Killeen_CSPB3287Project-Copy1

April 26, 2021

1 Kitchen Inventory Database

Amanda Killeen
University of Colorado Boulder
CSPB 3287, Spring 2021

Link to presentation on YouTube

1.1 Project Goal

Taking the concept of reverse grocery list, where you have a list of things you normally buy and figure out what you are out of, I have built a database and dashboard showing which items I have on hand in my kitchen and where inventory is running low, and map which stores I need to visit, so that I stay within budget, don't over purchase, and can plan meals accordingly.

Using USDA food and brand data, this inventory tracks attributes such as where the food is stored (e.g. Pantry, Fridge, Freezer), category, brand, quantityOnHand, quantityNeeded and more. In addition to tracking the items, it flags when inventory is low and the item needs to be purchased.

This project was an opportunity to apply my database and SQL skills to parse a dataset into multiple relations efficiently as part of my Database Systems and Design course at the University of Colorado, Boulder.

1.2 Tools

- DataGrip - Database & SQL IDE. This was the main tool I used to create tables and write queries before transferring them to JupyterLab.
- JupyterLab + Python - Final report write-up
- SQLAlchemy - Handles database connection and query execution in Python
- Pandas - Used for query output, for better table visuals than SQL output.
- Tableau - Data visualization tool for creating visualizations related to the database, including the mapping of stores and item inventory status.
- CSV file Inventory - A single table will all inventory data data to be parsed into multiple relations using SQL.
- MySQL - SQL dialect to be used in creation and management of database
- Google Cloud Platform - Database hosting platform

[46]:

```
%load_ext sql
import sqlalchemy
import pandas as pd
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

[47]:
```
# connect to DB
db_string = 'mysql://root:k1tch3n@34.82.68.23:3306/inventory'
try:
 engine = sqlalchemy.create_engine(db_string);
 con = engine.connect()
 print("Connection, success!")
except Exception as exp:
 print("Create engine failed:", exp)
```

```
Connection, success!
```

1.3 Data Preparation

The starting point of my inventory, is a csv file, containing all of the data that I will be parsing into multiple tables. In order to create my the base of the inventory, I have used two datasets from the USDA FoodData Central:

- Branded Foods - October 2020: Foods with associated brand names
  - branded_food.csv
  - food.csv
- FNDDS 2017-2018 - October 2020: Generic Foods (produce, dairy, meat)
  - food.csv

I did some column deletion directly in Excel before creating the tables, then created a table schema for my base inventory table.

[52]:
```
# Create base table for inventory

con.execute (
'''
drop table if exists inventory_base;
create table if not exists inventory_base
(
    productID   int,
    UPC         text,
    productName text,
    category    text,
    brandName   text
);'''
)
```

[52]:
```
<sqlalchemy.engine.result.ResultProxy at 0x7ffa6a668610>
```

[53]:
```
# Populate inventory_base with a mix of random generic and brand name items for␣
 ↪a total of 1000
# Remove rows with incomplete data from both datasets
```

```
con.execute(
'''
insert into inventory_base

with brand as (
    select f.fdc_id                   as productID
         , b.gtin_upc                 as UPC
         , description                as productName
         , b.branded_food_category as category
         , b.brand_owner              as brandName
    from food_base f
             join branded_food_base b on f.fdc_id = b.fdc_id
    where b.brand_owner is not null
      and (b.branded_food_category is not null and b.brand_owner is not null)
      and f.fdc_id != 0
    order by rand()
    limit 800
),

generic as (
     select max(`FDC ID`)                          as productID
          , concat('033383', `ingredient code`) as UPC
          , `Ingredient description`            as productName
          , null                                as category
          , 'generic'                           as brandName
     from fresh_food
     group by 2, 3, 4, 5
     having max(`FDC ID`) != 0
     order by rand()
     limit 200
     )
select *
from brand
union
select *
from generic;
'''
)
```

```
[53]: <sqlalchemy.engine.result.ResultProxy at 0x7ffa557ae0d0>
```

```
[54]: # use pandas to preview first 5 rows in a tabular format

r = pd.read_sql(
'''
select *
from inventory_base
''', con
)
r.head()
```

```
[54]:    productID           UPC                         productName  \
      0     455506   41303060827                   OIL FASHIONED PIE
      1     570883  716519045011                         GREEN BEANS
      2     796805   46675013501  VANILLA COOKIE PIECES LOWFAT YOGURT
      3    1019175   36800374454                 SHREDDED HASH BROWNS
      4     598160   73723301211      ORGANIC STRAWBERRY FRUIT SPREAD

                                        category                brandName
      0                    Other Frozen Desserts          Supervalu, Inc.
      1         Pre-Packaged Fruit & Vegetables                   MANN'S
      2                                   Yogurt       The Yofarm Company
      3   French Fries, Potatoes & Onion Rings  Topco Associates, Inc.
      4                 Jam, Jelly & Fruit Spreads         DANISH ORCHARDS
```

```
[55]: #export to csv for manual data additions in Excel (Stores, fill-in missing␣
       ↪categories for generics)
      r.to_csv('inventory_base.csv')
```

After manually adding a number of fields to the dataset, I imported the .csv into MySQL using the gui available in DataGrip. I now have my final inventory in the form of a large table (preview below), but before I begin parsing into my relations, I want to do a bit of QA to make sure there aren't any duplicate values that will cause issues down the line.

```
[129]: # Preview of inventory_final table:
       f = pd.read_sql(
           '''
           select *
           from inventory_final
           order by productID;
           ''', con
       )
       f.head()
```

```
[129]:    productID           UPC                                      productName  \
       0        NaN           NaN                                             None
       1        NaN           NaN                                             None
       2        NaN           NaN                                             None
       3   167606.0  3.338331e+09  Sweet Potatoes, french fried, frozen as packag…
       4   167681.0  3.338342e+09  Beverages, fruit-flavored drink, dry powdered …

          quantityNeeded  quantityOnHand  minimumQuantity brandName  \
       0             NaN             NaN              NaN      None
       1             NaN             NaN              NaN      None
```

