Pizza Parlor Point-Of-Sales System

CMPS 342 Database Systems

Chris Perry

Ruben Castaneda

Table of Contents

1 Pizza Parlor: Point-Of-Sales Database

1.1 *Description of Business*

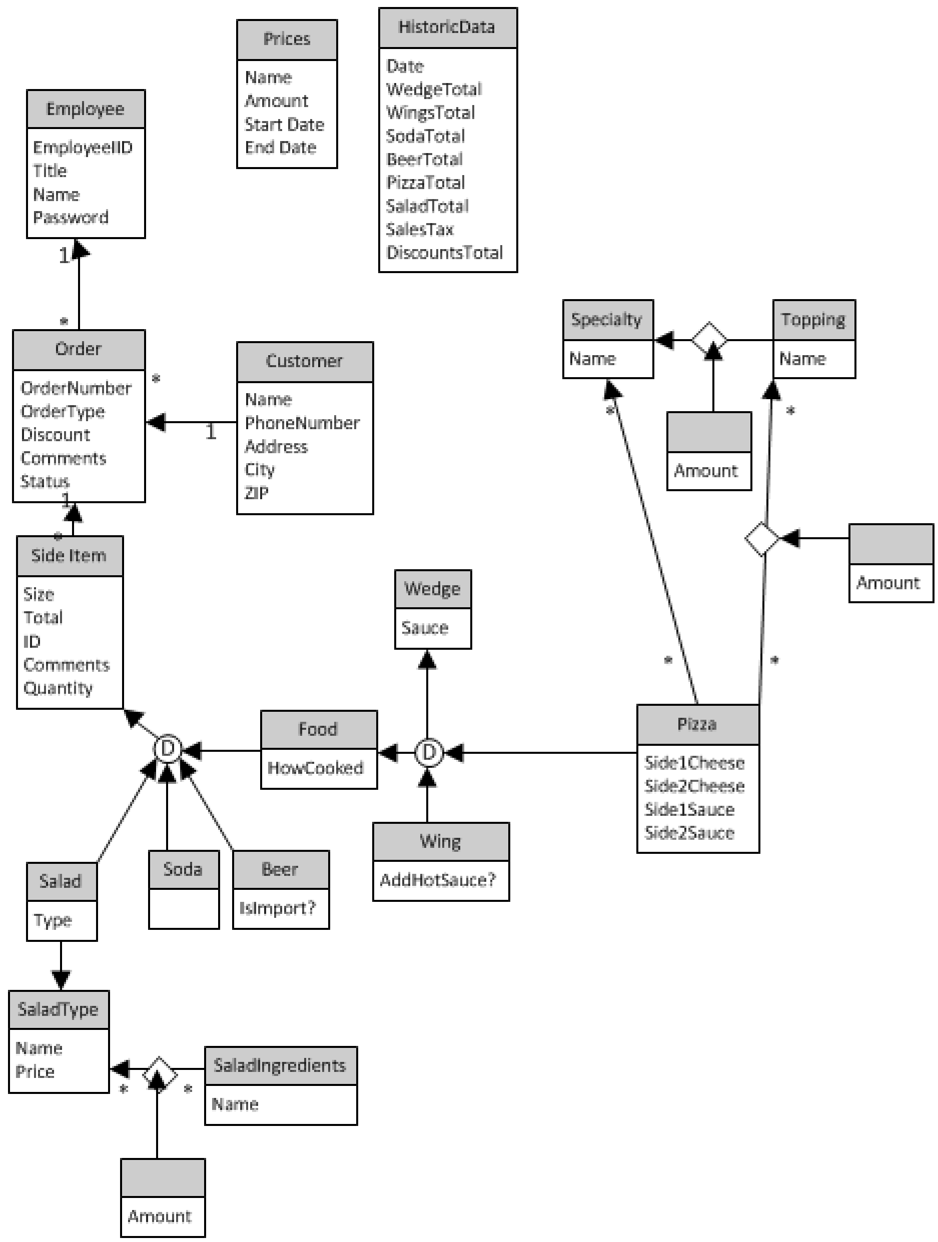
Using our group members personal experience and access, working at a pizza parlor, we were able to begin the fact-finding process. At the pizza parlor we interviewed management about the types of reports they use and would like to see in a Point-Of-Sales (POS) database. For this project we are modeling just the sales portion of the business. We are not including tasks and portions of the business that have to do with inventory, ordering, accounts payable, and labor, among others. Based on their answers we planned accordingly. From a business perspective we will be able to query historical data and produce total sales per day/week/month; sales based on pizza's sold; types of pizzas; salad and wings sales; soft drink and beer sales; most sold items per category.

From a user perspective, all employee's will be able to view orders based on time, total, name, order number, and/or telephone number. There three types of employee permissions. Management will have elevated permissions and will be split into two groups. Employees will only be able to take orders. Assistant managers will be able to discount prices, comp-orders, override transactions, and take orders. Managers will have the same permissions as assistant managers plus they can change prices, items, and quantities and view reports.

We analyzed and documented all the visible features we could find on the the existing POS database. To have an ideal POS interface we had all employees who operate the registers answer the following questionnaire:

1. *What do you like about the current POS database?*
2. *What do you dislike about the current POS database?*
3. *What would like to see added?*
4. *What would like to see modified?*
5. *What would like to see removed?*

1.2 *Conceptual Database*



2 Conceptual Database Design

2.1 *Entities*

Employee

The Employee entity holds employee's information and controls what permissions each employee has and what data they can view. The only candidate key is the EmployeeID attribute and thus the EmployeeID attribute is the primary key. An employee's name can be unique and it is not necessarily always going to be the case. Since the entity has a primary key, the Employee entity is a strong entity.

Name of Attribute	*Description*	*Domain/ Type*	*Value-Range*	*Default Value*	*NULL?*	*Unique?*	*Single or multi-value*	*Simple or Composite*
EmployeeID	A user's unique identifier	int	> 0	0	no	yes	single	simple
Title	A user's title determines their permissions and what data they can view	String	Employee, Assistant Manager, or Manager	Employee	no	no	Single	Simple
Name	A user's name	String		Empty string	no	no	Single	Composite
Password	A user's hashed (encrypted) password	16 byte array		All 0s	no	no	Single	Simple

Order

The Order entity holds information about orders that have been taken. The only possible candidate key is the OrderNumber attribute and thus the OrderNumber attribute is the primary key and the Order entity is a strong entity.

