FAST FOOD DATABASE

BY TRISH BEEKSMA, ALEX MELLO, AND MATTHEW SIEBOLD

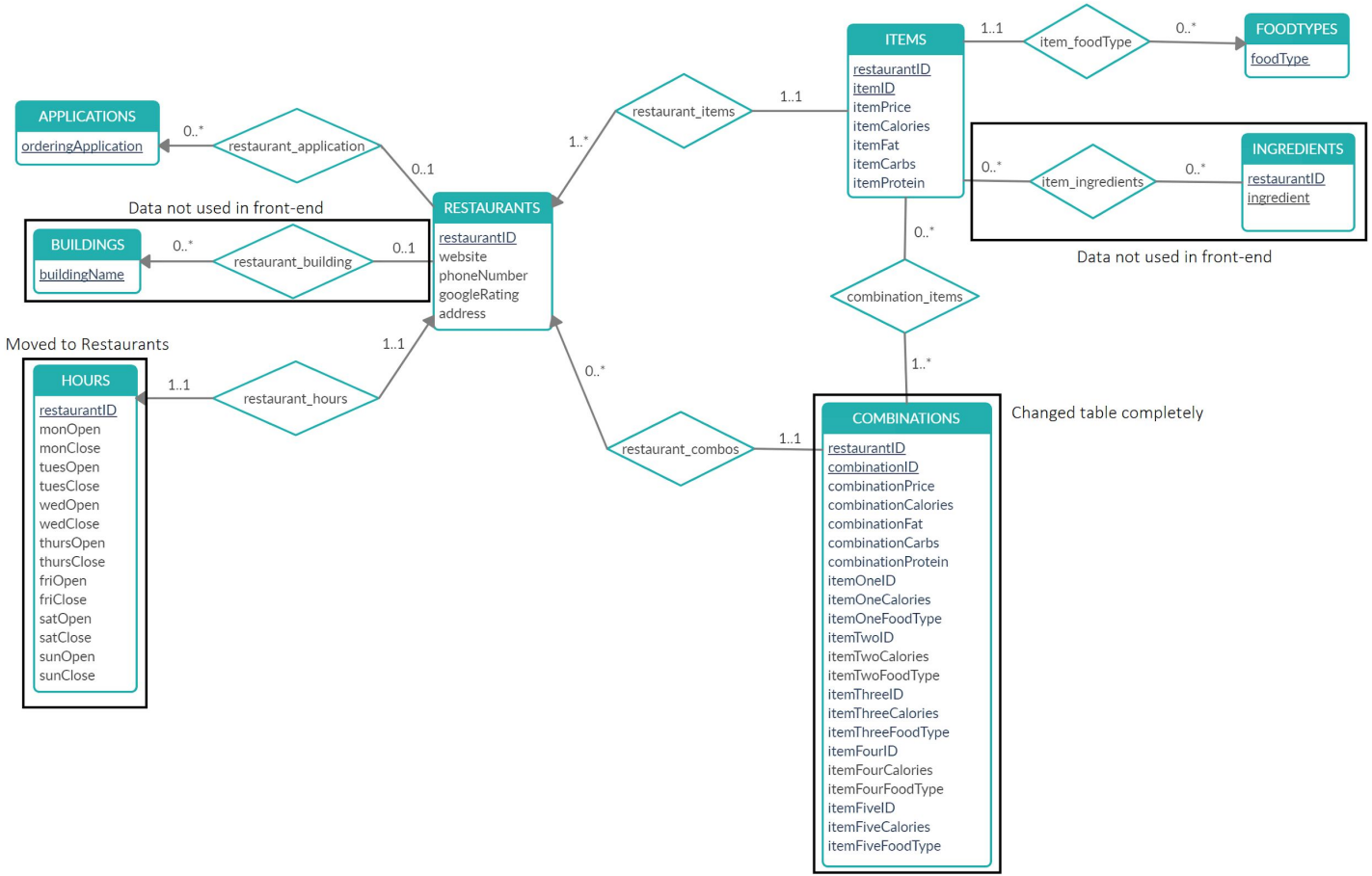
Original Application Description

- Our original project idea was to create a replica ordering application similar to DoorDash or Grubhub specifically for UNCW

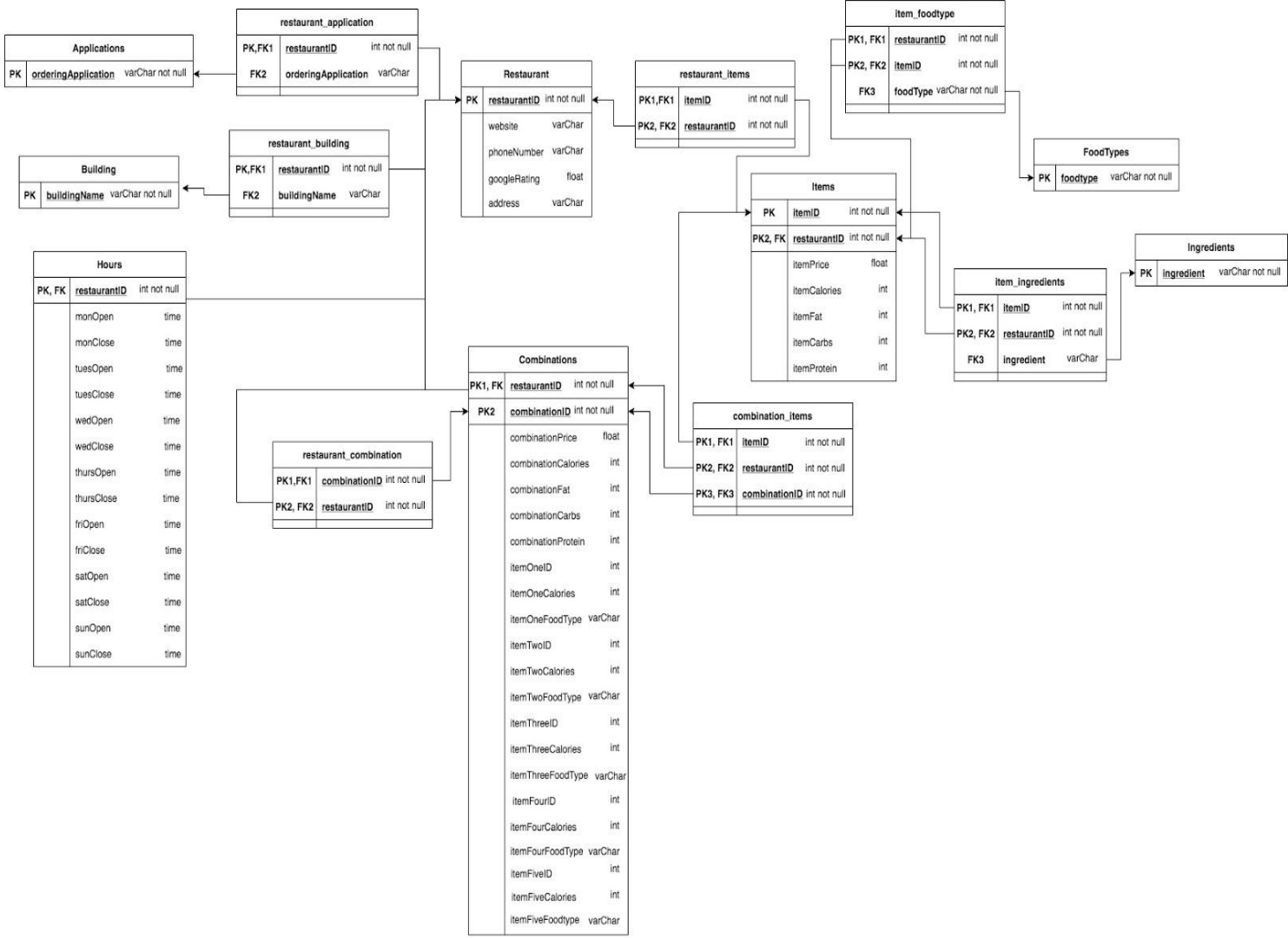
- Users would be able to view items from different restaurants
- Items would show information such as price and nutrition information
- Users could add items to their cart from different restaurants