```
2             NaN             NaN              NaN      None
3             2.0             1.0              1.0   generic
4             2.0             2.0              2.0   generic

            category storageLocation               store        addressLine1  \
0               None            None                None                None
1               None            None                None                None
2               None            None                None                None
3  Frozen Vegetables   Chest Freezer  Whole Foods Market  9940 NE Cornell Rd
4          Beverages          Pantry        Trader Joe's   2285 NW 185th Ave

  addressLine2       city stateAbbrev  zipCode
0         None       None        None      NaN
1         None       None        None      NaN
2         None       None        None      NaN
3         null  Hillsboro          OR  97124.0
4         null  Hillsboro          OR  97124.0
```

For my Product and ProductBrand tables, I plan to use the productID and UPC fields as unique keys, so I am checking that there are no duplicates and removing any bad rows. First, I will do a count of the IDs and compare to a count of distinct IDs.

[130]:

```
# Checking for duplicate rows or other anomalies

u = pd.read_sql('''select count(*)                  as totalRows
     , count(productID)          as productIDRows
     , count(distinct productID) as productIDUnique
     , count(UPC)                as upcRows
     , count(distinct UPC)       as upcCount
from inventory_final;
''', con)
u.head()
```

[130]:

```
   totalRows  productIDRows  productIDUnique  upcRows  upcCount
0       1003           1000             1000     1000       886
```

[131]:

```
# Explore why UPC's appear to have duplicates.

badUPC = pd.read_sql(
'''
select UPC
, count(UPC) as countUPC
from inventory_final
group by 1
having countUPC > 1
''', con

)
```

```
badUPC
```

```
[131]:               UPC  countUPC
        0   1.930000e+11         2
        1   6.370000e+11         3
        2   6.380000e+11         2
        3   6.590000e+11         3
        4   6.810000e+11         4
        5   6.880000e+11         3
        6   7.050000e+11         2
        7   7.090000e+11         7
        8   7.120000e+11         4
        9   7.190000e+11         2
        10  7.220000e+11         4
        11  7.230000e+11         3
        12  7.250000e+11         4
        13  7.270000e+11         2
        14  7.310000e+11         2
        15  7.420000e+11         3
        16  7.440000e+11         3
        17  7.540000e+11         2
        18  7.550000e+11         2
        19  7.590000e+11         2
        20  7.610000e+11         2
        21  7.620000e+11         2
        22  7.680000e+11         2
        23  7.810000e+11         3
        24  7.830000e+11         3
        25  7.870000e+11         2
        26  7.900000e+11         2
        27  8.100000e+11         4
        28  8.110000e+11         6
        29  8.120000e+11         2
        30  8.130000e+11         2
        31  8.140000e+11         2
        32  8.150000e+11         3
        33  8.170000e+11         2
        34  8.180000e+11         9
        35  8.190000e+11         3
        36  8.500000e+11         7
        37  8.510000e+11         4
        38  8.520000e+11         3
        39  8.530000e+11         6
        40  8.540000e+11         2
        41  8.560000e+11         5
        42  8.570000e+11         7
        43  8.580000e+11         4
```

```
44  8.590000e+11         4
45  8.850000e+11         3
46  8.880000e+11         2
47  8.890000e+11         6
48  8.990000e+11         2
```

It looks like there are some duplicate UPC values due to the use of some placeholder/dummy values, so those will need to be removed.

[132]:

```
# remove rows where the UPC is in the list of duplicate UPCs above.

con.execute(
'''
delete
from inventory_final t1
where t1.UPC in (
    select *
    from (select t2.UPC
          from inventory_final t2
          group by 1
          having count(t2.UPC) > 1) t3
);''')
```

[132]:

```
<sqlalchemy.engine.result.ResultProxy at 0x7fbbbeb205d0>
```

[133]:

```
#confirm deletion of rows

badUPCcheck = pd.read_sql(
'''
select UPC
, count(UPC) as countUPC
from inventory_final
group by 1
having countUPC > 1
''', con

)
badUPCcheck
```

[133]:

```
Empty DataFrame
Columns: [UPC, countUPC]
Index: []
```

[134]:

```
# check count of values for each column again

uCheck = pd.read_sql(
'''
select count(*)                     as totalRows
```

```
     , count(productID)           as productIDRows
     , count(distinct productID) as productIDUnique
     , count(UPC)                 as upcRows
     , count(distinct UPC)        as upcCount
from inventory_final;
''', con)
uCheck.head()
```

[134]:

```
   totalRows  productIDRows  productIDUnique  upcRows  upcCount
0        840            837              837      837       837
```

The totalRows count is showing some extract rows, which can also be seen in the table preview. To confirm those are the culprites, I will run a query to check for null ProductIDs to start:

[135]:

```
# find null rows

n = pd.read_sql(
'''
select *
from inventory_final
where productID is null
''', con)
n
```

[135]:

```
  productID   UPC productName quantityNeeded quantityOnHand minimumQuantity  \
0      None  None        None           None           None            None
1      None  None        None           None           None            None
2      None  None        None           None           None            None

  brandName category storageLocation store addressLine1 addressLine2  city  \
0      None     None            None  None         None         None  None
1      None     None            None  None         None         None  None
2      None     None            None  None         None         None  None

  stateAbbrev zipCode
0        None    None
1        None    None
2        None    None
```

Success! 3 null rows. I will delete these and then my base dataset should be cleaned and ready to parse.

[136]:

```
# delete null rows
con.execute(
'''
delete from inventory_final where productID is null;
'''
)
```

```
[136]: <sqlalchemy.engine.result.ResultProxy at 0x7fbbbeb7d850>
```

```
[137]: # final check to make sure counts align across the dataset

       nCheck = pd.read_sql(
       '''
       select count(*)                    as totalRows
            , count(productID)            as productIDRows
            , count(distinct productID)   as productIDUnique
            , count(UPC)                  as upcRows
            , count(distinct UPC)         as upcCount
       from inventory_final;
       ''', con)
       nCheck
```

```
[137]:    totalRows  productIDRows  productIDUnique  upcRows  upcCount
       0        837            837              837      837       837
```

All clean, now ready to parse!

1.4 Table Creation

The following will be a series of table creation and insertion statements to create the 8 relations that will compose the final database.

1.4.1 Create Storage Table and Insert Data

```
[48]: con.execute(
      '''
      drop table if exists Storage;

      create table if not exists Storage
      (
          storageID INTEGER PRIMARY KEY AUTO_INCREMENT,
          location  VARCHAR(32) UNIQUE NOT NULL,
          loadDate DATE DEFAULT (current_date())
      );

      truncate table Storage;

      insert into Storage (location, loadDate)
      select distinct storageLocation
           , current_date()                as loadDate
      from inventory_final;
      '''
      )
```

