Leslie’s Korean Catering Database

IEOR 115: Team 1

Anthony Blair, Camille Stuczynski, Jerry Cortez, Liam Campbell, Moses (Yu-chieh) Lee, Rebecca Martin, Stella Bao, Susheel Krishnamurthy, Waverly Runion, Wesley Jin
Presentation Menu

Client Summary
Background about project client

Project Benefits
Benefits of database system

EER Diagram
Entities & Relationships

Relational Schema
Access Implementation of database

Queries
Application of database contents

Normalization
Optimization
Leslie’s Korean Catering is a meal catering & delivery service based in Buena Park, CA.

Business is operated by Leslie Jin, who does all the marketing, cooking, deliveries, & accounting.

New menu each week that consists of entree, various side dishes, and soups.

Orders placed through social media.
Project Benefits

- Data Collection
- Best Practices
- Improve Workflow
Relational Tables
1. Menu_Item (ItemName, Price, RecipeName)
   a. Retired Menu Item (ItemName, DateRetired)
   b. Seasonal Menu Item (ItemName, Season)
2. Recipe (RecipeName, type_of_dish, cook_style, directions)
3. Ingredient (IngredientName, SupplierName, ing_category)
4. Food_Inventory_Item (FoodInventoryID, IngredientName, Quantity, Cost, SupplierName, PurchaseDate)
5. Package_Inventory_Item (PackagingName, Cost, quantity, SupplierName, PurchaseDate)
6. Supplier (SupplierName, Address, PhoneNumber)
7. Sale (SaleName, SaleNotes)
8. Week_Menu_Item (WeekMenuID, WeekStartDate, ItemName)
9. Cooking_Class (CourseID, ClassName, StartDate, EndDate, Price, Capacity)
10. Stop (Address, LocationNotes)
11. Social Platform (SocialPlatformName, UserName, NumFollowers)
12. Order (OrderID, OrderDate, DeliveryPersonID, CustomerID, DeliveryAddress, Delivery Zip, Total, PaidStatus, SpecialRequests)
   a. Pre-Order (OrderID, PreOrderDate)
   b. Recurring (OrderID, Day, Recurrence)
13. Employee(EmployeeID, EmployeeName, Birthdate, PhoneNumber, Address, VacationDays, Salary)
   a. Delivery Person(DeliveryPersonID, EmployeeID, VehicleName, VehicleType, LicensePlate, hourly_wage)
   b. Chef(ChefID, EmployeeID, salary, CulinarySchool, Experience, ChefPosition)
   c. Office Staff(OfficeStaffID, EmployeeID, JobTitle, salary)
14. Feedback(FeedbackID, date, CustomerID)
   d. Review(FeedbackID, text, stars, OrderID, SocialPlatform)
   e. Survey(FeedbackID, answers)
15. Customer(CustomerID, CustomerName, PhoneNumber, Address)
   f. Person (CustomerID, birthdate)
      i. College Student (PersonID, College)
      ii. Business Person (CustomerID, GroupCustomerID, Place_of_business)
      iii. Senior/Elder (CustomerID, DietaryRestrictions)
   g. Group (CustomerID, GroupName, GroupSize)
      i. Church (CustomerID, Church_address)
      ii. Business (CustomerID, Business_address)
      iii. Wedding (CustomerID, wedding_date, wedding_address)
      iv. Friend_group (CustomerID, referred_by)
16. Recipe_ingredient(RecipeName, IngredientName, Quantity)
17. Menu_item_uses_Food_inventory(ItemName, FoodInventoryID, IngredientName, quantity)
18. Order_uses_packaging_inventory_item(OrderID, PackagingName, QuantityUsed)
19. Week_menu_item_has_sale(WeekMenuItemID, SaleName, sale_percentage)
20. Order_Contents(OrderID, WeekMenuItemID, SaleName, Quantity)
21. Customer_signs_up_for_Cooking_Class(CustomerID, CourseID, SignUpDate)
22. Cooking_Class_advertised_on_Social_Platform(CourseID, SocialPlatformName, number_views, date_post)
23. Cooking_Class_taught_by_Chef(CourseID, ChefID)
24. Makes_videos_on_Social_Platform(ChefID, SocialPlatformName, NumViews)
25. Person_is_related_to_Person(CustomerID, CustomerID, relationship)
26. Customer_uses_Social_Platform(CustomerID, SocialPlatformName, username, followerCount)
27. Delivery_Stop(DeliveryPersonID, Address, Date_delivery, time)
28. Office_staff_reviews_feedback(OfficeStaffID, FeedbackID)
29. Chef_creates_recipe(ChefID, RecipeName)
Relational Schema
As implemented in Access
Relational Schema