- Restaurants could add and edit all of their information through the ordering application

Original ER-Diagram



ORIGINAL RELATIONAL MODEL



Problems With Original Idea

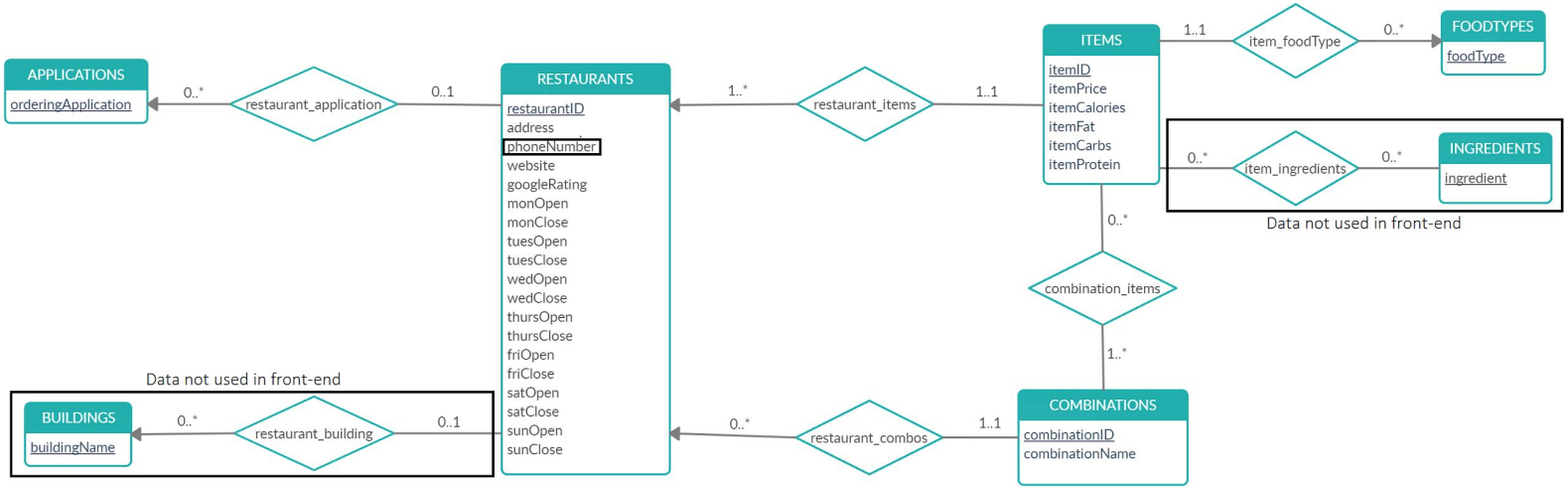
- Most on-campus restaurants do not display prices online
- Most on-campus restaurants do not display nutrition information online
- Some on-campus restaurants have menus that change weekly
- Restaurant combinations are completely different between restaurants

Updated Application Description

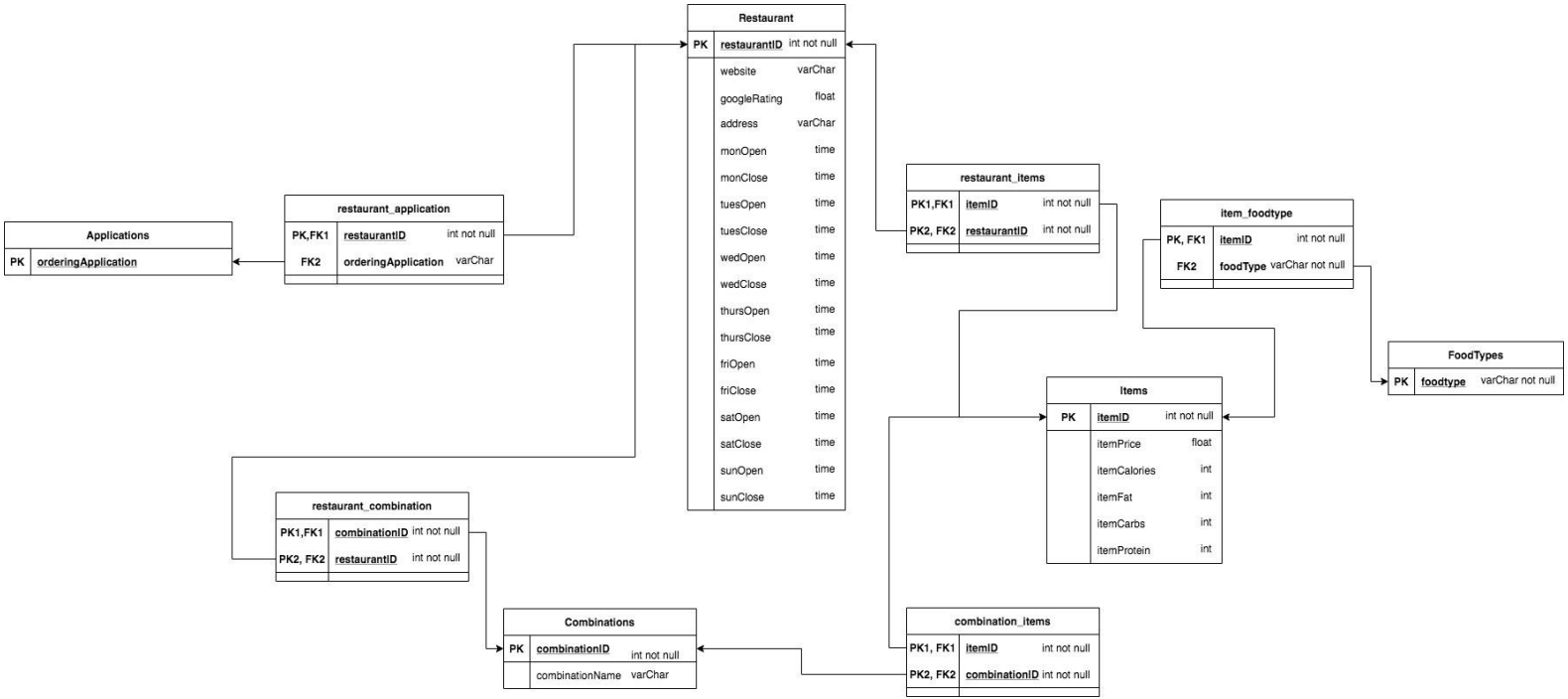
- Our updated idea is to mainly focus on displaying nutritional information of fast food restaurants