Name of Attribute	*Description*	*Domain/ Type*	*Value-Range*	*Default Value*	*NULL?*	*Unique?*	*Single or multi-value*	*Simple or Composite*
OrderNumber	This identifies the order	int	> 0	0	no	yes	Single	Simple
OrderType	This is the type of order	string	Walk-in, To-go, or Phone Order	Walk-in	no	no	Single	Simple
Discount	Any coupon or discount that has been applied to the order	Decimal	>= 0.00	0	no	no	Single	Simple
Comments	Any special instructions or considerations that should be known	Text		Empty String	no	no	Single	Simple
Status	Whether or not the order has been paid. Phone Orders are taken and put into the database but are paid when the order is picked up	String	Paid or Unpaid	Unpaid	no	no	Single	Simple

Customer

The Customer entity holds basic information about customers when phone orders are placed, whether it is a pick-up or delivery. For a pick-up customers are identified by their name and phone number. For a delivery the address is obviously needed to get the order to the customer. Although no attributes are necessarily unique, the combination of a customer's name and phone number are unique and thus those two attributes form the primary key and the Customer entity is a strong entity.

Name of Attribute	*Description*	*Domain/ Type*	*Value-Range*	*Default Value*	*NULL?*	*Unique?*	*Single or multi-value*	*Simple or Composite*
Name	Customer's name	string		Empty string	no	no	Single	Composite
Phone Number	Customer's phone number	int	> 0, 7 or 10 digits	1111111	no	no	Single	Simple
Address	Customer's address	string		Empty string	no	no	Single	Composite

Side Item

The Side Item entity is the basis for all the items that can be associated with an order. This is a super class to all other items. The ID attribute is the primary key so this, and all subclasses, is a strong entity.

Name of Attribute	*Description*	*Domain/ Type*	*Value-Range*	*Default Value*	*NULL?*	*Unique?*	*Single or multi-value*	*Simple or Composite*
ID	Unique identifier for item	int	> 0	0	no	yes	single	Simple
Size	The size of the item. This is used to calculate the total	string		Empty string	no	no	single	Simple
Total	The cost of the item	decimal	>= 0.00	0	no	no	single	Simple

Comments	Any special instructions or considerations	string		Empty string	no	no	single	Simple
Number	Number of the item to be ordered. This is to save space in the database	int	> 0	1	no	no	single	Simple

Food

The Food entity is a subclass of the Side Item entity and a generalization of the Wedge, Wing, and Pizza entities. It is a strong entity because it inherits the ID attribute from Side Item.

Name of Attribute	*Description*	*Domain/ Type*	*Value-Range*	*Default Value*	*NULL?*	*Unique?*	*Single or multi-value*	*Simple or Composite*
HowCooked	Describes how the food item should be cooked	string		normal	no	no	single	Simple

Salad

The Salad entity represents a salad from an order. There are several different types of salads, each with its own price.

Name of Attribute	*Description*	*Domain/ Type*	*Value-Range*	*Default Value*	*NULL?*	*Unique?*	*Single or multi-value*	*Simple or Composite*
Type	The type of salad order	string		Empty string	no	no	single	Composite

Salad Type

The Salad Type entity represents the different kinds of salads that can be ordered. The Salad Type entity is a strong entity because the Name attribute is the primary key.

Name of Attribute	*Description*	*Domain/ Type*	*Value-Range*	*Default Value*	*NULL?*	*Unique?*	*Single or multi-value*	*Simple or Composite*
Name	The name of the salad type	string		Empty string	no	yes	single	Simple
Price	How much the salad type costs	decimal	>= 0.00	0	no	no	single	simple

SaladIngredients

The SaladIngredients entity represents different items that can be put in a Salad Type. The Name Attribute is the primary key so the SaladIngredients entity is a strong entity.

Name of Attribute	*Description*	*Domain/ Type*	*Value-Range*	*Default Value*	*NULL?*	*Unique?*	*Single or multi-value*	*Simple or Composite*
Name	Name of the salad ingredient	string		Empty string	no	no	Single	Composite

Beer

The Beer entity represents a beer item on an order.

Name of Attribute	*Description*	*Domain/ Type*	*Value-Range*	*Default Value*	*NULL?*	*Unique?*	*Single or multi-value*	*Simple or Composite*
IsImport?	Represents whether the beer is an import. This is	Bool	True or false	FALSE	no	no	single	Simple

	used to determine the price							

Soda

The Soda entity represents a soda item on an order. The Soda entity is a subclass but adds no attributes.

Wedge

The Wedge entity represents an order of potato wedges.

Name of Attribute	*Description*	*Domain/ Type*	*Value-Range*	*Default Value*	*NULL?*	*Unique?*	*Single or multi-value*	*Simple or Composite*
Sauce	Indicates what kind of dipping sauce the customer wants, if any.	string	Honey Mustard, 1000 Island, None	None	no	no	single	simple

Wing

The Wing entity represents an order of hot wings.

Name of Attribute	*Description*	*Domain/ Type*	*Value-Range*	*Default Value*	*NULL?*	*Unique?*	*Single or multi-value*	*Simple or Composite*
AddHotSauce?	Indicates whether or not the customer wants hot	Bool	True or false	FALSE	no	no	single	simple

	sauce added							

Prices

The Prices entity represents the price of an item and the date range for which the price is good for. The Name attribute is the primary key so the Prices entity is a strong entity.

Name of Attribute	*Description*	*Domain/ Type*	*Value-Range*	*Default Value*	*NULL?*	*Unique?*	*Single or multi-value*	*Simple or Composite*
Name	Name of the item	string		Empty String	no	yes	Single	simple
Amount	Cost of the item	decimal	>= 0.00	0	no	no	single	simple
StartDate	When the price starts	Date		Today	no	no	single	simple
EndDate	When the price ends	Date		NULL	yes	no	single	simple

Pizza

The Pizza entity represents a pizza on an order.

Name of Attribute	*Description*	*Domain/ Type*	*Value-Range*	*Default Value*	*NULL?*	*Unique?*	*Single or multi-value*	*Simple or Composite*
Side1Cheese	How much cheese on side 1	String	None, Light, Normal, Extra	Normal	no	no	Single	Simple
Side2Cheese	How much cheese on side 2	String	None, Light, Normal, Extra	Normal	no	no	Single	Simple
Side1Sauce	How much sauce on side 1	String	None, Light, Normal, Extra	Normal	no	no	Single	Simple
Side2Sauce	How much sauce on side 2	String	None, Light, Normal, Extra	Normal	no	no	Single	Simple

Specialty

The Specialty entity represents a specialty pizza type (combination, supreme, meat lovers, etc...). The Name attribute is the primary key so the Specialty entity is a strong entity.

Name of Attribute	*Description*	*Domain/ Type*	*Value-Range*	*Default Value*	*NULL?*	*Unique?*	*Single or multi-value*	*Simple or Composite*
Name	Name of the specialty type	string		Empty string	no	yes	single	simple

Topping

The topping entity represents a pizza topping. The Name attribute is the primary key so the Topping entity is a strong entity. This entity is used both to determine what toppings are in a specialty and what individual toppings go on a pizza.

Name of Attribute	*Description*	*Domain/ Type*	*Value-Range*	*Default Value*	*NULL?*	*Unique?*	*Single or multi-value*	*Simple or Composite*
Name	Name of the pizza topping	String		Empty string	no	yes	single	simple

HistoricalData

The HistoricalData entity holds totals for different categories from previous days. The Date attribute is the primary key so the HistoricalData entity is a strong entity. This entity will primarily be used for reports.