Outline  Client  Benefits  EER  Schema  Queries  Normalization
Relational Schema Outline

Client

Benefits

EER

Schema

Queries

Normalization
Query 1: Dish-Review Correlation

What Dishes Have the Highest Correlation to Good Reviews?

Benefit: The Restaurant is able to see which of their dishes are most enjoyed by their customers.
Query 1 Methodology: Extract Data

```
SELECT Order_Contents.OrderID, 
       Week_Menu_Item.ItemName, 
       Review.Stars, Review.Text, 
FROM (Order_Contents LEFT JOIN Review ON Order_Contents.OrderID = Review.OrderID) 
    LEFT JOIN Week_Menu_Item ON Order_Contents.WeekItemID = Week_Menu_Item.WeekMenuItemId;
```
Query 1 Methodology: R Extension

```r
install.packages(c("plyr", "ggplot2", "Gally"))
library(plyr)
library(ggplot2)
library(Gally)
install.packages("car")
library(car)  # for VIF
install.packages("broom")
library(broom)
data <- read.csv("Query1.csv")
summary(data)
head(data)
ggscatmat(data, columns = 2:18, alpha = 0.8)
summary(mod1)
ggcoef(
  mod1,
  vline_color = "red",
  vline_linetype = "solid",
  errorbar_color = "blue",
  errorbar_height = .25,
  exclude_intercept = TRUE
)
vif(mod1)
```
### Query 1 Methodology: R Output

#### Coefficients:

| Term                          | Estimate | Std. Error | t value | Pr(>|t|) |
|-------------------------------|----------|------------|---------|----------|
| (Intercept)                   | 4.51905  | 0.55543    | 8.136   | 1.12e-06 *** |
| Tong.Katsu                    | 0.06308  | 0.83419    | 0.076   | 0.9408   |
| BSR                           | 0.56689  | 1.71895    | 0.330   | 0.7464   |
| Marinated.Pepper              | 0.42549  | 1.28149    | 0.332   | 0.7448   |
| Picked.Radish                 | -0.58213 | 1.54807    | -0.376  | 0.7125   |
| Bulgogi                       | -0.79763 | 0.76217    | -1.047  | 0.3131   |
| Seaweed.Soup                  | 0.27858  | 1.46920    | 0.190   | 0.8523   |
| Broccoli.with.bean            | -2.72142 | 2.33213    | -1.167  | 0.2627   |
| Fried.Anchovies               | 1.00000  | 1.81115    | 0.552   | 0.5896   |
| Bean.Paste.Pork.Bulgogi       | -1.47798 | 0.71980    | -2.053  | 0.0592 . |
| Eggplant.Salad                | 1.45893  | 1.10190    | 1.324   | 0.2067   |
| Jinsen.Chicken.Soup           | -0.47079 | 0.68571    | -0.687  | 0.5036   |
| Side.Dish.Trio                | -0.50974 | 1.49636    | -0.341  | 0.7384   |
| Vegetable.Stew                | -1.53853 | 0.92835    | -1.657  | 0.1197   |
| Pickled.Radish.and.Pepper     | 3.00000  | 1.81115    | 1.656   | 0.1199   |

#### Chart:

- Vegetables.Stew
- Tong.Katsu
- Side Dish.Trio
- Seaweed.Soup
- Pickled.Radish and Pepper
- Pickled Radish
- Marinated Pepper
- Jinsen Chicken Soup
- Garlic Black Beans
- Fried Anchovies
- Eggplant Salad
- Bulgogi
- RSR
- Broccoli with bean
- Bean Paste Pork Bulgogi

![Chart showing coefficient estimates](chart.png)
Query 1 Findings:

**Best 3 Dishes:**
- Pickled Radish & Pepper
- Eggplant Salad
- Marinated Pepper

**Worst 3 Dishes:**
- Vegetable Stew
- Garlic Black Beans
- Broccoli with Bean

Some of Leslie's Most Popular Dishes (actual):
Query 2: Translating Recipe Features to Profit

What type of dishes generate profit? What sells, and what can we learn from for future dishes or revisions?