- We could build onto this idea which would lead to an ordering application similar to DoorDash or Grubhub

Updated E-R Diagram

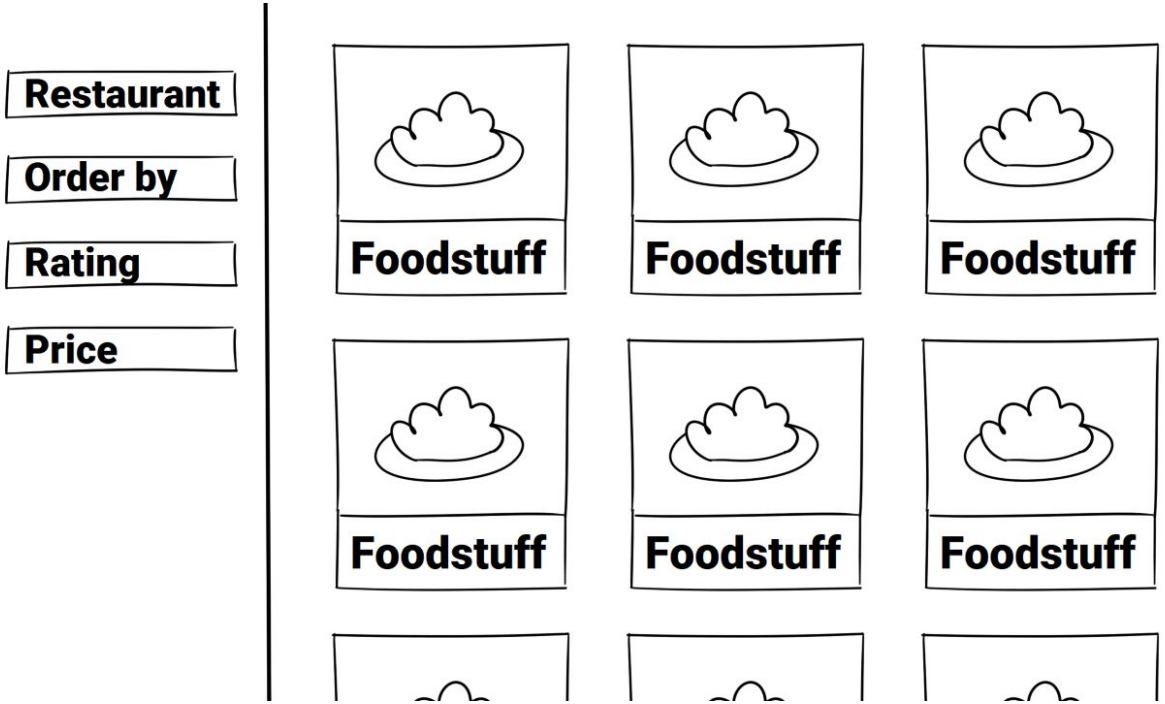


Updated Relational Model



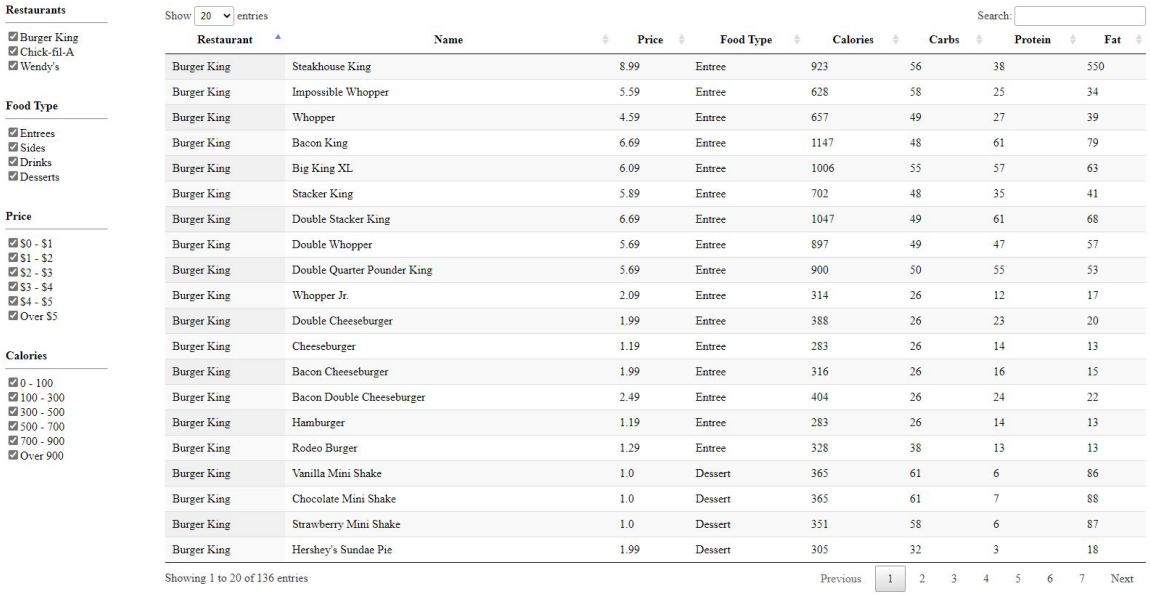
Original Sketch

- Design was inspired by popular online stores
- Filter criteria displayed on the left side
- Items displayed on the right side in an icon view