```
[48]: <sqlalchemy.engine.result.ResultProxy at 0x7ff7ac314fd0>
```

1.4.2 Check Storage Table

[49]:

```
storage = pd.read_sql(
'''
select *
from Storage
order by storageID
''', con)
storage.head()
```

[49]:

```
   storageID               location    loadDate
0          1          Chest Freezer  2021-04-27
1          2                 Pantry  2021-04-27
2          3           Refrigerator  2021-04-27
3          4  Refrigerator-Freezer  2021-04-27
4          5         Liquor Cabinet  2021-04-27
```

1.4.3 Create Category Table and Insert Data

[50]:

```
con.execute(
'''
drop table if exists  Category;

create table if not exists Category
(
    categoryID INTEGER PRIMARY KEY AUTO_INCREMENT,
    name       VARCHAR(64) UNIQUE NOT NULL,
    loadDate   DATE DEFAULT (current_date())
);

truncate table Category;

insert into Category (name, loadDate)
select distinct category
, current_date()  as loadDate
from inventory_final;
'''
)
```

[50]:

```
<sqlalchemy.engine.result.ResultProxy at 0x7ff7abfefc50>
```

[51]:

```
category = pd.read_sql (
'''
select *
from Category
order by categoryID
''', con
)
```

```
category.head()
```

[51]:
```
   categoryID                           name    loadDate
0           1              Frozen Vegetables  2021-04-27
1           2                         Cereal  2021-04-27
2           3                 Herbs & Spices  2021-04-27
3           4  Pepperoni, Salami & Cold Cuts  2021-04-27
4           5                           Soda  2021-04-27
```

1.4.4 Create Product Table and Insert Data

[52]:
```
con.execute(
'''
drop table if exists Product;
create table if not exists Product
(
    productID       INTEGER PRIMARY KEY, -- inventory_final.ProductID
    categoryID      INTEGER,
    name            VARCHAR(256)  NOT NULL,
    quantityNeeded  INTEGER             NOT NULL,
    quantityOnHand  INTEGER             NOT NULL,
    minimumQuantity INTEGER             NOT NULL,
    lowStock        BOOLEAN             NOT NULL,
    stockModifiedDate    DATETIME       NOT NULL,
    loadDate         DATE DEFAULT       (current_date()),
    foreign key (categoryID) references Category (categoryID)
        on delete cascade
        on update cascade
);

truncate table Product;

insert into Product (productID, categoryID, name, quantityNeeded,␣
↪quantityOnHand, minimumQuantity, lowStock,
                     stockModifiedDate, loadDate)
select distinct t1.productID
     , t2.categoryID
     , t1.productName                                          as name
     , t1.quantityNeeded
     , t1.quantityOnHand
     , t1.minimumQuantity
     , IF(t1.quantityOnHand <= t1.minimumQuantity, TRUE, FALSE) as lowStock
     , current_timestamp()                                     as␣
↪stockModifiedDate
     , current_date()                                          as loadDate
from inventory_final t1
```

```
         left join Category t2 on t1.category = t2.name;
'''
)
```

```
[52]: <sqlalchemy.engine.result.ResultProxy at 0x7ff7ac31b390>
```

```
[53]: product = pd.read_sql(
'''
select *
from Product
order by productID;
''', con)

product.head()
```

```
[53]:    productID  categoryID                                               name  \
0     167606           1  Sweet Potatoes, french fried, frozen as packag…
1     167681          31  Beverages, fruit-flavored drink, dry powdered …
2     167684          14  Creamy dressing, made with sour cream and/or b…
3     167689          23  Candies, MARS SNACKFOOD US, M&M's Peanut Butte…
4     167727          31     Beverages, ABBOTT, ENSURE PLUS, ready-to-drink

   quantityNeeded  quantityOnHand  minimumQuantity  lowStock  \
0               2               1                1         1
1               2               2                2         1
2               2               1                1         1
3               3               2                2         1
4               3               2                2         1

    stockModifiedDate    loadDate
0 2021-04-27 01:14:33  2021-04-27
1 2021-04-27 01:14:33  2021-04-27
2 2021-04-27 01:14:33  2021-04-27
3 2021-04-27 01:14:33  2021-04-27
4 2021-04-27 01:14:33  2021-04-27
```

1.4.5 Create StorageProduct Table and Insert Data

```
[54]: con.execute(
'''
drop table if exists StorageProduct;

create table if not exists StorageProduct
(
    storageProductID INTEGER PRIMARY KEY AUTO_INCREMENT,
    storageID        INTEGER NOT NULL,
    productID        INTEGER NOT NULL,
    loadDate         DATE DEFAULT (current_date()),
    foreign key (storageID) references Storage (storageID)
        on delete cascade
        on update cascade,
    foreign key (productID) references Product (productID)
        on delete cascade
        on update cascade
);

truncate table StorageProduct;

insert into StorageProduct (storageID, productID, loadDate)
select distinct t3.storageID
       , t2.productID
       , current_date()  as loadDate
from inventory_final t1
    left join Product t2 on t1.productID = t2.productID
    left join Storage t3 on t1.storageLocation = t3.location;
'''
)
```

```
[54]: <sqlalchemy.engine.result.ResultProxy at 0x7ff7abf23a10>
```

```
[55]: storageProduct = pd.read_sql(
      '''
      select *
      from StorageProduct
      order by storageProductID;
      ''', con
      )
      storageProduct.head()
```

```
[55]:    storageProductID  storageID  productID    loadDate
      0                 1          1     727914  2021-04-27
      1                 2          2     548394  2021-04-27
      2                 3          2     957539  2021-04-27
      3                 4          3     563600  2021-04-27
      4                 5          2     991513  2021-04-27
```

1.4.6 Create Brand Table and Insert Data

