON DELETE SET NULL
) ENGINE=InnoDB AUTO_INCREMENT=8 DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `location`
--

LOCK TABLES `location` WRITE;
/*!40000 ALTER TABLE `location` DISABLE KEYS */;
INSERT INTO `location` VALUES (1,'Graves Creek Trailhead',2,1),(2,'Damascus PO',1,7),(5,'Enchanted
Valley Station',7,1),(6,'Point A',2,2),(7,'Point B',2,2);
/*!40000 ALTER TABLE `location` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `meal`
--

DROP TABLE IF EXISTS `meal`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
 SET character_set_client = utf8mb4 ;
CREATE TABLE `meal` (
  `name` varchar(45) NOT NULL,
  `food_idfood` int(11) NOT NULL,
  `quantity` int(11) NOT NULL,
  PRIMARY KEY (`name`,`food_idfood`),
  UNIQUE KEY `no_dups` (`name`,`food_idfood`),
  KEY `fk_meal_food_idx` (`food_idfood`),
  KEY `fk_meal_name` (`name`),
  CONSTRAINT `fk_meal_food` FOREIGN KEY (`food_idfood`) REFERENCES `food` (`idfood`) ON
DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `meal`
--

LOCK TABLES `meal` WRITE;
/*!40000 ALTER TABLE `meal` DISABLE KEYS */;
INSERT INTO `meal` VALUES ('Carb Bomb',36,2),('Carb Bomb',37,2),('Chicken Tuna
Teriyaki',2,1),('Chicken Tuna Teriyaki',45,1),('Clif Bar Snack',5,3),('Dense Insulation',35,6),('Dense
Insulation',43,3),('Lunch Wrap',3,2),('Lunch Wrap',41,1),('Lunch Wrap',42,1),('PB Tortilla',3,1),('PB
Tortilla',4,3),('PBJ Tortilla',3,1),('PBJ Tortilla',4,2),('PBJ Tortilla',40,2),('Test meal',44,1),('Tuna
Mac',1,3),('Tuna Mac',2,1),('Tuna Mac',5,1);
/*!40000 ALTER TABLE `meal` ENABLE KEYS */;
```

```
UNLOCK TABLES;

--
-- Table structure for table `meal_plan`
--

DROP TABLE IF EXISTS `meal_plan`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
 SET character_set_client = utf8mb4 ;
CREATE TABLE `meal_plan` (
  `idmeal_plan` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(45) NOT NULL,
  PRIMARY KEY (`idmeal_plan`),
  UNIQUE KEY `name_UNIQUE` (`name`)
) ENGINE=InnoDB AUTO_INCREMENT=10 DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `meal_plan`
--

LOCK TABLES `meal_plan` WRITE;
/*!40000 ALTER TABLE `meal_plan` DISABLE KEYS */;
INSERT INTO `meal_plan` VALUES (8,'4 Day Trip'),(9,'A to B'),(5,'Day hike'),(6,'Long Hike'),(3,'No
Cook'),(7,'Short Hike');
/*!40000 ALTER TABLE `meal_plan` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `meal_plan_has_day`
--

DROP TABLE IF EXISTS `meal_plan_has_day`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
 SET character_set_client = utf8mb4 ;
CREATE TABLE `meal_plan_has_day` (
  `meal_plan_idmeal_plan` int(11) NOT NULL,
  `meal_name` varchar(45) NOT NULL,
  `meal_food_idfood` int(11) NOT NULL,
  PRIMARY KEY (`meal_name`,`meal_food_idfood`,`meal_plan_idmeal_plan`),
  KEY `fk_meal_plan_has_day_meal_plan1_idx` (`meal_plan_idmeal_plan`),
  CONSTRAINT `fk_meal_plan_has_day_meal1` FOREIGN KEY (`meal_name`, `meal_food_idfood`)
REFERENCES `meal` (`name`, `food_idfood`),
  CONSTRAINT `fk_meal_plan_has_day_meal_plan1` FOREIGN KEY (`meal_plan_idmeal_plan`)
REFERENCES `meal_plan` (`idmeal_plan`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `meal_plan_has_day`
```