Final Design

- Icon view was left behind in favor of a straight-forward list view
- Item information (such as price and nutrition facts) are now listed alongside the item rather than being in a separate submenu



Restaurants

- ☑ Burger King
- ☑ Chick-fil-A
- ☑ Wendy's

Food Type

- ☑ Entrees
- ☑ Sides
- ☑ Drinks
- ☑ Desserts

Price

- ☑ $0 - $1
- ☑ $1 - $2
- ☑ $2 - $3
- ☑ $3 - $4
- ☑ $4 - $5
- ☑ Over $5

Calories

- ☑ 0 - 100
- ☑ 100 - 300
- ☑ 300 - 500
- ☑ 500 - 700
- ☑ 700 - 900
- ☑ Over 900

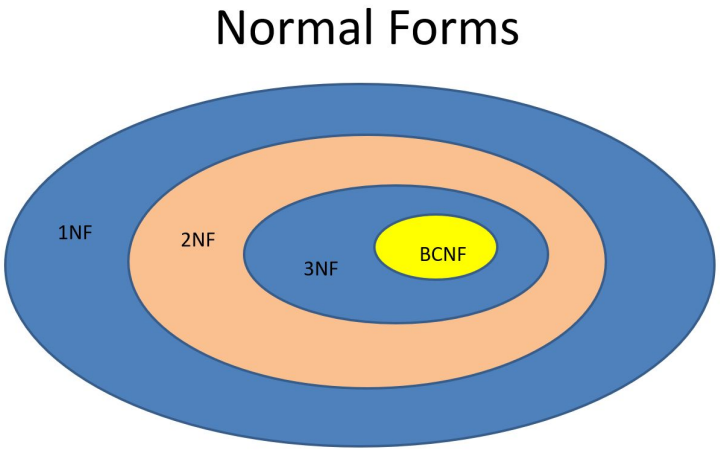
Show 20 entries Search:

Restaurant	Name	Price	Food Type	Calories	Carbs	Protein	Fat
Burger King	Steakhouse King	8.99	Entree	923	56	38	550
Burger King	Impossible Whopper	5.59	Entree	628	58	25	34
Burger King	Whopper	4.59	Entree	657	49	27	39
Burger King	Bacon King	6.69	Entree	1147	48	61	79
Burger King	Big King XL	6.09	Entree	1006	55	57	63
Burger King	Stacker King	5.89	Entree	702	48	35	41
Burger King	Double Stacker King	6.69	Entree	1047	49	61	68
Burger King	Double Whopper	5.69	Entree	897	49	47	57
Burger King	Double Quarter Pounder King	5.69	Entree	900	50	55	53
Burger King	Whopper Jr.	2.09	Entree	314	26	12	17
Burger King	Double Cheeseburger	1.99	Entree	388	26	23	20
Burger King	Cheeseburger	1.19	Entree	283	26	14	13
Burger King	Bacon Cheeseburger	1.99	Entree	316	26	16	15
Burger King	Bacon Double Cheeseburger	2.49	Entree	404	26	24	22
Burger King	Hamburger	1.19	Entree	283	26	14	13
Burger King	Rodeo Burger	1.29	Entree	328	38	13	13
Burger King	Vanilla Mini Shake	1.0	Dessert	365	61	6	86
Burger King	Chocolate Mini Shake	1.0	Dessert	365	61	7	88
Burger King	Strawberry Mini Shake	1.0	Dessert	351	58	6	87
Burger King	Hershey's Sundae Pie	1.99	Dessert	305	32	3	18

Showing 1 to 20 of 136 entries Previous 1 2 3 4 5 6 7 Next

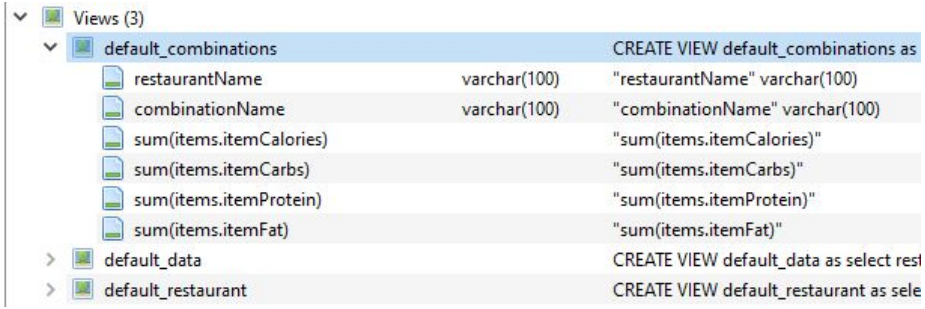
NORMALIZATION

- All tables are in BCNF because:
  - All attribute values are atomic
  - All non-key attributes are fully dependent on primary keys
  - All non-key attributes are mutually independent
  - All determinants are candidate keys



Rubric

Rubric




```
CREATE VIEW default_combinations as

select restaurant.restaurantName, combinations.combinationName,
sum(items.itemCalories), sum(items.itemCarbs), sum(items.itemProtein), sum(items.itemFat) from combinations
inner join restaurant_combinations on combinations.combinationID=restaurant_combinations.combinationID
inner join restaurant on restaurant_combinations.restaurantID=restaurant.restaurantID
inner join combination_items on combinations.combinationID = combination_items.combinationID
inner join items on combination_items.itemID = items.itemID
group by combinations.combinationID
```

Rubric

```
# default items query
defaultView = "select * from default_data"
```

```
# execute items query
c.execute(defaultView)
defaultData = c.fetchall()
defaultData = convert(defaultData)
```

```
# items webpage
@app.route('/', methods=['GET', 'POST'])
def home():
    # when form is submitted
    if request.method == 'POST':
        # get checkbox data
        checkboxData = getCheckboxes()
        checkboxData = combine(checkboxData)
        # get query string
        itemsQueryString = createItemsQuery(checkboxData)
        # get query data
        itemsQueryData = runQuery(itemsQueryString)

        # return webpage with updated data
        return render_template('frontEnd.html', data=itemsQueryData)
    # return webpage with default data
    return render_template('frontEnd.html', data=defaultData)
```