Name of Attribute	*Description*	*Domain/ Type*	*Value-Range*	*Default Value*	*NULL?*	*Unique?*	*Single or multi-value*	*Simple or Composite*

Date	Day for which the totals are calculated	Date		Yesterday	no	yes	single	simple
WedgeTotal	Total sales from wedges	Decimal	>= 0.00	0	no	no	single	simple
WingsTotal	Total sales from wings	Decimal	>= 0.00	0	no	no	single	simple
SodaTotal	Total sales from sodas	Decimal	>= 0.00	0	no	no	single	simple
BeerTotal	Total sales from beer	Decimal	>= 0.00	0	no	no	single	simple
PizzaTotal	Total sales from pizza	Decimal	>= 0.00	0	no	no	single	simple
SaladTotal	Total sales from salads	Decimal	>= 0.00	0	no	no	single	simple
SalesTax	Total sales tax	Decimal	>= 0.00	0	no	no	single	simple
DiscountTotal	Total of all discounts	Decimal	>= 0.00	0	no	no	single	simple

2.2 *Relationships*

Name	Order to Employee
Description	This relationship connects each employee to orders that they took. This allows stats to be kept of the performance of individuals to be tracked. And if there is a problem with an order, management can easily look up who took the order.
Entities Involved	Order and Employee
Mapping Cardinality	* to 1
Participation Constraint	total

Name	Customer to Order

Description	If an order is a phone order then it is associated with a customer.
Entities Involved	Customer and Order
Mapping Cardinality	1 to *
Participation Constraint	Total

Name	Side Item to Order
Description	When a side item is ordered it must be associated with an order, otherwise it would be unknown who the item belongs to.
Entities Involved	SideItem and Order
Mapping Cardinality	* to 1
Participation Constraint	Total

Name	Salad to SaladType
Description	When a salad is ordered it must be associated with a salad type so employees know which items to put on the salad.
Entities Involved	Salad and SaladType
Mapping Cardinality	1 to *
Participation Constraint	total

Name	SaladIngredients to SaladType
Description	Associates ingredients with a salad type so that employees will know what to put on each salad

Entities Involved	SaladIngredients and SaladType
Mapping Cardinality	* to *
Participation Constraint	optional

Name	Pizza to Specialty
Description	If a customer orders a pizza with a specialty (meaning they want a combination pizza or a meat lovers pizza) then the pizza will be associated with that specialty
Entities Involved	Pizza and Specialty
Mapping Cardinality	* to *
Participation Constraint	optional

Name	Pizza to Topping
Description	If a customer orders a pizza but just orders by toppings (a large pepperoni and sausage for example) then the pizza will be associated with each of those toppings
Entities Involved	Pizza and Topping
Mapping Cardinality	* to *
Participation Constraint	optional

Name	Topping to Specialty
Description	When a specialty pizza type is created it must be associated with different toppings so employees will know what goes on that specialty

Entities Involved	Topping and Specialty
Mapping Cardinality	* to *
Participation Constraint	optional

2.3 *Related Entities*

Side Item

This entity is an instance of specialization. All side items have basic attributes and then their own unique attributes. This entity contains those attributes that all side items use.

Food

This entity is an instance of generalization. Food items that are cooked need an attribute for how they are to be cooked. That attribute was pulled out of the Wedge, Wing, and Pizza entities into this one.

3 ER-Model vs. Relational Model

3.1 *Description*

In 1976, Peter Chen derived the Entity-Relationship (ER) model, a high-level data model that is useful in developing a conceptual design for a database. At the time other models existed but Chen's ER model's was appealing and accepted due to its conceptual simplicity, visual representation, effective communication, and integration with the relational database model. The following are key elements of the ER model:

- Entities: A thing in the real world with an independent existence (Elmasri/Navathe, 2007, p. 61).
- Attributes: Each entity has attributes-the particular properties that describe it (Elmasri/Navathe, 2007, p. 62).

- Key attribute: distinct values in each entity that can be used to identify each entity uniquely (Elmasri/Navathe, 2007, p. 66).
- Relationships: Exist between two entities that are related to each other.

3.2 *Comparison*

While Chen's ER model is used to create an accurate reflection of the real world in a database, Ted Codd of IBM research introduced the relational model that is used to show how this data will be represented in a Relational Database Management System (RDBMS). The relational model represents the database as a collective of relationships, consists of tables with rows that define relationships between a set of values, and uses relational algebra to relations (Elmasri/Navathe, 2007,p. 46).

3.3 *Conversion from E-R model to relational model*

The ER Model is an important preliminary stage of conceptual design use to communicate between users and the DBA. We convert from ER to Relational Model because it is the logical level of database design.

Relational Model Concepts

- A row of table is a relational instance/tuple
- A column of table is an attribute
- A table is the schema/relation
- Cardinality is the number of rows
- Degree is the number of columns

We can convert the ER Model to the Relational Model using the following principal idea:

- create a table for each entity set
- create a table for each relationship
- using columns for each attributes

- indivisibility rules and ordering rules
- primary key

First step would be to create a relation for all strong entity types with columns to represent each attributes. One of the attributes will be selected as the Primary key (composite and foreign keys can also be implemented). Weak Entities must include a column on the right side of the table with the primary key of the Strong Enitity Set. For composite attributes the Relational Model Indivisibility Rule applies: one column for each component attribute; no column for the composite attribute itself. For multi-valued attributes, take the attribute and turn it into a new entity of its own. Then make a 1:M relationship between the new entity and the existing one. Then convert as normal. The Primary Key of the Weak Entity Set should include 'Discriminator + Foreign Key'.

For Unary/Binary Relationships there are two approaches. For a 1:1 relationship with out total participation we build a table with two columns. One column for each participating entity set's primary key and we add successive columns, one for each descriptive attributes of the relationship set if any exists. The other approach is for a 1:1 relationship with total participation. We add an extra column and insert the primary key of the entity set with out complete participation to the relationship. The issue with an N-ary relationships, a single relationship including three or more entities, is that they can usually be better represented by using an additional entity and a set of binary relationships.

3.4 *Constraints*

An entity constrain will require primary keys not to be null. Another constraint for the primary key is that values must be unique. Constrains to a foreign key are enforce through a referential constraint which is any references to other existing tuples in other relations must be valid. The check constraint checks the values entered are valid according to the requirements of the attribute.

