Small-Scale Aquaponic Raw Fruit and Vegetable Production and Distribution Optimization Through Database Design

Hayden Elza – May, 15 2015
University of Wisconsin – Madison

Aquaponics, a combination of aquaculture and hydroponics, has been growing in popularity over the last few decades due to its unique combination of environmental and economic benefits. Aquaculture is the practice of raising fish in closed recirculating systems. These systems are often plagued by high levels of waste nitrogen produced by fish in the form of ammonia. Traditionally this requires the used of expensive biofilters, but what is a problem of one system may serve as a solution for another system. Hydroponics, the production of plant crops with liquid nutrients in the absence of soil, requires a constant supply of liquid nutrients in relatively large amounts. By combining the two systems, the dissolved waste nutrients from the fish are recovered by the plants, thereby solving the problem of each system. Aquaponics also has the added benefit making the combined systems more stable, requiring less monitoring and a wider safety margin for good water quality.[1]

The advantages of aquaponics extend beyond the production of food and into its consumption. Due to the nature by which the plants are grown, the resulting crop is considered organic and free of herbicide and pesticide residues.[2] This is because even pesticides that are registered will still pose a threat to fish, and therefore are not permitted for using in aquaponic systems.[1] More generally, aquaponics on a small scale can provide local economic benefits such as providing cheaper and more fresh produce due to low overhead expenses, no middleman to handle distribution and sales, and less time spent in transport from producer to consumer.

Many people see the benefits in such a system and are trying their hand a starting such an operation. One such person is a family member of mine. The goal is to first provide enough food to sustain his household, while selling surplus food to friends and family. Given the advantages of aquaponics for the consumer, interest is expected to grow allowing for expansion of operation. Customers will be acquired by word of mouth and directed to a website where they will be able to make an account and order fresh fruits and vegetables from the available stock to be delivered to their home or picked up on location. In order to support the functions of the website and delivery service, a database will need to be created to track inventory, orders, and customer information, as well as a way to calculate the optimal delivery route given a particular set of customers on any given day to minimize travel costs, time, or any other variable of interest.

METHODS

Database Requirements

The resulting database will work to support the website and ordering functions. This includes inventory (e.g. food item, date planted, expected maturity date, harvest by date, horticultural application, and notes), customer information (e.g. name, phone number, address, etc.), and specific orders. These data can be used to calculate daily routes to minimize time or fuel use to minimize costs associated with delivery, track crop maturity, and analyze consumer trends (e.g. what time of year certain foods are bought, what time of day most people place orders, where most customers are located to target future customer expansion, what are the most popular items). Realistically many applications of the data will likely be discovered after its implementation, but creating a simple yet robust database design will go a long way to ensure all future needs of the operation are met.

Datasets

Due to the timing of the design of the database before any fruits or vegetables are actually produced, there are no data concerning the planting or sales of the produce. The actual datasets will be created when the database is finally implemented. In the interim, a mock dataset was created (i.e. fake inventory, customers, and orders) in order to prove the concept. The road dataset originates from the U.S. Census TIGER dataset from 2010.[3]

Model Design

Figure 1 on the next page shows the conceptual model of the database. Entities, i.e. the planting record, inventory, orders, customers, and roads, are represented as rectangular boxes. These entities have attributes that are facts about, or properties of the entity. For example, a customer is described by a unique customer identification number, first and last name, phone number, email, and password for account. Attributes which are underlined represent primary keys of the entity that can uniquely identify a specific entity. In the case of a customer, the customer identification number is used. In the case of the planting record, a composite key comprised of bed number, date planted, and food item is used to uniquely identify the specific entity. Weak entities are represented with double bordered rectangles. Weak entities, such as order details, rely on another entity for their existence. Without an order, the order details cannot exist. Relationships between entities are represented as diamonds. A single line connecting an entity to a relationship indicates optional participation, whereas a double line indicates mandatory participation. One such case of mandatory participation is that a customer must have an address. This is under the assumption that customers will, baring special circumstances, have the food items delivered to their residence, therefore an address is necessary. Maximum carnality is represented as numbers or letters along the relationship lines. Relationships can be one to one, one to many, or many to many. An example of this is that *one* customer may have *many* orders, but *one* customer will only have *one* address.