```
--

LOCK TABLES `meal_plan_has_day` WRITE;
/*!40000 ALTER TABLE `meal_plan_has_day` DISABLE KEYS */;
INSERT INTO `meal_plan_has_day` VALUES (5,'Carb Bomb',36),(5,'Carb Bomb',37),(5,'Tuna
Mac',1),(5,'Tuna Mac',2),(5,'Tuna Mac',5),(6,'Lunch Wrap',3),(6,'Lunch Wrap',42),(6,'Tuna Mac',2),(9,'Carb
Bomb',36),(9,'Carb Bomb',37),(9,'Dense Insulation',35),(9,'Dense Insulation',43);
/*!40000 ALTER TABLE `meal_plan_has_day` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `packing_list`
--

DROP TABLE IF EXISTS `packing_list`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
 SET character_set_client = utf8mb4 ;
CREATE TABLE `packing_list` (
  `idpacking_list` int(11) NOT NULL AUTO_INCREMENT,
  `hiker_idhiker` int(11) DEFAULT NULL,
  `name` varchar(45) NOT NULL,
  PRIMARY KEY (`idpacking_list`),
  KEY `fk_packing_list_hiker1_idx` (`hiker_idhiker`),
  CONSTRAINT `fk_packing_list_hiker1` FOREIGN KEY (`hiker_idhiker`) REFERENCES `hiker` (`idhiker`)
) ENGINE=InnoDB AUTO_INCREMENT=8 DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `packing_list`
--

LOCK TABLES `packing_list` WRITE;
/*!40000 ALTER TABLE `packing_list` DISABLE KEYS */;
INSERT INTO `packing_list` VALUES (1,1,'3 Day Overnight'),(4,3,'Day Hike'),(6,5,'Day Hike'),(7,3,'A to B');
/*!40000 ALTER TABLE `packing_list` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `region`
--

DROP TABLE IF EXISTS `region`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
 SET character_set_client = utf8mb4 ;
CREATE TABLE `region` (
  `idregion` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(45) NOT NULL,
  `owner` varchar(45) NOT NULL,
  PRIMARY KEY (`idregion`),
  UNIQUE KEY `name_UNIQUE` (`name`)
) ENGINE=InnoDB AUTO_INCREMENT=10 DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `region`
--

LOCK TABLES `region` WRITE;
/*!40000 ALTER TABLE `region` DISABLE KEYS */;
INSERT INTO `region` VALUES (1,'Olympic National Park','National Park Service'),(2,'Lassen Volcanic
National Park','National Park Service'),(4,'Shenandoah National Park','National Park Service'),(7,'Damascus,
VA','Private'),(8,'Jasper National Park','Parks Canada'),(9,'Annapurna Conservation Area','The National Trust
for Nature Conservation');
/*!40000 ALTER TABLE `region` ENABLE KEYS */;
UNLOCK TABLES;
/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;

-- Dump completed on 2018-08-10 11:49:40
```

Appendix B - Data Dictionary Index

Field	Table
age	hiker
brand	food
brand	gear
breakfast	day
budget	hiker
calories	food
cooked	food
day_idday	meal_plan_has_day
description	feature
dinner	day
end_idlocation	itinerary
feature_idfeature	location
food_idfood	meal
gear_idgear	gear_has_packing_list
gear_restrictions_idgear_restrictions	gear_restrictions_has_region
gear_type_type	gear
gear_type_type	gear_restrictions
hiker_idhiker	itinerary
hiker_idhiker	packing_list

idday	day
idfeature	feature
idfood	food
idgear	gear
idgear_restrictions	gear_restrictions
idhiker	hiker
iditinerary	itinerary
idlocation	location
idmeal	meal
idmeal_plan	meal_plan
idmeal_plan_has_day	meal_plan_has_day
idpacking_list	packing_list
idregion	int(11)
lunch	day
meal_plan_idmeal_plan	itinerary
meal_plan_idmeal_plan	meal_plan_has_day
name	feature
name	food
name	gear
name	hiker
name	location
name	meal
name	meal_plan
name	packing_list
name	varchar(45)
no_days	itinerary
notes	gear_type
owner	varchar(45)
packing_list_idpacking_list	gear_has_packing_list
protein	food
quantity	gear_has_packing_list
quantity	meal
region_idregion	gear_restrictions_has_region
region_idregion	location
restriction	
	gear_restrictions
servings	food
snacks	day
start_idlocation	itinerary
type	gear_type
weight	food
weight	gear
weight	hiker

References

Controller application can be downloaded from:
https://github.com/asgray/HikePlanner

Food and gear data retrieved from the respective manufacturer’s website.