```
[56]: con.execute(
      '''
      drop table if exists Brand;

      create table if not exists Brand
      (
          brandID INTEGER PRIMARY KEY AUTO_INCREMENT,
          name    VARCHAR(64) UNIQUE NOT NULL,
          loadDate DATE DEFAULT (current_date())
      );

      truncate table Brand;

      insert into Brand(name, loadDate)
      select distinct brandName as name
      , current_date()                as loadDate
      from inventory_final;
      '''
      )
```

[56]:
```
<sqlalchemy.engine.result.ResultProxy at 0x7ff7ac3145d0>
```

[57]:
```
brand = pd.read_sql(
'''
select *
from Brand
order by brandID
''', con
)
brand.head()
```

[57]:
```
   brandID                      name    loadDate
0        1               BEST CHOICE  2021-04-27
1        2              CAP'N CRUNCH  2021-04-27
2        3  ACH Food Companies, Inc.  2021-04-27
3        4                 UNDERWOOD  2021-04-27
4        5  Coca-Cola USA Operations  2021-04-27
```

1.4.7 Create Store Table and Insert Data

[58]:
```
con.execute(
'''
drop table if exists Store;

create table if not exists Store
(
    storeID      INTEGER PRIMARY KEY AUTO_INCREMENT,
    name         VARCHAR(32) UNIQUE NOT NULL,
    addressLine1 VARCHAR(256),
    addressLine2 VARCHAR(256),
    city         VARCHAR(64),
    stateAbbrev  VARCHAR(2),
    zipCode      VARCHAR(5),
    loadDate     DATE DEFAULT (current_date())
);

truncate table Store;

insert into Store (name, addressLine1, addressLine2, city, stateAbbrev,␣
 ↪zipCode, loadDate)
select distinct store as name
, addressLine1
, addressLine2
, city
, stateAbbrev
, zipCode
, current_date()  as loadDate
from inventory_final;
'''
)
```

[58]:
```
<sqlalchemy.engine.result.ResultProxy at 0x7ff7ac2b5950>
```

[59]:
```
store = pd.read_sql(
'''
select *
from Store
order by storeID;''', con
)

store.head()
```

[59]:
```
   storeID                name                  addressLine1 addressLine2  \
0        1          Fred Meyer              2200 E Baseline St         null
1        2  Whole Foods Market              9940 NE Cornell Rd         null
2        3             Walmart                  220 N Adair St         null
3        4         Trader Joe's               2285 NW 185th Ave         null
4        5              Target  2295 SE Tualatin Valley Hwy,         null

        city stateAbbrev zipCode    loadDate
0  Cornelius          OR   97113  2021-04-27
1  Hillsboro          OR   97124  2021-04-27
2  Cornelius          OR   97113  2021-04-27
3  Hillsboro          OR   97124  2021-04-27
4  Hillsboro          OR   97123  2021-04-27
```

1.4.8 Create ProductBrand Table and Insert Data

[60]:
```
con.execute(
'''
drop table if exists ProductBrand;

create table if not exists ProductBrand
(
    productBrandID DOUBLE PRIMARY KEY, #UPC
    productID      INTEGER NOT NULL,    #FDC_ID
    brandID        INTEGER NOT NULL,
    loadDate       DATE DEFAULT (current_date()),
    foreign key (productID) references Product (productID)
        on delete cascade
        on update cascade,
    foreign key (brandID) references Brand (brandID)
        on delete cascade
        on update cascade
);

truncate table ProductBrand;

insert into ProductBrand(productBrandID, productID, brandID, loadDate)
select distinct t1.UPC as productBrandID
, t2.productID
, t3.brandID
, current_date()  as loadDate
from inventory_final t1
    left join Product t2 on t1.productID = t2.productID
    left join Brand t3 on t1.brandName = t3.name;
'''
)
```

[60]:
```
<sqlalchemy.engine.result.ResultProxy at 0x7ff7ac359510>
```

[61]:
```
productBrand = pd.read_sql(
'''
select *
from ProductBrand
order by productBrandID;
''', con
)

productBrand.head()
```

[61]:
```
   productBrandID  productID  brandID    loadDate
0       2025988.0     727914        1  2021-04-27
1       3032802.0     548394        2  2021-04-27
```

```
2       4031918.0      957539         3  2021-04-27
3       4783830.0      563600         4  2021-04-27
4       4909806.0      991513         5  2021-04-27
```

1.4.9 Create ProductBrandStore Table and Insert Data

[62]:
```
con.execute(
'''
drop table if exists ProductBrandStore;

create table if not exists ProductBrandStore
(
    productBrandStoreID INTEGER PRIMARY KEY AUTO_INCREMENT,
    productBrandID      DOUBLE NOT NULL,
    storeID             INTEGER NOT NULL,
    loadDate            DATE DEFAULT (current_date()),
    foreign key (productBrandID) references ProductBrand (productBrandID)
        on delete cascade
        on update cascade,
    foreign key (storeID) references Store (storeID)
        on delete cascade
        on update cascade
);

truncate table ProductBrandStore;

insert into ProductBrandStore(productBrandID, storeID, loadDate)
select distinct t2.productBrandID
              , t3.storeID
              , current_date()  as loaddDate
from inventory_final t1
         left join ProductBrand t2 on t1.UPC = t2.productBrandID and t1.
 ↪productID = t2.productID
         left join Store t3 on t1.Store = t3.name;
'''
)
```

[62]:
```
<sqlalchemy.engine.result.ResultProxy at 0x7ff7ac0ef490>
```

[63]:
```
productBrandStore = pd.read_sql(
'''
select *
from ProductBrandStore
order by productBrandStoreID;
''', con
)
productBrandStore.head()
```

```
[63]:    productBrandStoreID  productBrandID  storeID   loadDate
      0                    1       2025988.0        1  2021-04-27
      1                    2       3032802.0        1  2021-04-27
      2                    3       4031918.0        2  2021-04-27
      3                    4       4783830.0        1  2021-04-27
      4                    5       4909806.0        1  2021-04-27
```

1.5 Testing

1.5.1 Check Contraints

[64]:

```python
# checks that foreign key contraint on ProductBrand table, should get an error
try:
    con.execute(
    '''
    insert into ProductBrand (productBrandID, productID, brandID, loadDate)
    values (5, 4, 8, curdate());
    '''
    )
except Exception as exp:
 print("Table Update Failed:", exp)
```

```
Table Update Failed: (MySQLdb._exceptions.IntegrityError) (1452, 'Cannot add or
update a child row: a foreign key constraint fails (`inventory`.`ProductBrand`,
CONSTRAINT `ProductBrand_ibfk_1` FOREIGN KEY (`productID`) REFERENCES `Product`
(`productID`) ON DELETE CASCADE ON UPDATE CASCADE)')
[SQL:
    insert into ProductBrand (productBrandID, productID, brandID, loadDate)
    values (5, 4, 8, curdate());
    ]
(Background on this error at: http://sqlalche.me/e/gkpj)
```

[65]:

```python
# checks that dropping a table with foreign key dependencies will fail
try:
    con.execute(
    '''
    drop table if exists ProductBrand;
    '''
    )
except Exception as exp:
 print("Drop Table Failed:", exp)
```

```
Drop Table Failed: (MySQLdb._exceptions.OperationalError) (3730, "Cannot drop
table 'ProductBrand' referenced by a foreign key constraint
'ProductBrandStore_ibfk_1' on table 'ProductBrandStore'.")
[SQL:
    drop table if exists ProductBrand;
    ]
(Background on this error at: http://sqlalche.me/e/e3q8)
```

1.5.2 Join New Tables and Compare to Original Dataset

Here I take the newly created relations that I parsed from the original dataset and join them back together to check for data loss and to confirm that they align to the data in the original dataset.

```
[66]: # join all of the new relations into a single table and show the first 10 rows

newJoined = pd.read_sql(
'''
select P.productID
  , PB.productBrandID as UPC
  , P.name            as productName
  , P.quantityNeeded
  , P.quantityOnHand
  , P.minimumQuantity
  , B.name            as brandName
  , C.name            as category
  , S2.location       as storageLocation
  , S.name            as Store
  , S.addressLine1
  , S.addressLine2
  , S.city
  , S.stateAbbrev
  , S.zipCode
from ProductBrandStore PBS
      left join ProductBrand PB on PBS.productBrandID = PB.productBrandID
      left join Store S on PBS.storeID = S.storeID
      left join Brand B on PB.brandID = B.brandId
      left join Product P on P.productID = PB.productID
      left join Category C on P.categoryID = C.categoryID
      left join StorageProduct SP on P.productID = SP.productID
      left join Storage S2 on SP.storageID = S2.storageID
order by 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15;
''', con
)
newJoined.head(10)
```

```
[66]:    productID           UPC                                        productName  \
0     167606  3.338331e+09  Sweet Potatoes, french fried, frozen as packag…
1     167681  3.338342e+09  Beverages, fruit-flavored drink, dry powdered …
2     167684  3.338342e+09  Creamy dressing, made with sour cream and/or b…
3     167689  3.338342e+09  Candies, MARS SNACKFOOD US, M&M's Peanut Butte…
4     167727  3.338344e+09     Beverages, ABBOTT, ENSURE PLUS, ready-to-drink
5     167735  3.338344e+09                     Cheese, mozzarella, low sodium
6     167781  3.338394e+08                                      Candied fruit
7     167792  3.338395e+08                       Orange Pineapple Juice Blend
8     167847  3.338310e+09  Pork, fresh, shoulder, arm picnic, separable l…
9     167957  3.338319e+09                              Syrup, fruit flavored
```

```
   quantityNeeded  quantityOnHand  minimumQuantity brandName  \
0               2               1                1   generic   
1               2               2                2   generic   
2               2               1                1   generic   
3               3               2                2   generic   
4               3               2                2   generic   
5               2               1                1   generic   
6               3               2                2   generic   
7               2               1                1   generic   
8               1               1                1   generic   
9               3               3                3   generic   

                                          category       storageLocation  \
0                                Frozen Vegetables         Chest Freezer   
1                                        Beverages                Pantry   
2                      Salad Dressing & Mayonnaise                Pantry   
3                                            Candy                Pantry   
4                                        Beverages                Pantry   
5                                            Dairy          Refrigerator   
6                                            Candy                Pantry   
7  Fruit & Vegetable Juice, Nectars & Fruit Drinks          Refrigerator   
8   Meat/Poultry/Other Animals  Prepared/Processed  Refrigerator-Freezer   
9  Fruit & Vegetable Juice, Nectars & Fruit Drinks          Refrigerator   

                Store        addressLine1 addressLine2       city stateAbbrev  \
0  Whole Foods Market  9940 NE Cornell Rd         null  Hillsboro          OR   
1        Trader Joe's   2285 NW 185th Ave         null  Hillsboro          OR   
2  Whole Foods Market  9940 NE Cornell Rd         null  Hillsboro          OR   
3          Fred Meyer  2200 E Baseline St         null  Cornelius          OR   
4          Fred Meyer  2200 E Baseline St         null  Cornelius          OR   
5  Whole Foods Market  9940 NE Cornell Rd         null  Hillsboro          OR   
6          Fred Meyer  2200 E Baseline St         null  Cornelius          OR   
7  Whole Foods Market  9940 NE Cornell Rd         null  Hillsboro          OR   
8          Fred Meyer  2200 E Baseline St         null  Cornelius          OR   
9          Fred Meyer  2200 E Baseline St         null  Cornelius          OR   

  zipCode  
0   97124  
1   97124  
2   97124  
3   97113  
4   97113  
5   97124  
6   97113  
7   97124  
8   97113  
```

```
9   97113
```

[67]:

```
# show the first 10 rows of the original dataset inventory_final

original = pd.read_sql(
'''
select *
from inventory_final
order by 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15;
''', con
)
original.head(10)
```

[67]:

```
   productID           UPC                                          productName  \
0     167606  3.338331e+09  Sweet Potatoes, french fried, frozen as packag…
1     167681  3.338342e+09  Beverages, fruit-flavored drink, dry powdered …
2     167684  3.338342e+09  Creamy dressing, made with sour cream and/or b…
3     167689  3.338342e+09  Candies, MARS SNACKFOOD US, M&M's Peanut Butte…
4     167727  3.338344e+09     Beverages, ABBOTT, ENSURE PLUS, ready-to-drink
5     167735  3.338344e+09                     Cheese, mozzarella, low sodium
6     167781  3.338394e+08                                      Candied fruit
7     167792  3.338395e+08                          Orange Pineapple Juice Blend
8     167847  3.338310e+09  Pork, fresh, shoulder, arm picnic, separable l…
9     167957  3.338319e+09                               Syrup, fruit flavored

   quantityNeeded  quantityOnHand  minimumQuantity brandName  \
0               2               1                1   generic
1               2               2                2   generic
2               2               1                1   generic
3               3               2                2   generic
4               3               2                2   generic
5               2               1                1   generic
6               3               2                2   generic
7               2               1                1   generic
8               1               1                1   generic
9               3               3                3   generic