Rubric

```
select subquery.* from (

select restaurant.restaurantName, combinations.combinationName,
sum(items.itemCalories) as calories, sum(items.itemCarbs) as carbs, sum(items.itemProtein) as protein, sum(items.itemFat) as fat from combinations
inner join restaurant_combinations on combinations.combinationID=restaurant_combinations.combinationID
inner join restaurant on restaurant_combinations.restaurantID=restaurant.restaurantID
inner join combination_items on combinations.combinationID = combination_items.combinationID
inner join items on combination_items.itemID = items.itemID
group by combinations.combinationID
having ((calories>=0 and calories<=400) ) and ( (carbs>=0 and carbs<=40) ) and ( (protein>=0 and protein<=15) ) and ( (fat>=0 and fat<=20) )

) subquery;
```

Front-End

Food Database

Home | Combinations | Combination Items | Restaurants | Edit

HTML Form

Restaurants

- ☑ Burger King
- ☑ Chick-fil-A
- ☑ Wendy's

Food Type

- ☑ Entrees
- ☑ Sides
- ☑ Drinks
- ☑ Desserts

Price

- ☑ $0 - $1
- ☑ $1 - $2
- ☑ $2 - $3
- ☑ $3 - $4
- ☑ $4 - $5
- ☑ Over $5

Calories

- ☑ 0 - 100
- ☑ 100 - 300
- ☑ 300 - 500
- ☑ 500 - 700
- ☑ 700 - 900
- ☑ Over 900

Submitted through JavaScript

Start

HTML Table -> DataTable

Show 20 entries

Search:

Restaurant	Name	Price	Food Type	Calories	Carbs	Protein	Fat
Burger King	Steakhouse King	8.99	Entree	923	56	38	550
Burger King	Impossible Whopper	5.59	Entree	628	58	25	34
Burger King	Whopper	4.59	Entree	657	49	27	39
Burger King	Bacon King	6.69	Entree	1147	48	61	79
Burger King	Big King XL	6.09	Entree	1006	55	57	63
Burger King	Stacker King	5.89	Entree	702	48	35	41
Burger King	Double Stacker King	6.69	Entree	1047	49	61	68
Burger King	Double Whopper	5.69	Entree	897	49	47	57
Burger King	Double Quarter Pounder King	5.69	Entree	900	50	55	53
Burger King	Whopper Jr.	2.09	Entree	314	26	12	17
Burger King	Double Cheeseburger	1.99	Entree	388	26	23	20
Burger King	Cheeseburger	1.19	Entree	283	26	14	13
Burger King	Bacon Cheeseburger	1.99	Entree	316	26	16	15
Burger King	Bacon Double Cheeseburger	2.49	Entree	404	26	24	22
Burger King	Hamburger	1.19	Entree	283	26	14	13
Burger King	Rodeo Burger	1.29	Entree	328	38	13	13
Burger King	Vanilla Mini Shake	1.0	Dessert	365	61	6	86
Burger King	Chocolate Mini Shake	1.0	Dessert	365	61	7	88
Burger King	Strawberry Mini Shake	1.0	Dessert	351	58	6	87
Burger King	Hershey's Sundae Pie	1.99	Dessert	305	32	3	18

Showing 1 to 20 of 136 entries

Previous 1 2 3 4 5 6 7 Next

Jinja HTML Table

```
<table class="display nowrap" id="queryData">
    <thead>
    <tr>
        <th>Restaurant</th>
        <th>Name</th>
        <th>Price</th>
        <th>Food Type</th>
        <th>Calories</th>
        <th>Carbs</th>
        <th>Protein</th>
        <th>Fat</th>
    </tr>
    </thead>
    <tbody>
    {% for row in data %}
    <tr>
        {% for cell in row %}
        <td> {{ cell }}</td>
        {% endfor %}
    </tr>
    {% endfor %}
    </tbody>
</table>
```

Converting to DataTables

```
<!--datatables cdn-->
<script src="//cdn.datatables.net/1.10.22/js/jquery.dataTables.min.js"></script>
<!--datatables css cdn-->
<link rel="stylesheet" href="//cdn.datatables.net/1.10.22/css/jquery.dataTables.min.css">
```

```
<script>
    $(document).ready(function() {
        $('#queryData').DataTable( {
            "lengthMenu": [ 20, 50, 100 ],
            "pageLength": 20,
        } );
    } );

</script>
```

JavaScript for HTML Form

```
<script>
    $("#filterForm").on("change", "input:checkbox", function(){
        $("#filterForm").submit();
    });

</script>
```

```
<script>
    $(":checkbox").on("change", function(){
        var checkboxValues = {};
        $(":checkbox").each(function(){
            checkboxValues[this.id] = this.checked;
        });
        var now = new Date();
        var time = now.getTime();
        $.cookie('checkboxValues', checkboxValues)
    });

    function repopulateCheckboxes(){
        var checkboxValues = $.cookie('checkboxValues');
        if(checkboxValues){
            Object.keys(checkboxValues).forEach(function(element) {
                var checked = checkboxValues[element];
                $("#" + element).prop('checked', checked);
            });
        }
    }

    $.cookie.json = true;
    repopulateCheckboxes();

</script>
```

Updating Tables

```python
# items webpage
@app.route('/', methods=['GET', 'POST'])
def home():
    # when form is submitted
    if request.method == 'POST':
        # get checkbox data
        checkboxData = getCheckboxes()
        checkboxData = combine(checkboxData)
        # get query string
        itemsQueryString = createItemsQuery(checkboxData)
        # get query data
        itemsQueryData = runQuery(itemsQueryString)

        # return webpage with updated data
        return render_template('frontEnd.html', data=itemsQueryData)
    # return webpage with default data
    return render_template('frontEnd.html', data=defaultData)
```

```python
# get checkbox data
def getCheckboxes():
    checkboxData = []
    checkboxData.append(request.form.getlist('restaurant'))
    checkboxData.append(request.form.getlist('foodtype'))
    checkboxData.append(request.form.getlist('price'))
    checkboxData.append(request.form.getlist('calories'))
    checkboxData.append(request.form.getlist('carbs'))
    checkboxData.append(request.form.getlist('protein'))
    checkboxData.append(request.form.getlist('fat'))

    # return data as list of lists
    return checkboxData
```

```python
# run the query
def runQuery(queryString):
    # execute query
    c.execute(queryString)
    # returns list of tuples
    queryData = c.fetchall()
    # convert list of tuples -> tuple of tuples
    queryData = convert(queryData)

    # return data as tuple of tuples
    return queryData
```