The resulting logical schema diagram can be seen on page four in Figure 2. This time entities are represented as bold titles and attributes as gray boxes. Primary keys are underlined and tail ends of arrows represent foreign keys. Arrows with dotted lines are not part of the standard logical schema format, but were added to represent attributes which are derived from another.

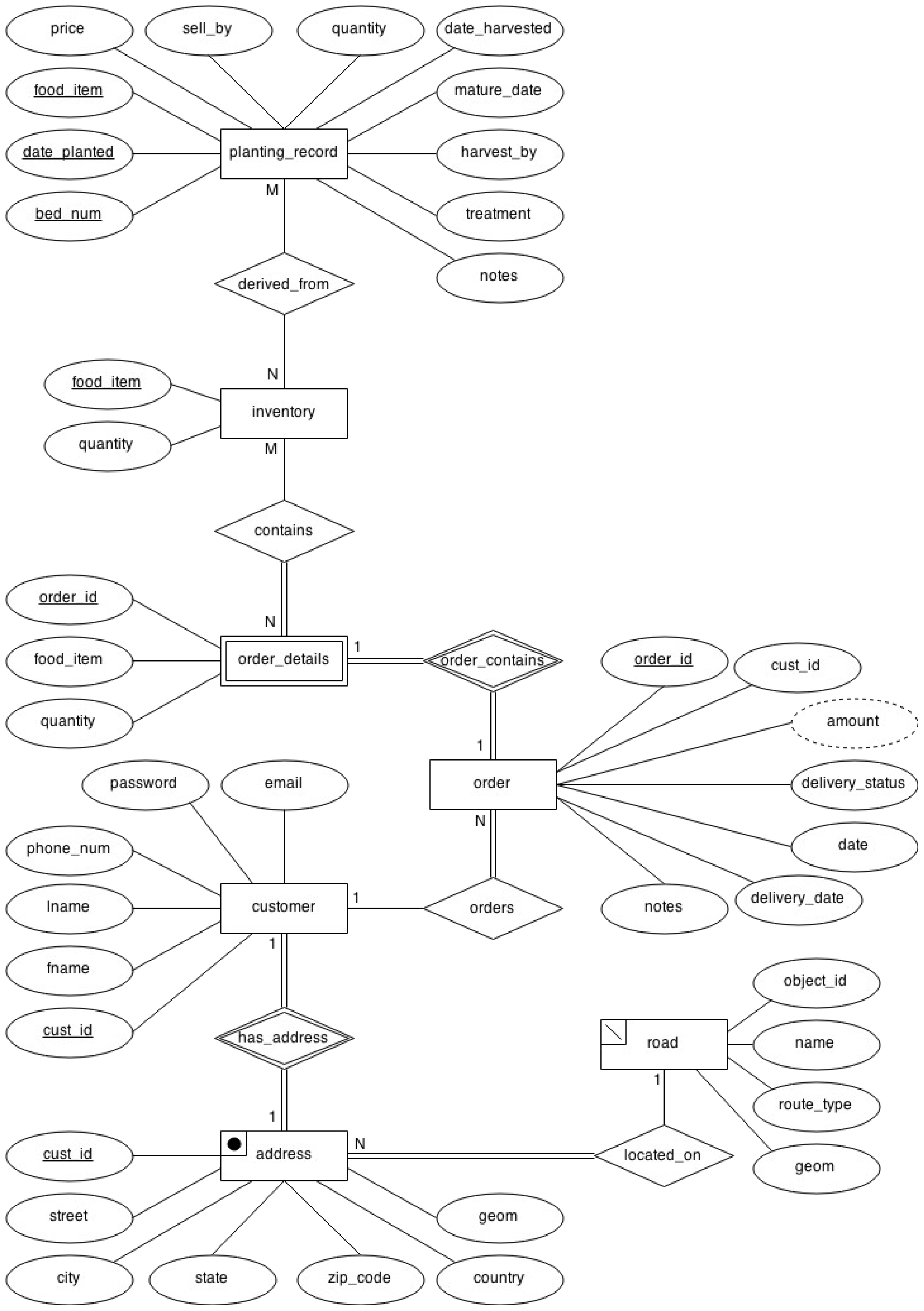

Figure 1. *An ER diagram defining both spatial and aspatial objects and their respective attributes.*