**Benefit:** The restaurant can increase profit on their dishes by knowing how to edit the recipe.

**Dish attributes:**
1) Type of dish
2) Preparation methodology
3) Day of the month dish is ordered
4) Complexity of cooking directions
SELECT avg(mi.price*oc.quantity) as AVG_REVENUE_BEFORE_DISCOUNT,
  wmih.sale_percent as DISCOUNT,
  AVG((oc.quantity*miufi.quantity/fii.quantity)*fii.cost) as AVG_INGREDIENT_COST,
  SUM((oc.quantity*oupi.QuantityUsed/fii.quantity)*fii.cost) as AVG_PACKAGING_COST,
  mi.itemName,
  R.type_of_dish,
  R.cook_style,
  o.orderDate as orderDate
FROM Order o, Order_Contents oc,
  Week_Menu_Item wmi, Menu_Item mi, Recipe r, Recipe_Ingredient ri, Ingredient i,
  Food_Inventory_Item fii, Package_Inventory_Item pii,
  Menu_Item_Uses_Food_Inventory_Item miufi,
  Week_menu_item_has_sale wmihs
WHERE o.orderID = oc.orderID
  AND oc.weekItemID = wmi.WeekMenuItemId
  AND wmi.itemName = mi.itemName
  AND mi.RecipeName = r.RecipeName
  AND r.RecipeName = ri.RecipeName
  AND ri.IngredientName = i.IngredientName
  AND i.IngredientName = fii.IngredientName
  AND o.orderID = pii.OrderID
  AND pii.PackageInventoryNumber = fii.PackageInventoryNumber
  AND fiiPackageName = pii.PackageName
  AND miufi.FoodInventoryID = fii.FoodInventoryID
  AND wmih.salePercent = wmih.sale_percent
GROUP BY mi.itemName, Recipe.type_of_dish, Recipe.cook_style, Order.orderDate;
Query 2 Methodology:

normalization+PCA+K-means clustering

Check out the API at: Click [here](#) for PCA and [here](#) for K-means
Query 2: PCA+K-Means Code

Code snippet from Python demonstrating how PCA + K-means were conducted.

```python
from sklearn.decomposition import PCA as PCA
df_g3 = pd.read_csv('moseosdata.csv', names = ['revenue', 'direct'])

cf_q = (df_g3['day'] = df_g3['day'].astype('category'))
cf_q[('revenue')] = df_g3[('revenue')].astype('category')
cf_q[('profit')] = df_g3[('profit')].astype('category')
cf_q[('ingredient_cost')] = df_g3[('ingredient_cost')]
cf_q[('package_cost')] = df_g3[('package_cost')]
cf_q[('ingredient_cst')] = df_g3[('ingredient_cst')]
cf_q[('package_cst')] = df_g3[('package_cst')]
cf_q[('ingredient_ret')] = df_g3[('ingredient_ret')]
cf_q[('price')] = df_g3[('price')]
cf_q[('percentage')] = df_g3[('percentage')]
cf_q[('cost')] = df_g3[('cost')]
cf_q[('head')] = df_g3[('head')]

def q3 = df_g3[('revenue')] * (1 - df_g3[('package_cost')] / df_g3[('ingredient_ret')])
def q3[('directions')] = df_g3[('directions')].astype('category')
def q3[('profit')] = df_g3[('profit')].astype('int')

features = ['type_of_dish', 'cook_style', 'day', 'direct']
features = ['type_of_dish', 'cook_style', 'ingredient_cost', 'package_cost', 'profit']

t_train = scale(df_g3.loc[features])
t_train = pd.Series([df_g3[('profit')]]
```