Custom Query Function

```
# create the query string
def createItemsQuery(checkboxData):
    # empty query
    emptyQuery = "select restaurant.restaurantName, itemName, itemPrice, item_foodtype.foodtype, i
    from items \
    inner join item_foodtype on items.itemID=item_foodtype.itemID \
    inner join restaurant_items on items.itemID = restaurant_items.itemID \
    inner join restaurant on restaurant_items.restaurantID = restaurant.restaurantID where 1=0;"

    # base query
    queryString = "select subquery.* from (select restaurant.restaurantName, itemName, itemPrice,
    from items \
    inner join item_foodtype on items.itemID=item_foodtype.itemID \
    inner join restaurant_items on items.itemID = restaurant_items.itemID \
    inner join restaurant on restaurant_items.restaurantID = restaurant.restaurantID where ("
```

Custom Query Function

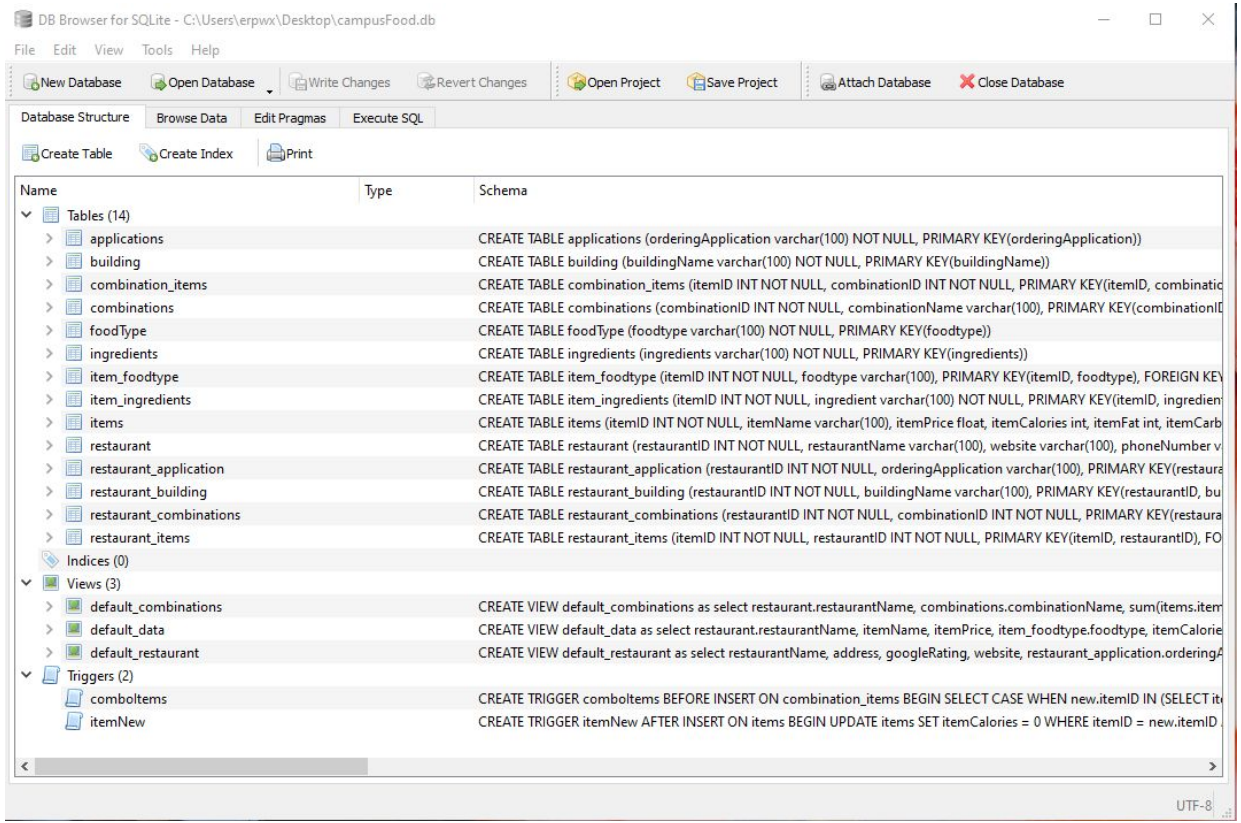
```
# restaurant part of query
restaurantPieces = 0
if checkboxData.__contains__('restaurant1'):
    queryString += 'restaurant.restaurantName = "Burger King" '
    restaurantPieces += 1
if checkboxData.__contains__('restaurant2'):
    if restaurantPieces > 0:
        queryString += 'or '
    queryString += 'restaurant.restaurantName = "Chick-Fil-A" '
    restaurantPieces += 1
if checkboxData.__contains__('restaurant3'):
    if restaurantPieces > 0:
        queryString += 'or '
    queryString += 'restaurant.restaurantName = "Wendys" '
    restaurantPieces += 1

if restaurantPieces == 0:
    return emptyQuery
queryString += ") and ("
```