4 Relational Model

4.1 *Relations*

Employee

Attribute Name	EmployeePK	Title	Name	Password
Domain	int	string	string	Binary data
Constraints	Primary Key			

Order

Attribute Name	OrderPK	EmployeeFK	CustomerFK	OrderType	Discount	Comments	Status
Domain	int	int	int	string	double	string	string
Constraints	Primary Key	referential	referential				

Customer

Attribute Name	CustomerPK	Name	PhoneNumber	Address
Domain	int	string	int	string
Constraints	Primary Key			

Salad

Attribute Name	SaladPK	OrderFK	SaladTypeFK	Size	Total	Comments	Quantity
Domain	int	int	int	string	double	string	int
Constraints	Primary Key	referential	referential				

Salad Type

Attribute Name	SaladTypePK	Name	Price
Domain	int	string	double
Constraints	Primary Key		

Salad Ingredients

Attribute Name	SaladIngredientsPK	Name
Domain	int	
Constraints	Primary Key	

Soda

Attribute Name	SodaPK	OrderFK	Size	Total	Comments	Quantity
Domain	int	int	string	double	string	int
Constraints	Primary	referential				

Beer

Attribute Name	BeerPK	OrderFK	IsImport?	Size	Total	Comments	Quantity
Domain	int	int	bool	string	double	string	int
Constraints	Primary Key	referential					

Wedge

Attribute Name	WedgePK	OrderFK	Size	Total	Comments	Quantity	HowCooke d	Sauce
Domain	int	int	string	double	string	int	string	string
Constraints	Primary Key	referential						

Wing

Attribute Name	WingPK	OrderPK	Size	Total	Comments	Quantity	HowCooked	AddHotSauce?
Domain	int	int	string	double	string	int	string	bool
Constraints	Primary Key	referential						

Pizza

Attribute Name	PizzaPK	OrderFK	Size	Total	Comments	Quantity	HowCooked	Side1 Cheese	Side2 Cheese	Side1 Sauce	Side2 Sauce
Domain	int	int	string	double	string	int	string				
Constraints	Primary Key	referential									

Specialty

Attribute Name	SpecialtyPK	Name
Domain	int	string
Constraints	Primary Key	

Topping

Attribute Name	ToppingPK	Name
Domain	int	string
Constraints	Primary Key	

Prices

Attribute Name	PricePK	Name	Amount	StartDate	EndDate

Domain	int	string	double	DateTime	DateTime
Constraints	Primary Key				

Historic Data

Attribute Name	HSPK	Date	Wedge Total	WingTotal	SodaTotal	BeerTotal	PizzaTotal	Salad Total	SalesTax	Discounts Total
Domain	int	DateTime	double	double	double	double	double	double	double	double
Constraints	Primary Key	Unique (Candidate Key)								

SaladTypeToIngredients

Attribute Name	STTIPK	SaladTypeFK	SaladIngredientsFK	Amount
Domain	int	int	int	double
Constraints	Primary Key	referential	referential	

PizzaToSpecialty

Attribute Name	PTSPK	PizzaFK	SpecialtyFK
Domain	int	int	int
Constraints	Primary Key	referential	referential

PizzaToTopping

Attribute Name	PTTPK	PizzaFK	ToppingFK	Amount
Domain	int	int	int	double
Constraints	Primary Key	referential	referential	

PizzaToppingToSpecialty

Attribute Name	PTTSPK	SpecialtyFK	ToppingFK	Amount
Domain	int	int	int	double
Constraints	Primary Key	referential	referential	

4.2 *Sample Data*

Employee

EmployeePK	Title	Name	Password
1	CEO	Chris	<Binary Data>
2	General Manager	Ruben	<Binary Data>
3	Store Manager	Eric	<Binary Data>
4	Assistant Manager	Bob	<Binary Data>
5	Shift Leader	Henry	<Binary Data>
6	Clerk	Joe	<Binary Data>
7	Driver	Randy	<Binary Data>
8	Clerk	Jimmy	<Binary Data>
9	Clerk	Eddie	<Binary Data>

Order

OrderPK	Employee	Customer	OrderTyp	Discount	Comments	Status

	FK	FK	e			
1	9		Dine-In	0		paid
2	9		Dine-In	0		paid
3	8		Dine-In	0		paid
4	8		Take-Out	0		paid
5	7		Delivery	5	$5.00 off coupon	unpaid
6	5	2	Take-Out	0		paid
7	6		Dine-In	0		paid
8	9		Dine-In	2.5	Free pitcher sode coupon	paid
9	4		Take-Out	7.5		paid

Customer

CustomerPK	**Name**	**PhoneNumber**	**Address**
1	Henry Johnson		
2	Jenny	8675309	
3			
4			
5			
6			
7			
8			
9			

Salad

SaladPK	**OrderFK**	**SaladType FK**	**Size**	**Total**	**Comments**	**Quantity**
1	1	1	side	5		3
2	2	1	side	5		1

3	5	4	chef	7		1
4	9	3	side	4.5		1
5	9	2	chef	7.75	Extra dressing	2

SaladType

SaladTypePK	Name	Price
1	Chicken - side	5
2	Chicken - chef	7.75
3	Caesar - side	4.5
4	Caesar - chef	7

SaladIngredients

SaladIngredientsPK	SaladTypeFK	Name
1	1	Chicken
2	2	Chicken
3	3	
4	4	
5	2	Cheddar Cheese

Soda

SodaPK	OrderFK	Size	Total	Comments	Quantity
1	1	small	1		1
2	2	small	1		4
3	3	large	1.5		2
4	3	small	1		1
5	3	large	1.5		1
6	4	pitcher	3.75		2
7	5	small	1		3
8	6	pitcher	3.75	Extra ice	1

9	7	pitcher	3.75		1
10	8	pitcher	3.75		1
11	8	large	1.5	No ice	2
12	8	pitcher	3.75		1
13	9	pitcher	3.75		1
14	9	small	1		2

Beer

BeerPK	**OrderFK**	**Size**	**Total**	**Comments**	**Quantity**	**IsImport?**
1	1	Mug	2		1	FALSE
2	2	Cup	3.5		4	TRUE
3	3	Pitcher	8		2	FALSE
4	3	Cup	3		1	FALSE
5	3	Pitcher	8		2	FALSE
6	6	Mug	2		2	FALSE
7	7	Pitcher	9		1	TRUE
8	8	Cup	3		2	FALSE

Wing

WingPK	**OrderFK**	**Size**	**Total**	**Comments**	**Quantity**	**AddHotSauce?**	**HowCooked**
1	2	10pc	6		1	TRUE	normal
2	2	10pc	6		2	TRUE	well-done
3	3	15pc	9		1	TRUE	normal
4	3	15pc	9		1	FALSE	normal
5	3	10pc	6		1	TRUE	normal
6	4	10pc	6		3	FALSE	normal

7	5	15pc	9		4	TRUE	normal
8	7	10pc	6		2	FALSE	well-done

Wedge

WedgePK	OrderFK	Size	Total	Comments	Quantity	Sauce	HowCooked
1	1	small	3		1	none	normal
2	2	large	5	Extra sauce	1	Honey mustard	well-done
3	2	large	5		2	ranch	normal
4	3	small	3		2	Honey mustard	well-done
5	4	large	5	Extra sauce	1	ranch	normal
6	6	large	5		1	Thousand island	normal
7	7	small	3		3	Honey mustard	normal
8	9	large	5		2	ranch	well-done