from sklearn.decomposition import PCA as PCA
c PCA = PCA(n_components=2) # 2-dimensional PCA
t X_transformed = pd.DataFrame(PCA.fit_transform(t_train))
range1 = X_transformed[X_transformed[('Y_train') > 0] & (Y_train <= 10)]
range2 = X_transformed[X_transformed[('Y_train') > 10] & (Y_train <= 30)]
range3 = X_transformed[X_transformed[('Y_train') > 30] & (Y_train <= 80)]
range4 = X_transformed[X_transformed[('Y_train') > 80] & (Y_train <= 125)]
range5 = X_transformed[X_transformed[('Y_train') > 125] & (Y_train <= 150)]
kmeans = KMeans(n_clusters=n, n_init=max, max_iter=200)
kmeans.fit(X_transformed)

# # Plot the decision boundary. For that, we will assign
# # X_min, X_max = X_transformed.iloc[[0, 1]].min() - 1, X_transformed.iloc[[0, 1]].max(),
# # X_train, y_train = np.meshgrid(np.arange(X_min, X_max, h), np.arange(Y_min, Y_max, h))
# # Obtain labels for each point in mesh. Use last trained
# # Z = kmeans.predict(np.c_[X_train.ravel(), y_train.ravel()])
# # Z = Z.reshape(xx.shape)
# # Plot the decision boundary. For that, we will assign
# # X_min, X_max = X_transformed.iloc[[0, 1]].min() - 1, X_transformed.iloc[[0, 1]].max(),
# # X_train, y_train = np.meshgrid(np.arange(X_min, X_max, h), np.arange(Y_min, Y_max, h))
# # Obtain labels for each point in mesh. Use last trained
# # Z = kmeans.predict(np.c_[X_train.ravel(), y_train.ravel()])
# # Z = Z.reshape(xx.shape)
```
K-means on recipe features vs profitability

- Profit intervals:
  - 0 ≤ profit ≤ 10
  - 10 ≤ profit ≤ 30
  - 30 ≤ profit ≤ 80
  - 80 ≤ profit ≤ 125
  - 125 ≤ profit ≤ 180

PCA vector 1 vs PCA vector 2
Query 2: Findings

- The blue and purple regions correspond to menu items that yield the most profit.
- We should cater our recipe to 2 specific types of people: people with heavy taste and light taste.
- We should have more entrees. There aren’t that many served on our current weekly menu.
Query 3: Economic Order Quantities for Inventory

When should we purchase more ingredients? How many ingredients should be purchased during each shopping trip?

Benefit: The restaurant will be able to make better informed decisions about when to purchase supplies and how many units to purchase.
Query 3: EOQ

EOQ involves determining the optimal order quantity $Q$, and the optimal reorder point $T$ (the time between orders) for inventory

- $h =$ holding cost = decline in freshness [cost/unit day]
- $k =$ reorder cost = price of ingredient [cost/order]
- $D =$ deterministic demand [items/day]

$$Q = \sqrt{\frac{2kD}{h}}$$
$$T = \frac{Q}{D} = \sqrt{\frac{2k}{hD}}$$
Query 3:
SQL Data Pull for EOQ

```
SELECT fii.IngredientName, fii.PurchaseDate, fii.Cost, i.ing_category
FROM Food_Inventory_Item AS fii, Ingredient AS i
WHERE fii.IngredientName=i.IngredientName
ORDER BY fii.IngredientName;
```
Query 3:
SQL Data Pull for EOQ