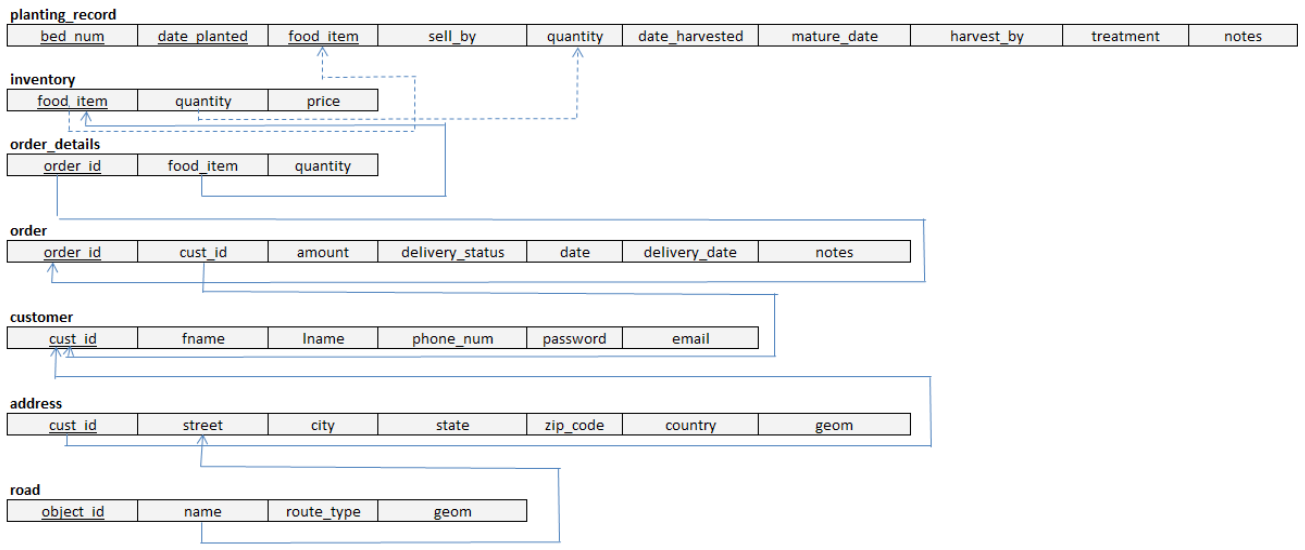
planting_record
bed_num | date_planted | food_item | sell_by | quantity | date_harvested | mature_date | harvest_by | treatment | notes
inventory
food_item | quantity | price
order_details
order_id | food_item | quantity
order
order_id | cust_id | amount | delivery_status | date | delivery_date | notes
customer
cust_id | fname | lname | phone_num | password | email
address
cust_id | street | city | state | zip_code | country | geom
road
object_id | name | route_type | geom

Figure 2. *Logical schema diagram of the database.*

Database Implementation

The database was created by first building the framework in pgModeler. A screenshot demonstrating this step can be seen below in figure 3. Each box represents an entity and within each box are the attributes which describe the entity. Boxes are connected by relationships. Constraints such as primary key, foreign key, and not null are explicitly defined during this stage. Attribute types such as char, date, and decimal are also defined during this stage. Some attributes or relationships have not been defined at this time as they were added later as the design developed and matured.