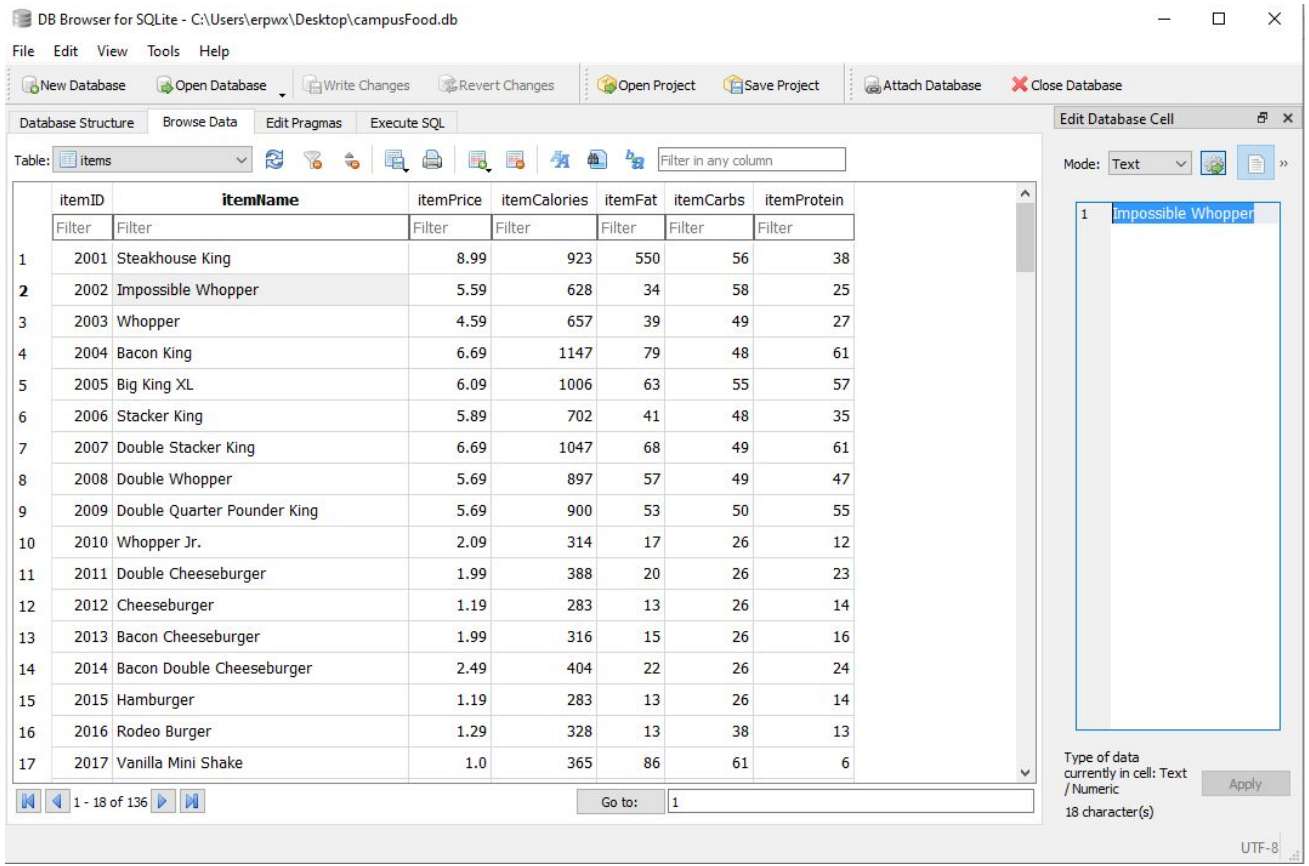
```
# foodtype part of query
foodtypePieces = 0
if checkboxData.__contains__('foodtype1'):
    if foodtypePieces > 0:
        queryString += 'or '
    queryString += 'item_foodtype.foodtype = "Entree" '
    foodtypePieces += 1
if checkboxData.__contains__('foodtype2'):
    if foodtypePieces > 0:
        queryString += 'or '
    queryString += 'item_foodtype.foodtype = "Side" '
    foodtypePieces += 1
if checkboxData.__contains__('foodtype3'):
    if foodtypePieces > 0:
        queryString += 'or '
    queryString += 'item_foodtype.foodtype = "Drink" '
    foodtypePieces += 1
if checkboxData.__contains__('foodtype4'):
    if foodtypePieces > 0:
        queryString += 'or '
    queryString += 'item_foodtype.foodtype = "Dessert" '
    foodtypePieces += 1

if foodtypePieces == 0:
    return emptyQuery
queryString += ") and ("
```

Review

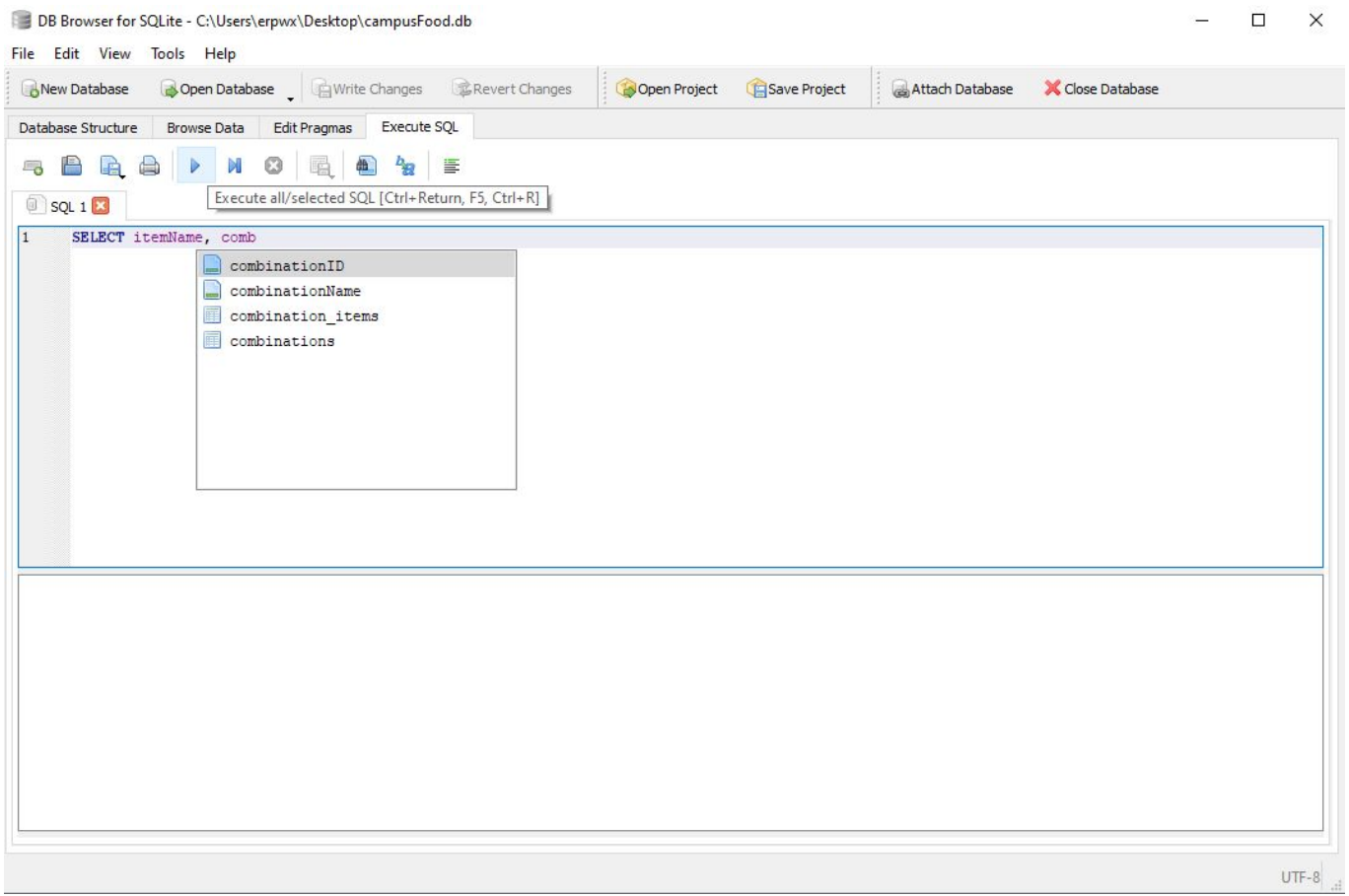
WHAT WE LEARNED

- Putting together a useful database requires extensive planning
- Learned some of Flask, SQLite, HTML, CSS, JavaScript
- DB Browser for SQLite is a visual tool for working with databases