```sql
SELECT ri.IngredientName,
       sum((ri.Quantity*oc.Quantity)) AS AmountUsed,
       (AmountUsed/30) AS DailyDemand
FROM Week_Menu_Item AS wmi, Recipe_Ingredients AS ri,
     Order_Contents AS oc, Menu_Item AS mi
WHERE wmi.ItemName=mi.ItemName
AND mi.RecipeName=ri.RecipeName
AND oc.WeekItemID=wmi.WeekMenuitemid
GROUP BY ri.IngredientName
ORDER BY ri.IngredientName;
```
Query 3: EOQ Matlab Code

Code snippet from Matlab demonstrating how the Q and T values are calculated.

DSP = Days since Purchase

NDemand is Daily Demand for each ingredient
Query 3: EOQ Results from Matlab

For Bean Sprouts purchased 8 days ago, we should order about 2 Bean Sprouts in 16 days

Q = 1.5803

T = 15.8035
Query 3: EOQ Results from Matlab

For Beef purchased 2 days ago, we should order about 6 pieces of Beef in 5 days

**Q = 5.4889**

**T = 4.8433**
Query 3: EOQ Results from Matlab

For Soy Sauce purchased 11 days ago, we should order about 4 more Soy Sauce packets in 4 days

Q = 3.7734

T = 4.1926
Query 4: Location Planning

What is the best location to place the next facility based on historical demand?

Benefit: Strategically place new location(s) that will reduce transportation cost/time.
Query 4:
Location Planning

Approach: P-Median Location Model

Idea: Minimize the demand-weighted average distances between demand nodes & location where facility will be placed → approximation to delivery cost

\[
\begin{align*}
X_j &= \begin{cases} 
1, & \text{if facility located at } j \in J \\
0, & \text{otherwise}
\end{cases} \\
Y_{i,j} &= \begin{cases} 
1, & \text{if demand node } i \in I \text{ assigned to facility located at } j \in J \\
0, & \text{otherwise}
\end{cases}
\end{align*}
\]

\[
\min \sum_{j \in J} \sum_{i \in I} h_i d_{i,j} Y_{i,j}
\]

s.t.

\[
\sum_{j \in J} Y_{i,j} = 1, \forall i \in I
\]

\[
Y_{i,j} - X_j \leq 0, \forall i \in I, j \in J
\]

\[
\sum_{j \in J} X_j = p
\]

\[
X_j \in \{0, 1\}, \forall j \in J
\]

\[
Y_{i,j} \in \{0, 1\}, \forall i \in I, j \in J.
\]

\(i\) = demand nodes

\(j\) = possible facility locations

\(h_i\) = demand at node \(i\)

\(d_{i,j}\) = distance between \(i\) & \(j\)
Query 4: Data for Location Planning

Demand Nodes:

```sql
SELECT o.DeliveryAddress, 
count(o.OrderID) AS numOrders, 
sum(o.Total) AS LocationRevenue
FROM [Order] AS o
GROUP BY o.DeliveryAddress;
```

Possible Facility Locations:

```sql
SELECT o.DeliveryZip, 
count(o.OrderID) AS numOrders, 
sum(o.Total) AS ZipRevenue
FROM [Order] AS o
GROUP BY o.DeliveryZip;
```
Query 4: Facility Location

Two ways: (1) Use revenue as demand
(2) Use number of orders as demand

Output (using heuristic):

<table>
<thead>
<tr>
<th>FacilityZip</th>
<th>X_i</th>
</tr>
</thead>
<tbody>
<tr>
<td>90620</td>
<td>0</td>
</tr>
<tr>
<td>90621</td>
<td>1</td>
</tr>
<tr>
<td>92653</td>
<td>0</td>
</tr>
<tr>
<td>92834</td>
<td>0</td>
</tr>
<tr>
<td>92835</td>
<td>0</td>
</tr>
<tr>
<td>94639</td>
<td>0</td>
</tr>
<tr>
<td>94704</td>
<td>0</td>
</tr>
</tbody>
</table>
Query 5: Social Media Campaigns

How many days before a cooking class should you post an advertisement in order to maximize enrollment?

Benefit: The restaurant will be able to use social media to maximize enrollment in cooking classes.
Query 5: Social Media Campaigns