                                     category       storageLocation  \
0                           Frozen Vegetables         Chest Freezer
1                                   Beverages                Pantry
2                 Salad Dressing & Mayonnaise                Pantry
3                                       Candy                Pantry
4                                   Beverages                Pantry
5                                       Dairy          Refrigerator
6                                       Candy                Pantry
7  Fruit & Vegetable Juice, Nectars & Fruit Drinks          Refrigerator
8   Meat/Poultry/Other Animals  Prepared/Processed  Refrigerator-Freezer
```

```
9  Fruit & Vegetable Juice, Nectars & Fruit Drinks         Refrigerator

                 store        addressLine1 addressLine2       city stateAbbrev  \
0  Whole Foods Market  9940 NE Cornell Rd         null  Hillsboro          OR
1        Trader Joe's   2285 NW 185th Ave         null  Hillsboro          OR
2  Whole Foods Market  9940 NE Cornell Rd         null  Hillsboro          OR
3          Fred Meyer  2200 E Baseline St         null  Cornelius          OR
4          Fred Meyer  2200 E Baseline St         null  Cornelius          OR
5  Whole Foods Market  9940 NE Cornell Rd         null  Hillsboro          OR
6          Fred Meyer  2200 E Baseline St         null  Cornelius          OR
7  Whole Foods Market  9940 NE Cornell Rd         null  Hillsboro          OR
8          Fred Meyer  2200 E Baseline St         null  Cornelius          OR
9          Fred Meyer  2200 E Baseline St         null  Cornelius          OR

  zipCode
0   97124
1   97124
2   97124
3   97113
4   97113
5   97124
6   97113
7   97124
8   97113
9   97113
```

The first 10 rows appear to match, but I also want to check that the row counts and counts of values in each column match. I created a new table Comparison and inserted a row for the original dataset along with it's count of rows and distinct values in each column, followed by a row of the same aggregations across the joined dataset.

[68]:
```
# creates table Comparison to hold counts from the original dataset and newly␣
 ↪joined parsed relations

con.execute(
'''
drop table if exists Comparison;

create table if not exists Comparison
(
    dataset                 text,
    TotalRows               int,
    Count_productID         int    null,
    Count_UPC               int    null,
    Count_productName       text   null,
    Count_quantityNeeded    int    null,
    Count_quantityOnHand    int    null,
    Count_minimumQuantity int    null,
    Count_brandName       text   null,
    Count_category        text   null,
    Count_storageLocation text   null,
    Count_Store           text   null,
    Count_addressLine1    text   null,
    Count_addressLine2    text   null,
    Count_city            text   null,
    Count_stateAbbrev     text   null,
    Count_zipCode         int    null

);

insert into Comparison (dataset, TotalRows, Count_productID, Count_UPC,␣
 ↪Count_productName, Count_quantityNeeded,
                        Count_quantityOnHand, Count_minimumQuantity,␣
 ↪Count_brandName, Count_category,
                        Count_storageLocation, Count_Store, Count_addressLine1,␣
 ↪Count_addressLine2, Count_city,
                        Count_stateAbbrev, Count_zipCode)
select 'inventory_final',
       count(*),
       count(distinct productID),
       count(distinct UPC),
       count(distinct productName),
       count(distinct quantityNeeded),
       count(distinct quantityOnHand),
       count(distinct minimumQuantity),
       count(distinct brandName),
       count(distinct category),
       count(distinct storageLocation),
       count(distinct Store),
       count(distinct addressLine1),
       count(distinct addressLine2),
       count(distinct city),
       count(distinct stateAbbrev),
       count(distinct zipCode)
from inventory_final
order by 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15;


insert into Comparison (dataset, TotalRows, Count_productID, Count_UPC,␣
 ↪Count_productName, Count_quantityNeeded,
                        Count_quantityOnHand, Count_minimumQuantity,␣
 ↪Count_brandName, Count_category,
                        Count_storageLocation, Count_Store, Count_addressLine1,␣
 ↪Count_addressLine2, Count_city,
                        Count_stateAbbrev, Count_zipCode)
select 'joinedTables'
     , count(*)
     , count(distinct P.productID)
     , count(distinct PB.productBrandID)
     , count(distinct P.name)
     , count(distinct P.quantityNeeded)
     , count(distinct P.quantityOnHand)
     , count(distinct P.minimumQuantity)
     , count(distinct B.name)
     , count(distinct C.name)
     , count(distinct S2.location)
     , count(distinct S.name)
     , count(distinct S.addressLine1)
     , count(distinct S.addressLine2)
     , count(distinct S.city)
     , count(distinct S.stateAbbrev)
     , count(distinct S.zipCode)
from ProductBrandStore PBS
      left join ProductBrand PB on PBS.productBrandID = PB.productBrandID
      left join Store S on PBS.storeID = S.storeID
      left join Brand B on PB.brandID = B.brandId
      left join Product P on P.productID = PB.productID
      left join Category C on P.categoryID = C.categoryID
      left join StorageProduct SP on P.productID = SP.productID
      left join Storage S2 on SP.storageID = S2.storageID
order by 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15;
'''
)
```

[68]:
```
<sqlalchemy.engine.result.ResultProxy at 0x7ff7ac69bb50>
```

[69]:
```
# Shows how the new joined relations compare to the original dataset
comparison = pd.read_sql(
'''
select *
from Comparison;
''', con
)
comparison.head()
```

[69]:
```
           dataset  TotalRows  Count_productID  Count_UPC Count_productName  \
0  inventory_final        837              837        837               832
1     joinedTables        837              837        837               832

   Count_quantityNeeded  Count_quantityOnHand  Count_minimumQuantity  \
0                     3                     4                      3
```

```
1                     3                     4                      3

  Count_brandName Count_category Count_storageLocation Count_Store  \
0             369            124                     5           7
1             369            124                     5           7

  Count_addressLine1 Count_addressLine2 Count_city Count_stateAbbrev  \
0                  7                  1          2                 1
1                  7                  1          2                 1

   Count_zipCode
0              3
1              3
```

All of the counts align so things are looking good!

1.5.3 Check for Missing and Matching Records between Datasets

[70]:

```
# Check for rows in the original dataset that may not be in the new dataset.