SaladTypeToIngredients

STTIPK	SaladTypeFK	SaladIngredientsFK	Amount
1			
2			
3			
4			

5			
6			

Pizza

PizzaPK	OrderFK	Size	Total	Comments	Quantity	HowCooked	Side1Cheese	Side2Cheese	Side1Sauce	Side2Sauce
1	1	small			1	normal	normal	normal	normal	normal
2	1	large			1	normal	normal	normal	none	normal
3	1	large			2	normal	normal	normal	light	normal
4	2	Extra large			1	normal	normal	extra	normal	normal
5	2	ind			1	normal	normal	normal	normal	normal
6	3	small			3	light	normal	normal	extra	extra
7	3	large			2	normal	extra	normal	normal	normal
8	3	Extra large			3	normal	normal	light	normal	normal
9	3	small			3	normal	normal	normal	normal	normal
10	4	large			1	normal	normal	normal	normal	normal
11	5	large			2	normal	normal	normal	extra	extra
12	6	large			3	normal	normal	normal	normal	normal
13	6	Extra large			1	well-done	normal	normal	normal	normal
14	6	large			2	normal	normal	normal	normal	normal
15	7	Extra large			2	normal	normal	normal	normal	normal
16	7	large			2	normal	normal	normal	normal	normal
17	7	Extra			3	normal	light	light	normal	normal

		large								
18	8	large			1	normal	normal	normal	normal	normal
19	9	Extra large			1	well-done	normal	none	normal	normal

Topping

ToppingPK	**Name**
1	Pepperoni
2	Sausage
3	Beef
4	Onion
5	Green Pepper
6	Black Olive
7	Jalapeno
8	Pineapple
9	Ham
10	Chicken
11	Mushroom
12	Bacon

HistoricData

HSPK	**Date**	**Wedge Total**	**Wing Total**	**Soda Total**	**Beer Total**	**Pizza Total**	**Salad Total**	**Sales Tax**	**Discounts Total**
1	10/19/11								
2	10/20/11								
3	10/21/11								
4	10/22/11								

Prices

PricePK	Name	Amount	StartDate	EndDate
1	soda-small	1	10/23/11	
2	soda-large	1.5	10/23/11	
3	soda-pitcher	3.75	10/23/11	11/12/11
4	beer-mug	2	10/23/11	
5	beer-cup	3	10/23/11	
6	beer-pitcher	8	10/23/11	
7	import-mug	2.5	10/23/11	
8	import-cup	3.5	10/23/11	
9	import-pitcher	9	10/23/11	
10	wedge-small	3	10/23/11	
11	wedge-large	5	10/23/11	
12	wings-10pc	5	10/23/11	
13	wings-15pc	9	10/23/11	
14	ind-1top	2.5	10/23/11	
15	ind-add	0.4	10/23/11	
16	small-1top	5	10/23/11	
17	small-add	0.5	10/23/11	
18	large-1top	10	10/23/11	
19	large-add	0.65	10/23/11	
20	xlarge-1top	13	10/23/11	
21	xlarge-add	0.75	10/23/11	
22	xlarge-1top	12	09/27/11	10/22/11

PizzaToSpecialty

PTSPK	PizzaFK	SpecialtyFK
1	1	1
2	2	3
3	5	4
4	6	2
5	12	1

Specialty

SpecialtyPK	Name
1	Combination
2	Meat-Lovers
3	Vegetarian
4	Bacon Cheddar Burger

PizzaToTopping

PTTPK	**PizzaFK**	**ToppingFK**	**Amount**
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			

PizzaToppingToSpecialty

PTTSPK	**ToppingFK**	**SpecialtyFK**	**Amount**
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
12			
13			
14			
15			
16			
17			
18			
19			
20			

5 Queries

1. List all orders with a pepperoni pizza.

$\Pi_{\text{order.*}}(\sigma_{\text{Topping.Name like 'Pepperoni'}}(((\text{Order} \bowtie_{\text{OrderPK=OrderFK}} \text{Pizza}) \bowtie_{\text{PizzaPK=PizzaFK}} \text{PizzaToTopping}) \bowtie_{\text{ToppingPK=ToppingFK}} \text{Topping}))$

{o | Order(o) ^(∃ p)(Pizza(p) ^ o.OrderPK=p.OrderFK ^(∃ ptt)(PizzaToTopping(ptt) ^ ptt.PizzaFK=p.PizzaPK ^ (∃ t)(Topping(t) ^ t.ToppingPK=ptt.ToppingFK ^ t.Name = "Pepperoni")))}

{<o, ot, d, c, s> | Order(o, ot, d, c, s) ^(∃ p)(∃ ptt)(∃ t)(Pizza(o,p,_,_,_,_,_,_,_,_) ^ PizzaToTopping(ptt,p,t,_) ^ Topping(t,ptt,"Pepperoni"))}

2. List all customers that order only pepperoni pizzas.

3. List customers who have only placed one order.

4. List employees who have placed no more than one order with the same customer.

5. List the customers who have ordered the most expensive pizza.

6. List employees that have taken the least expensive order.

7. List the order with the greatest discount.

8. List orders with more than one pizza with total greater than $20.

9. List employees that have taken an order for every customer.

10. List the item with the highest price between 10-12-11 and 10-22-11.

6 Normalization

Data Normalization is a a set of rules and techniques used to identify relationships among attributes, combine attributes to from relations, and combining relations to form a database. The purpose behind data normalization is to eliminate redundant data storage, construct data so that model is flexible, and close modeling of real world entities, processes, and their relationships. It is the best way to efficiently organize data in a database.

Definitions of Normal Forms:

- **First Normal Form:** A table is in first normal form if all the key attributes have been defined and repeating groups exist
- **Second Normal Form:** If a table is in First Normal Form and every non key attribute is fully functionally dependent, there are no partial dependencies, on the whole of the primary key.
- **Third Normal Form:** A table is in Thirds Normal Form if it is in Second Normal Form and every non key attribute has no transitive dependencies on the primary key.
- **Boyce-Codd Normal Form:** If and only if a table is in Third Normal Form and every determinant is a candidate key.

By Normalization the data, problems like duplication of data in several places in the database can be avoided and the risk of updates in one place but not the other will be eliminate. It is important to have data integrity since the information will live forever in a database and historical bad data can be hard to eliminate.

7 Oracle/SQL*Plus

The following is a list of Oracle/SQL*PLUS instructions that we used to create, load and query our database. We used the following commands used or practiced in this phase:

1. CREATE TABLE table_name ...;
2. CREATE VIEW view_name ...;
3. CREATE INDEX idx_name ...;
4. INSERT INTO ...;
5. DROP TABLE ... PURGE;
6. DROP VIEW ... ;
7. COMMIT;
8. ROLLBACK;
9. SELECT
10. DESC
11. System tables such as user_objects, user_indexes, user_tables, tab, col,
12. CREATE or REPLACE FUNCTION ...
13. CREATE or REPLACE PROCEDURE ..
14. CREATE or REPLACE TRIGGER ...
15. DROP PROCEDURE | FUNCTION ...
16. We Ran the following SQl statements to test our database:

    select * from tab;

    seelct * from user_objects;

    select * from user_constraints;
17. To remove strange tables found after running "select * rom tab" we ran command "purge recyclebin" in sqlplus to get rid of them.