```
SELECT Cooking_Class.CourseID,
    Sum((0.1)*Cooking_Class.Price+(0.9)*Cooking_Class.Capacity) AS Exclusivity_Score,
    Count(Customer_signs_up_for_Cooking_Class.CustomerID) AS Class_Enrollment,
    SUM(Cooking_Class.StartDate-Cooking_Class_advertised_on_Social_Platform.date_post) AS Post_days_before_class_start,
    MAX(Cooking_Class_advertised_on_Social_Platform.number_views) AS IG_views
FROM (Cooking_Class INNER JOIN Customer_signs_up_for_Cooking_Class
    ON Cooking_Class.[CourseID] = Customer_signs_up_for_Cooking_Class.[CourseID])
INNER JOIN Cooking_Class_advertised_on_Social_Platform
    ON Cooking_Class.[CourseID] = Cooking_Class_advertised_on_Social_Platform.[CourseID]
WHERE (((Cooking_Class_advertised_on_Social_Platform.SocialPlatformName="Instagram")
    AND ((Cooking_Class_advertised_on_Social_Platform.CourseID=Cooking_Class.CourseID))
    AND (Customer_signs_up_for_Cooking_Class.CourseID=Cooking_Class.CourseID))
GROUP BY Cooking_Class.CourseID;
```
Query 5: Social Media Campaigns

Exclusivity Score: Output of an equation based off of the Capacity and Price of a class

Idea: Calculate the relationship between exclusivity score and number of days prior to a class than an advertisement is posted

Graph this information to determine the most relevant relationships and how to utilize those relationships for future marketing campaigns
Query 5: Social Media Campaigns

Approach: Quadratic and Linear Regression

Methodology: We used a quadratic fit line and linear regression in MATLAB

```matlab
p1 = polyfit(exclusivity, studentsenrolled, 1);
p2 = polyfit(daysprior, studentsenrolled, 1);
p3 = polyfit(exclusivity, likes, 1);
p4 = polyfit(daysprior, likes, 1);

yfit1 = polyval(p1, exclusivity);
yfit2 = polyval(p2, daysprior);
yfit3 = polyval(p3, exclusivity);
yfit4 = polyval(p4, daysprior);

subplot(2,2,1);
plot(exclusivity, studentsenrolled, 'b*', exclusivity, yfit1, 'r-');
title('Students Enrolled vs Exclusivity');
xlabel('Exclusivity');
ylabel('Students Enrolled');
```
Query 5: Social Media Campaigns
Query 5: Social Media Campaigns

Predicting Success of future classes:

Results:
- Approximate linear relationship between likes and number of enrollments
- Quadratic relationship between exclusivity and number of likes
- Optimal time frame to post advertisements is around 45 days

Conclusion:
Advertise classes 45 days prior to class start date, and design classes with an exclusivity score of around 30 to maximize likes and enrollments.

```
function enrollments = enrollmentprediction( ex, days )

% ex = exclusivity score
% days = number of days post goes up prior to class
% This function can be used to predict enrollments for a class based on the class's exclusivity score and how many days it is advertised prior to the class.

likes = (-211/392)*(ex-28)^2 + 483;
enroll1 = (-1/192)*(days-48)^2 + 13.7;
enroll2 = (1/60)*likes + 3;
enrollments = .5*enroll1 + .5*enroll2;
```
Normalization

Decomposing our relations to optimize query time
Normalization Propositions—
1NF to 2NF

Rel. 20. Order_Contents(OrderID \textsuperscript{12}, WeekItemID \textsuperscript{8}, Quantity)

OrderID \rightarrow Quantity
WeekItemID \rightarrow Quantity;
partial dependency exists

2NF conversion:
Order_Contents(OrderID \textsuperscript{12}, WeekItemID \textsuperscript{8})
Order_Quantity(OrderID \textsuperscript{12}, Quantity)
Normalization Proposals—2NF to 3NF

Rel. 12. Order(OrderID, OrderDate, DeliveryPersonID\textsuperscript{13a}, CustomerID\textsuperscript{15}, DeliveryAddress, Total, PaidStatus)

OrderID → CustomerID
CustomerID → DeliveryAddress;
transitive dependency exists

3NF conversion:
Order(OrderID, OrderDate, DeliveryPersonID\textsuperscript{13a}, CustomerID\textsuperscript{15}, Total, PaidStatus)
OrderAddress(CustomerID\textsuperscript{15}, DeliveryAddress)
Thank You

Any Questions?