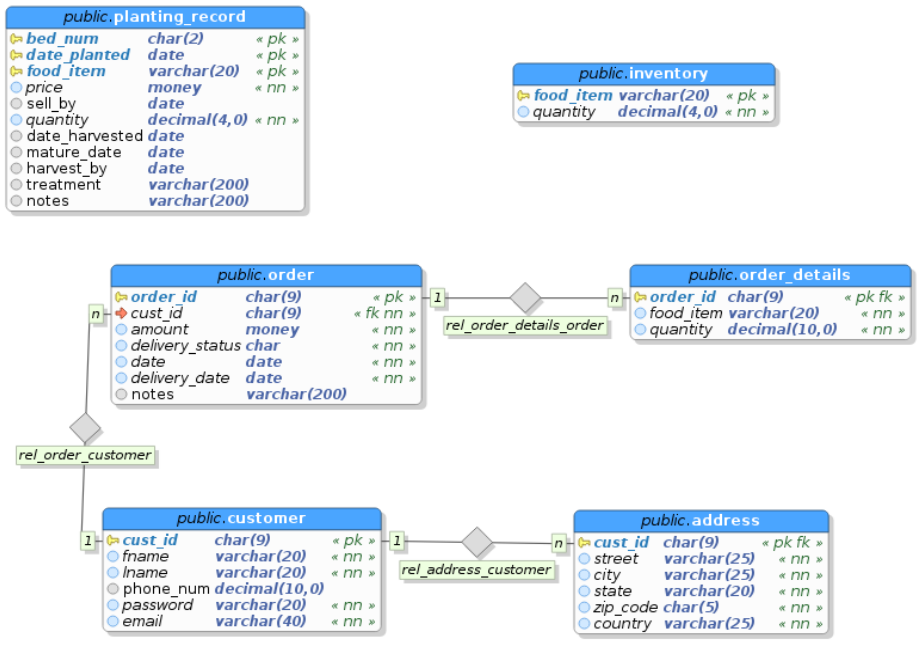

Figure 3. *Screen shot of pgModeler showing entities and their attributes and relationships.*

The database model was exported from pgModeler as an SQL file and imported into a PostgreSQL database using pgAdmin III. Using pgAdmin, more attributes and relationships where added. More importantly, at this stage data were added. Mock customer, order, and inventory data were added using the INSERT INTO command. The address geometry column and road table were added by importing data from shapefiles using the database manager in QGIS. Support for geographical data in the postgreSQL server is provided by PostGIS.

Database Manipulations

Following completion of the database implementation, a set of database manipulation commands were created to support the functions required by both the consumer and the producer. It is impractical for either the consumer or the producer to interact directly with the database using a tool such as pgAdmin. Instead, a python script was created to support actions such as customer registration, inventory updates, order placement, order fulfillment, and delivery route planning.[4] This also ensures that all commands used to perform an action are completed without error using transactions. If one command fails, early commands in the action are rolled back. For instance, if a customer tries to order an item that is not in stock, the order is first created, then the item subtracted from the inventory. The second command will fail due to insufficient supply, therefore the first command creating the order will be rolled back. As a business, maintaining the integrity of the database is of utmost importance, and using a python module with transaction functional should prove to be extremely useful.

RESULTS

Using the python script,[4] a list of orders to fulfill on a given day can be printed to a CSV file by calling the print_orders() function. In order to calculate the optimal route on any given day given a set of addresses, the function delivery_sequence() is used. The delivery_sequence() function utilizes a euclidean solution to the traveling salesman problem via the pgRouting extension. Using roads to calculate optimal routes can be advantageous in some cases, but in the case of this application where the goal is to minimize distance traveled, the euclidean method is a reasonable predictor of real-world instances.[5] Example outputs of both print_orders() and delivery_sequence() can be found in the GitHub repository.[4] The output of the delivery_sequence() function is a CSV file with an ordered list of customers and addresses. Other functions such as placing an order can be performed by calling a function with arguments. An example of placing an order is as follows: `place_order('654919981', '408827164', '35.00', '0', '03/15/2015', '03/16/2015', 'NULL', [['apple','10'], ['pear','5']])` where the arguments are order id, customer id, amount, delivery status, date ordered, delivery_date, notes, and items ordered respectively. There are many other functions in the script and the script is expected to grow as the need of certain functionality arises.

REFERENCES

1. Rakocy JE, Masser MP, Losordo TM. Recirculating Aquaculture Tank Production Systems: Aquaponics - Integrating Fish and Plant Culture. South Regional Aquaculture Center Publication No. 454. November 2006.
2. Gorder SV. Small-Scale Aquaculture and Aquaponics. Aquaponics Journal. Vol. 7, No. 3. 3rd Quarter, 2003.
3. United States Census TIGER/Line 2010 Wisconsin Roads shapefile. Accessed via Wisconsin Department of Natural Resources Public FTP server. ftp://dnrftp01.wi.gov/geodata/
4. Aquaponics Datbase. GitHub. Hayden Elza, 2015. https://github.com/HaydenElza/aquaponics-database/
5. Johnson DS. Local Optimization and the Traveling Salesman Problem. Automata, Languages and Programming Lecture Notes in Computer Science. Vol. 443, 1990, pp 446-461.