8 Relational Database Schema

Beer {BeerPK, SideItemFK, IsImport}

Cheese {CheesePK, Portion}

Customer {CustomerPK, Name, PhoneNumber, StreetAddress, City, Zip}

Employee {EmployeePK, TitleFK, Name, Password}

EmployeeTitle {TitlePK, Name}

Food {FoodPK, SideItemFK, HowCooked}

HistoricData {HistoricDataPK, Day, WedgeTotal, WingTotal, SodaTotal, BeerToatl, PizzaTotal, SaladTotal, SalesTax, DiscountsTotal}

Order {OrderPK, CustomerFK, OrderTypeFK, OrderStatusFK, OrderNumber, Discount, Comments}

OrderStatus {OrderStatusPK, Name}

OrderType {OrderTypePK, Name}

PizzaCheese {PizzaCheesePK, PizzaFK, CheeseFK, IsOnSide1, IsOnSide2}

PizzaSauce {PizzaSaucePK, PizzaFK, SauceFK, IsOnSide1, IsOnSide2}

Pizza {PizzaPK, FoodFK}

PizzaTopping {PizzaToppingPK, PizzaFK, ToppingFK, Amount, IsOnSide1, IsOnSide2}

Price {PricePK, Name, Amount, StartDate, EndDate}

SaladIngredient {SaladIngredientPK, Name}

Salad {SaladPK, SideItemFK}

SaladTypeIngredient {SaladIngredientPK, SaladTypeFK, SaladFK, Amount}

SaladType {SaladTypePK, Name}

Sauce {SaucePK, Portion}

SideItem {SideItemPK, OrderFK, PriceFK, Comments, Quantity}

Soda {SodaPK, SideItemFK}

SpecialtyPizza {SpecialtyPizzaPK, PizzaFK, SpecialtyFK, IsOnSide1, IsOnSide2}

Specialty {SpecialtyPK, Name}

ToppingSpecialty {ToppingSpecialtyPK, ToppingFK, SpecialtyFK, Amount}

Topping {ToppingPK, Name}

Wedge {WedgePK, FoodFK, Sauce}

Wing {WingPK, FoodFK, AddHotSauce}

9 SQL Queries

1. List all orders with a pepperoni pizza.

```
SELECT o.*
FROM Order o
	INNER JOIN SideItem si ON (si.OrderFK = o.OrderPK)
	INNER JOIN Food f ON (f.SideItemFK = si.SideItemPK)
	INNER JOIN Pizza p ON (f.FoodPK = p.FoodFK)
	INNER JOIN PizzaTopping pt ON (pt.PizzaFK = p.PizzaPK)
	INNER JOIN Topping t ON (ppt.ToppingFK = t.ToppingPK)
WHERE t.Name LIKE “Pepporoni”
	AND NOT EXISTS
		(
			SELECT t2.*
			FROM Topping t2
				INNER JOIN PizzaTopping pt2 ON (pt2.ToppingFK = t.ToppingPK)
```

```
            INNER JOIN Pizza p2 ON (ppt2.PizzaFK on pt.PizzaPK)
        WHERE p2.OrderFK = o.OrderPK
        AND t2.Name NOT LIKE "Pepperoni"
    )
```

2. List all customers that order only pepperoni pizzas.

```
SELECT c.*
FROM Customer c
    INNER JOIN Order o ON (o.CustomerFK = c.CustomerPK)
    INNER JOIN SideItem si ON (si.OrderFK = o.OrderPK)
    INNER JOIN Food f ON (f.SideItemFK = si.SideItemPK)
    INNER JOIN Pizza p ON (f.FoodPK = p.FoodFK)
    INNER JOIN PizzaToTopping ppt ON (ppt.PizzaFK = p.PizzaPK)
    INNER JOIN Topping t ON (ppt.ToppingFK = t.ToppingPK)
WHERE t.Name LIKE "Pepporoni"
    AND NOT EXISTS
    (
        SELECT t2.*
        FROM Topping t2
            INNER JOIN PizzaToTopping ppt2 ON (ppt2.ToppingFK = t.ToppingPK)
            INNER JOIN Pizza p2 ON (ppt2.PizzaFK on pt.PizzaPK)
        WHERE p2.OrderFK = o.OrderPK
        AND t2.Name NOT LIKE "Pepperoni"
    )
```

3. List customers who have only placed one order.

```
SELECT c.*
```

```
FROM Customer c
	INNER JOIN Order o ON (o.CustomerFK = c.CustomerPK)
WHERE NOT EXISTS
	(
		SELECT o2.*
		FROM Order o2
		WHERE o2.Customer.FK = c.CustomerPK
		AND o2.OrderPK != o.OrderPK
	)
```

4. List employees who have placed no more than one order with the same customer.

```
SELECT e.*
FROM Employees e
	INNER JOIN Order o ON (e.EmployeePK = o.EmployeeFK)
	INNER JOIN Customer c ON (o.OrderPK = c.OrderFK)
WHERE NOT EXISTS
	(
		SELECT *
		FROM Employee e2
			INNER JOIN Order o2 ON (e2.EmployeePK = o2.EmployeeFK)
			INNER JOIN Customer c2 ON (o2.OrderPK = c2.OrderFK)
		WHERE e.EmployeePK = e2.EmployeePK
			AND c.CustomerPK = c2.CustomerPK
			AND o.OrderPK != o2.OrderPK
	)
```

5. List the customers who have ordered the most expensive pizza.

```
SELECT c.*
FROM Customer c
	INNER JOIN Order o ON (o.CustomerFK = c.CustomerPK)
	INNER JOIN SideItem si ON (si.OrderFK = o.OrderPK)
	INNER JOIN Food f ON (f.SideItemFK = si.SideItemPK)
	INNER JOIN Pizza p ON (f.FoodPK = p.FoodFK)
	INNER JOIN Prices pr ON (pr.PricePK = p.PriceFK)
GROUP BY c.CustomerPK
HAVING pr.Amount = MAX(pr.Amount)
```

6. List Employees that have taken the least expensive order.

```
SELECT e.*
FROM Employee e
	INNER JOIN Order o ON (e.EmployeePK = o.EmployeeFK)
	INNER JOIN SideItem si ON (si.OrderFK = o.OrderPK)
	INNER JOIN Price pr ON (pr.PricePK = si.PriceFK)
GROUP BY e.EmployeePK
HAVING SUM(pr.Amount) = MIN(SUM(pr.Amount))
```

7. List the order with the greatest discount.

```
SELECT o.*
FROM Order o
GROUP BY o.OrderPK
HAVING o.Discount = MAX(o.Discount)
```