DB Browser for SQLite - C:\Users\erpwx\Desktop\campusFood.db
File Edit View Tools Help
New Database
Open Database
Write Changes
Revert Changes
Open Project
Save Project
Attach Database
Close Database
Database Structure
Browse Data
Edit Pragmas
Execute SQL
Create Table
Create Index
Print
Name
Type
Schema
Tables (14)
applications
CREATE TABLE applications (orderingApplication varchar(100) NOT NULL, PRIMARY KEY(orderingApplication))
building
CREATE TABLE building (buildingName varchar(100) NOT NULL, PRIMARY KEY(buildingName))
combination_items
CREATE TABLE combination_items (itemID INT NOT NULL, combinationID INT NOT NULL, PRIMARY KEY(itemID, combinatic
combinations
CREATE TABLE combinations (combinationID INT NOT NULL, combinationName varchar(100), PRIMARY KEY(combinationI
foodType
CREATE TABLE foodType (foodtype varchar(100) NOT NULL, PRIMARY KEY(foodtype))
ingredients
CREATE TABLE ingredients (ingredients varchar(100) NOT NULL, PRIMARY KEY(ingredients))
item_foodtype
CREATE TABLE item_foodtype (itemID INT NOT NULL, foodtype varchar(100), PRIMARY KEY(itemID, foodtype), FOREIGN KE
item_ingredients
CREATE TABLE item_ingredients (itemID INT NOT NULL, ingredient varchar(100) NOT NULL, PRIMARY KEY(itemID, ingredien
items
CREATE TABLE items (itemID INT NOT NULL, itemName varchar(100), itemPrice float, itemCalories int, itemFat int, itemCarb
restaurant
CREATE TABLE restaurant (restaurantID INT NOT NULL, restaurantName varchar(100), website varchar(100), phoneNumber v
restaurant_application
CREATE TABLE restaurant_application (restaurantID INT NOT NULL, orderingApplication varchar(100), PRIMARY KEY(restaura
restaurant_building
CREATE TABLE restaurant_building (restaurantID INT NOT NULL, buildingName varchar(100), PRIMARY KEY(restaurantID, bu
restaurant_combinations
CREATE TABLE restaurant_combinations (restaurantID INT NOT NULL, combinationID INT NOT NULL, PRIMARY KEY(restaura
restaurant_items
CREATE TABLE restaurant_items (itemID INT NOT NULL, restaurantID INT NOT NULL, PRIMARY KEY(itemID, restaurantID), FO
Indices (0)
Views (3)
default_combinations
CREATE VIEW default_combinations as select restaurant.restaurantName, combinations.combinationName, sum(items.item
default_data
CREATE VIEW default_data as select restaurant.restaurantName, itemName, itemPrice, item_foodtype.foodtype, itemCalorie
default_restaurant
CREATE VIEW default_restaurant as select restaurantName, address, googleRating, website, restaurant_application.orderingA
Triggers (2)
comboItems
CREATE TRIGGER comboItems BEFORE INSERT ON combination_items BEGIN SELECT CASE WHEN new.itemID IN (SELECT it
itemNew
CREATE TRIGGER itemNew AFTER INSERT ON items BEGIN UPDATE items SET itemCalories = 0 WHERE itemID = new.itemID
UTF-8



	itemID	itemName	itemPrice	itemCalories	itemFat	itemCarbs	itemProtein
	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	2001	Steakhouse King	8.99	923	550	56	38
2	2002	Impossible Whopper	5.59	628	34	58	25
3	2003	Whopper	4.59	657	39	49	27
4	2004	Bacon King	6.69	1147	79	48	61
5	2005	Big King XL	6.09	1006	63	55	57
6	2006	Stacker King	5.89	702	41	48	35
7	2007	Double Stacker King	6.69	1047	68	49	61
8	2008	Double Whopper	5.69	897	57	49	47
9	2009	Double Quarter Pounder King	5.69	900	53	50	55
10	2010	Whopper Jr.	2.09	314	17	26	12
11	2011	Double Cheeseburger	1.99	388	20	26	23
12	2012	Cheeseburger	1.19	283	13	26	14
13	2013	Bacon Cheeseburger	1.99	316	15	26	16
14	2014	Bacon Double Cheeseburger	2.49	404	22	26	24
15	2015	Hamburger	1.19	283	13	26	14
16	2016	Rodeo Burger	1.29	328	13	38	13
17	2017	Vanilla Mini Shake	1.0	365	86	61	6


DB Browser for SQLite - C:\Users\erpwx\Desktop\campusFood.db
File
Edit
View
Tools
Help
New Database
Open Database
Write Changes
Revert Changes
Open Project
Save Project
Attach Database
Close Database
Database Structure
Browse Data
Edit Pragmas
Execute SQL
Execute all/selected SQL [Ctrl+Return, F5, Ctrl+R]
SQL 1
1
SELECT itemName, comb
combinationID
combinationName
combination_items
combinations
UTF-8

SURPRISES

- The same webpage can appear differently on different browsers

- It is very hard to format data in SQL queries
  - For example, formatting restaurant open and closing time to 5AM - 12AM

- You cannot store short-term data on front-end
  - Should always store checkbox data on back-end since it alters page
  - Data like google searches can be stored in cookies since it doesn't alter page
  - Very difficult to make cookies expire in less than a day

SURPRISES

- Checkboxes in Google Chrome vs Microsoft Edge

WHAT WE WOULD DO DIFFERENTLY

- Start the project with a smaller scope
- Make sure we have correct data ahead of time
- Create a system for editing database information ahead of time
- Plan ahead on how to store/pass data between front-end, back-end, and database
- Make the formatting more dynamic