missingRecords = pd.read_sql(
'''
select count(*) as countMissingRecords
from inventory_final
where not exists(select P.productID
                      , PB.productBrandID as UPC
                      , P.name            as productName
                      , P.quantityNeeded
                      , P.quantityOnHand
                      , P.minimumQuantity
                      , B.name            as brandName
                      , C.name            as category
                      , S2.location       as storageLocation
                      , S.name            as Store
                      , S.addressLine1
                      , S.addressLine2
                      , S.city
                      , S.stateAbbrev
                      , S.zipCode
                 from ProductBrandStore PBS
                          left join ProductBrand PB on PBS.productBrandID = PB.
 ↪productBrandID
                          left join Store S on PBS.storeID = S.storeID
                          left join Brand B on PB.brandID = B.brandId
                          left join Product P on P.productID = PB.productID
                          left join Category C on P.categoryID = C.categoryID
                          left join StorageProduct SP on P.productID = SP.
 ↪productID
                          left join Storage S2 on SP.storageID = S2.storageID
                 order by 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15)

''', con
)
missingRecords.head()
```

[70]:

```
   countMissingRecords
0                    0
```

Yay! No missing records, now to check that all of the records match.

[71]:

```
# check that all of the rows in the original dataset are in the new dataset

matchingRecords = pd.read_sql(
'''
select count(*) as countMatchingRecords
from inventory_final
where exists (select P.productID
                      , PB.productBrandID as UPC
                      , P.name            as productName
                      , P.quantityNeeded
                      , P.quantityOnHand
                      , P.minimumQuantity
                      , B.name            as brandName
                      , C.name            as category
                      , S2.location       as storageLocation
                      , S.name            as Store
                      , S.addressLine1
                      , S.addressLine2
                      , S.city
                      , S.stateAbbrev
                      , S.zipCode
                 from ProductBrandStore PBS
                          left join ProductBrand PB on PBS.productBrandID = PB.
 ↪productBrandID
                          left join Store S on PBS.storeID = S.storeID
                          left join Brand B on PB.brandID = B.brandId
                          left join Product P on P.productID = PB.productID
                          left join Category C on P.categoryID = C.categoryID
                          left join StorageProduct SP on P.productID = SP.
 ↪productID
                          left join Storage S2 on SP.storageID = S2.storageID
                 order by 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15)
''', con)
```

```
matchingRecords.head()
```

[71]:

```
   countMatchingRecords
0                   837
```

[72]:

```
# Confirms that the number of matching records in the new dataset is equivalent␣
 ↪to the number of rows in the original dataset.

TR = comparison.iloc[1]['TotalRows'] # number of rows in original dataset
MR = matchingRecords.iloc[0]['countMatchingRecords'] # number of rows from the␣
 ↪joined dataset that match the original dataset.
print('Do the number of Matching Rows in the joined dataset equal the total␣
 ↪number of rows in the orignal dataset? ', TR == MR)
```

```
Do the number of Matching Rows in the joined dataset equal the total number of
rows in the orignal dataset?  True
```

1.6 Create Triggers

Now that my relations are built and match the orignal dataset, I'm going to add my triggers to handle data insertions and updates. I intentionally am doing this as a separate step because I wanted to make sure the core data all worked and matched before moving on to how changes to the table could impact it. Originally, I was going to have 3 triggers, two that controlled the lowStock value based on changes to the minimumQuantity and quantityOnHand values, then one to update the date that the stock was last modified, however I found I was able to add all 3 capabilities into a single trigger.

[73]:

```
# Trigger that updates the lowStock flag value when the quantityOnHand or␣
 ↪minimumQuantity is updated. It triggers the flag to change if needed and␣
 ↪updates the stock and date the stockModifiedDate.

con.execute(
'''
drop trigger if exists lowStockFlag;

create trigger lowStockFlag
    before update
    on inventory.Product
    for each row
begin
    if (NEW.quantityOnHand > NEW.minimumQuantity) then
        set NEW.lowStock = 0;
        set NEW.stockModifiedDate = current_timestamp();
    else
        set NEW.lowStock = 1;
        set NEW.stockModifiedDate = current_timestamp();
    end if;
end;
'''
)
```

```
[73]: <sqlalchemy.engine.result.ResultProxy at 0x7ff7ac321650>
```

1.7 Test Triggers: lowStockFlag

```
[74]: # Find a product that is lowStock = TRUE
ls = pd.read_sql(
'''select *
from Product
where lowStock = 1
limit 1;
''', con
)
ls.head()
```

```
[74]:    productID  categoryID                                               name  \
      0     167606           1  Sweet Potatoes, french fried, frozen as packag…

         quantityNeeded  quantityOnHand  minimumQuantity  lowStock  \
      0               2               1                1         1

          stockModifiedDate    loadDate
      0 2021-04-27 01:14:33  2021-04-27
```

```
[75]: # Update quantityOnHand
con.execute(
'''UPDATE Product
SET quantityOnHand = 2
WHERE productID = 167606;
''')
```

```
[75]: <sqlalchemy.engine.result.ResultProxy at 0x7ff7ac3592d0>
```

```
[76]: # Check that trigger flipped lowStock flag when quantityOnHand was updated to␣
 ↪exceed the minimumQuantity and stockModifieddate was updated
uls = pd.read_sql(
'''select *
from Product
where productID = 167606;
''', con
)
uls.head()
```

```
[76]:    productID  categoryID                                               name  \
      0     167606           1  Sweet Potatoes, french fried, frozen as packag…
```

```
   quantityNeeded  quantityOnHand  minimumQuantity  lowStock  \
0               2               2                1         0

    stockModifiedDate    loadDate
0 2021-04-27 01:14:56  2021-04-27
```

1.8 Queries and Aggregations

[77]:

```
# Identifies Stores associated with the most Products flagged as lowStock

storeLowStock = pd.read_sql(
'''select S.name as storeName
, count(distinct P.productID) as lowStockProducts
from ProductBrandStore PBS
         left join ProductBrand PB on PBS.productBrandID = PB.productBrandID
         left join Store S on PBS.storeID = S.storeID
         left join Brand B on PB.brandID = B.brandId
         left join Product P on P.productID = PB.productID
where lowStock = TRUE
group by 1
order by lowStockProducts desc
''', con
)
storeLowStock.head(10)
```

[77]:

```
            storeName  lowStockProducts
0          Fred Meyer               432
1  Whole Foods Market                76
2             Walmart                73
3              Target                63
4   Blooming Junction                10
5        Trader Joe's                10
6           Walgreens                 2
```

[78]:

```
# Identifies how stocked the kitchen as a whole is, using percentages.

kitchenStock = pd.read_sql(
'''with countAllProducts as (
    select count(distinct productID) as totalProducts
    from Product
),
     countLowstock as (
         select count(distinct productID) as totalLowProducts
         from Product
         where lowStock = 1
     )

select round(totalLowProducts / totalProducts * 100, 2) as percentKitchenStocked
from countAllProducts
         join countLowstock
''', con
)
kitchenStock.head()
```

[78]:

```
   percentKitchenStocked
0                  79.57
```

[79]:

```
# Identifies how stocked the storage locations are, using percentages.

storageStock = pd.read_sql('''with storageTotal as (
    select S2.location                 as storageLocation
         , count(distinct P.productID) as totalProducts
    from Product P
             left join StorageProduct SP on P.productID = SP.productID
             left join Storage S2 on SP.storageID = S2.storageID
    group by 1
),
     storageLow as (
         select S2.location                 as storageLocation
              , count(distinct P.productID) as totalProducts
         from Product P
                  left join StorageProduct SP on P.productID = SP.productID
                  left join Storage S2 on SP.storageID = S2.storageID
         where P.lowStock = 1
         group by 1
     )

select st.storageLocation
, round(sl.totalProducts/st.totalProducts * 100, 2) as percentStorageStocked
from storageTotal st
         join storageLow sl on st.storageLocation = sl.storageLocation
order by percentStorageStocked desc;
''', con)
storageStock.head()
```

[79]:

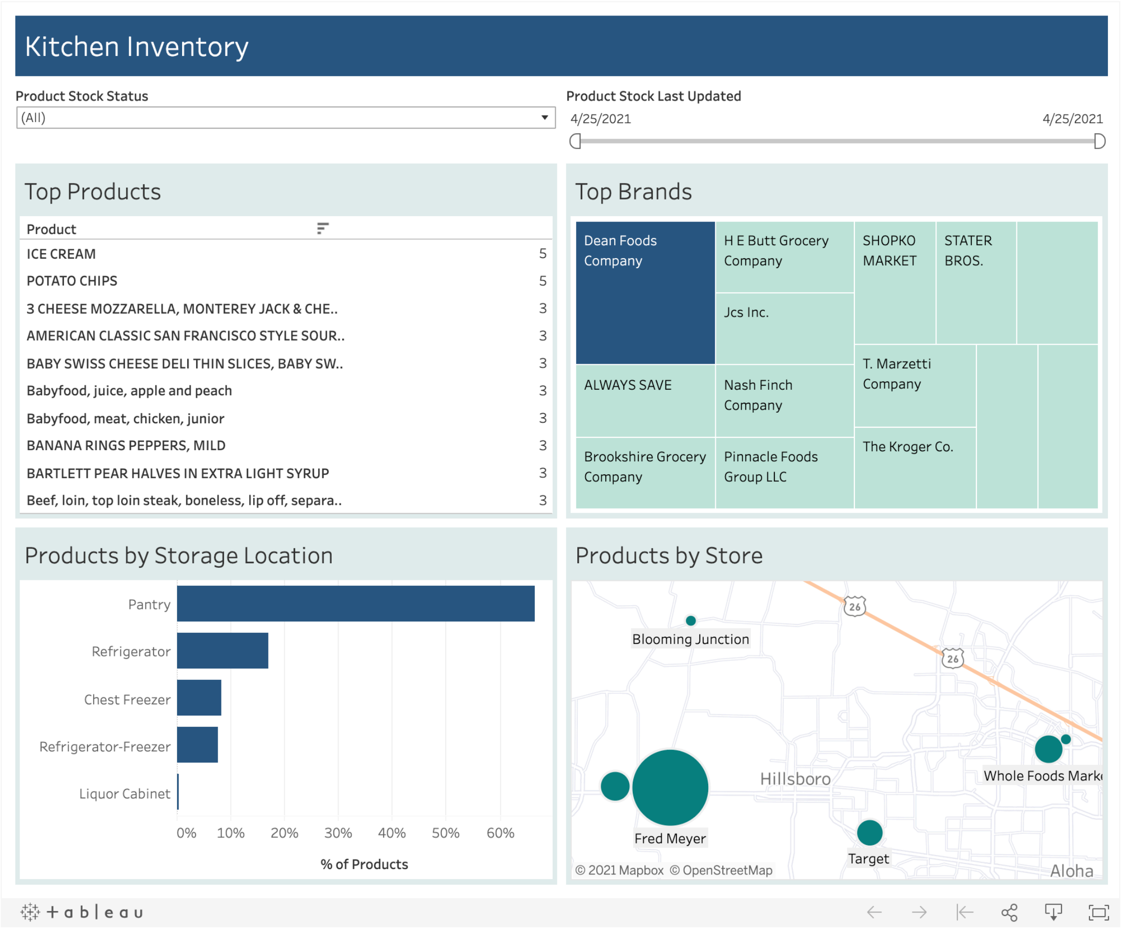
```
       storageLocation  percentStorageStocked
0        Liquor Cabinet                 100.00
1          Refrigerator                  80.28
2                Pantry                  79.89
3  Refrigerator-Freezer                  78.46
4         Chest Freezer                  75.36
```

[80]:

```
con.close()
```

1.9 Visualization

To support the database, I created a simple visualization on Tableau Public. The visualization tracks the top brands, products and stores, with the ability to filter based on kitchen stock availability and date the stock was last modified.



Kitchen Inventory

Product Stock Status: (All)

Product Stock Last Updated: 4/25/2021 – 4/25/2021

Top Products

Product	
ICE CREAM	5
POTATO CHIPS	5
3 CHEESE MOZZARELLA, MONTEREY JACK & CHE..	3
AMERICAN CLASSIC SAN FRANCISCO STYLE SOUR..	3
BABY SWISS CHEESE DELI THIN SLICES, BABY SW..	3
Babyfood, juice, apple and peach	3
Babyfood, meat, chicken, junior	3
BANANA RINGS PEPPERS, MILD	3
BARTLETT PEAR HALVES IN EXTRA LIGHT SYRUP	3
Beef, loin, top loin steak, boneless, lip off, separa..	3