8. List orders with more than one pizza with a total greater than $20.

```
SELECT o.*
FROM Order o
```

```
    INNER JOIN SideItem si ON (si.OrderFK = o.OrderPK)
    INNER JOIN Food f ON (f.SideItemFK = si.SideItemPK)
    INNER JOIN Pizza p ON (f.FoodPK = p.FoodFK)
    INNER JOIN Prices pr ON (p.PriceFK = pr.PricePK)
WHERE  pr.Amount > 20
GROUP BY o.OrderPK
HAVING  COUNT(*) > 2
```

9. List employees that have taken an order for every customer.

```
SELECT e.*
FROM Employee e
    INNER JOIN Order o ON (e.EmployeePK = o.EmployeeFK)
    INNER JOIN Customer c ON (o.CustomerFK = c.CustomerPK)
GROUP BY e.EmployeePK, c.CustomerPK
HAVING COUNT(*) = COUNT(SELECT * FROM Customer)
```

10. List the item with the highest price between 10-12-11 and 10-22-11.

```
SELECT pr.*
FROM Prices pr
WHERE pr.StateDate > 10-12-11
    AND pr.EndDate < 10-22-11
GROUP BY pr.PricePK
HAVING pr.Amount = MAX(pr.Amount)
```

11. List how many orders each employee has taken for each customer

```
SELECT e.EmployeePK AS Employee_ID, e.Name AS Employee_Name, c.Name AS
    Customer_Name, COUNT(*) AS Number_Of_Orders
FROM Employee e
```

```
        INNER JOIN Order o ON (e.EmployeePK = o.OrderFK)
        INNER JOIN Customer c ON (o.CustomerFK = c.CustomerPK)
GROUP BY e.EmployeePK, c.CustomerPK
```

10 Common Features of PL/SQL and T-SQL

There are common features in Oracle PL/SQL and MS Trans-SQL for example both support features such as constraints, functions, cursors, stored procedures, triggers, and packages. However the syntax is not the same.

Stored subprograms or procedures can perform an action and/or compute value and can be restricted by user permissions. Example of actions by a subprogram would be insertion, deletion, or updating records in a database.

Benefits of a stored subprogram provide modularity, re-usability, and maintainability. Using stored subprograms will increase performance and turn repetitive task to be automated and scheduled.

11 PL/SQL

- A stored procedure are saved in the databases to improve performance and re-usability. The following is the syntax for creating a stored procedure:

```
DECLARE (Declarative section: variables, types, and local subprograms)
BEGIN (Executable section: procedural and SQL statements go here)
        (This is the only section of the block that is required)
EXCEPTION (Exception handling section: error handling statements go here)
END;
```

- A stored function returns a result. The following is the syntax use to create a stored function:

```
CREATE [OR REPLACE] FUNCTION function_name [ (parameter [,parameter]) ]
```

```
IS
  [declaration_section]
BEGIN
 executable_section
 return [return value]
 [EXCEPTION exception_section]
END [procedure_name];
```

- A package is stored functions and procedures that can be packaged into a larger unit, essentially a library of procedures and functions. The following is the syntax of creating a package:

```
CREATE [OR REPLACE] PACKAGE package_name

  [AUTHID {CURRENT_USER | DEFINER}]
  {IS | AS}
  [PRAGMA SERIALLY_REUSABLE;]
  [collection_type_definition ...]
  [record_type_definition ...]
  [subtype_definition ...]
  [collection_declaration ...]
  [constant_declaration ...]
  [exception_declaration ...]
  [object_declaration ...]
  [record_declaration ...]
  [variable_declaration ...]
  [cursor_spec ...]
  [function_spec ...]
```

(Package Syntax Contd.)

```
[procedure_spec ...]
[call_spec ...]
[PRAGMA RESTRICT_REFERENCES(assertions) ...]
END [package_name];

[CREATE [OR REPLACE] PACKAGE BODY package_name {IS | AS}
[PRAGMA SERIALLY_REUSABLE;]
  [collection_type_definition ...]
  [record_type_definition ...]
  [subtype_definition ...]
  [collection_declaration ...]
  [constant_declaration ...]
  [exception_declaration ...]
  [object_declaration ...]
  [record_declaration ...]
  [variable_declaration ...]
  [cursor_body ...]
  [function_spec ...]
  [procedure_spec ...]
```

```
  [call_spec ...]
[BEGIN
  sequence_of_statements]
END [package_name];]
```

- A trigger is fired when when a DML statement like Insert, Delete, or Updated are called.The following is the syntax used to create a trigger:

```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
BEGIN
 [sql statements]
END;
```

12 Sub Program

```
–  Trigger for updating and deleting values
CREATE OR REPLACE TRIGGER CPRC_PriceUpdateTrigger
BEFORE
UPDATE OR DELETE
ON CPRC_Price
FOR EACH ROW

BEGIN

   INSERT INTO CPRC_Log VALUES (:old.Name || :old.Amount,:new.Name || :new.Amount);

END;
/

–    Stored procedure for deleting rows based on the primary key
CREATE OR REPLACE PROCEDURE CPRC_DeleteFromPrice (pPK IN INTEGER) AS

BEGIN
```

```
   DELETE FROM CPRC_Price
   WHERE PricePK = pPK;

End;
/

–     Stored procedure for inserting a row
CREATE OR REPLACE PROCEDURE CPRC_InsertIntoPrice(
   pPK IN INTEGER,
   name IN VARCHAR2,
   amount IN FLOAT,
   sDate IN DATE,
   eDate IN DATE,
   groupName IN VARCHAR
   )
   AS

gPK INTEGER;

BEGIN

   SELECT g.GroupPK
   INTO gPK
   From CPRC_Group g
   WHERE g.Name LIKE groupName;

      INSERT INTO CPRC_Price VALUES(pPK, name, amount, sDate, eDate, gPK);
END;
/

–    function for returning the average of the top 'n' days
CREATE OR REPLACE FUNCTION CPRC_TopNDays(n IN INTEGER) RETURN FLOAT
AS

   averageDay FLOAT;
   i INTEGER;

   CURSOR c1 is
      SELECT h.Day, SUM(g.Total) AS "A"
      FROM CPRC_HistoricData h
         INNER JOIN CPRC_GroupData g on (h.HistoricDataPK = g.HistoricDataFK )
      GROUP BY h.Day;
BEGIN

   averageDay := 0.0;
   i := 0;

   OPEN c1;
```

```
    FOR hDay in c1
    LOOP
        IF i <= n
            THEN
                averageDay := averageDay + hDay.A;
        END IF;
        i := + 1;
    END LOOP;

    CLOSE c1;

    RETURN averageDay / n;
END;
```

13 General Description

The following is a list of operations for each group of users in our database project.

- **General Users:** Can only take customer orders, both phone or walk-in, and Open and Close tabs.
- **Managers:** Have elevated privileges that allows them to give discounts, view daily, monthly, and yearly sales/order reports. They can also take orders but can not change prices or anything else on the database.
- **Administrators:** Have full permissions to the databases to perform the same operations as the General and Manger users, including the ability to change prices, and add and delete new items to the database.

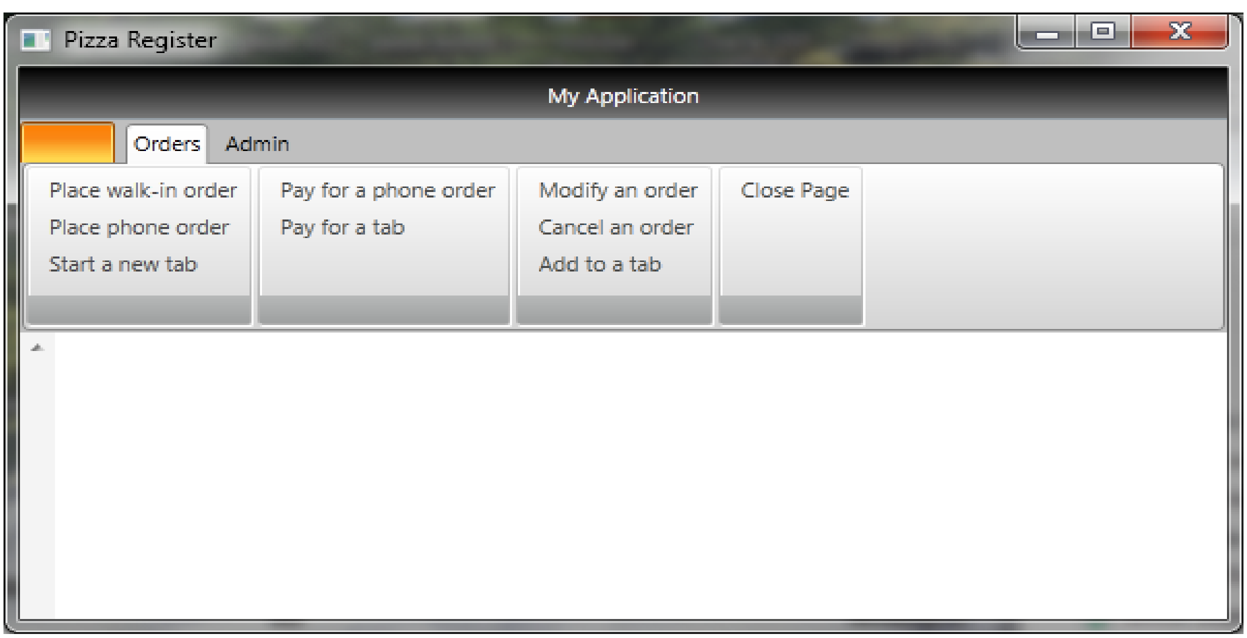
14 Group Activities

In the Pizza Parlor that we use for our fact-finding part of Phase 1, we found there to be three types of users. Below is a list of who their role is and the daily activities the perform.

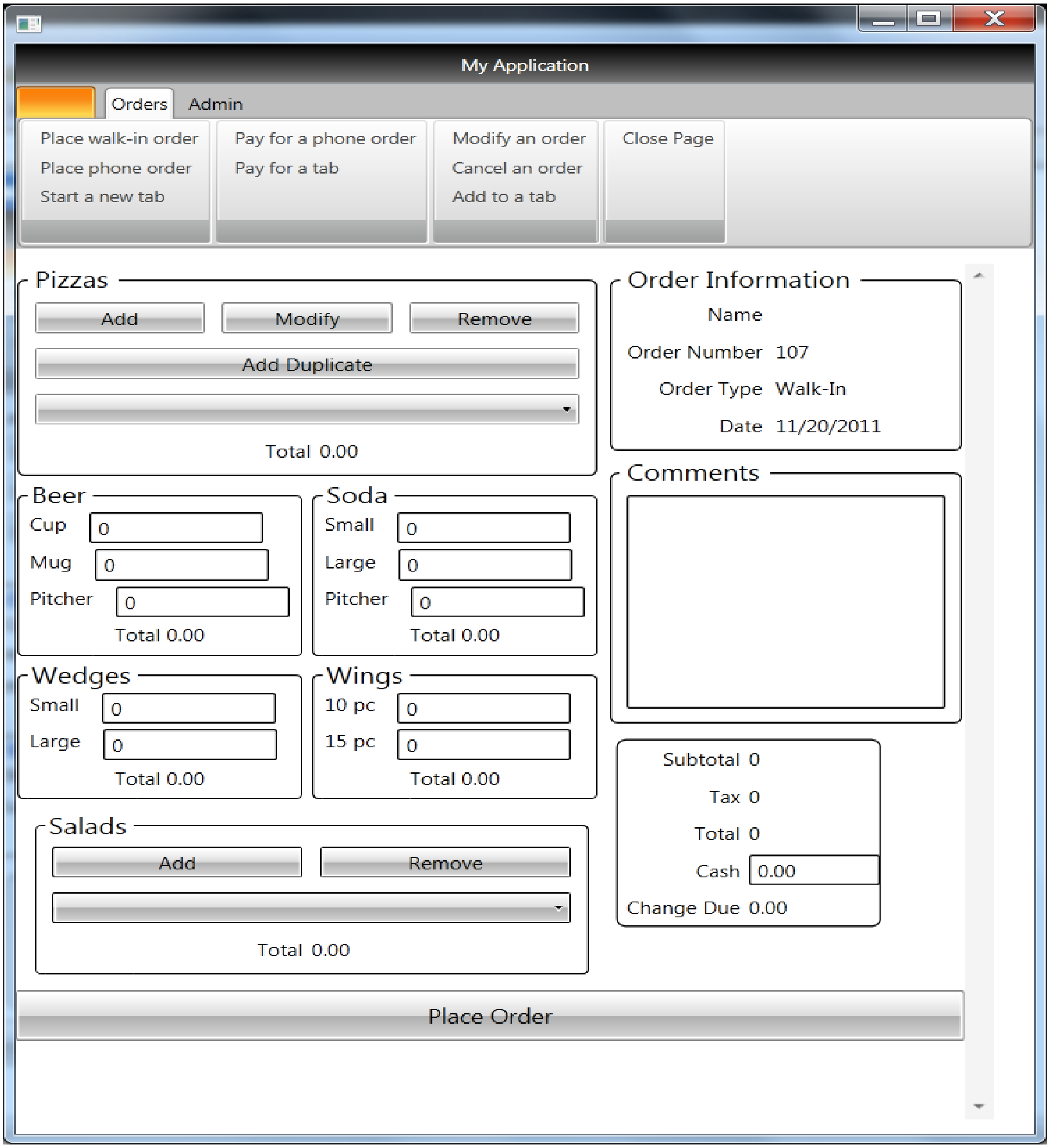
- **General Users:** Are cash register operators who take order by phone or walk-in. Their duty is to make note of the type, size, toppings, sauce, and cook preference of the pizza including the sides and beverages if any. The most important part of their job is to collect and dispense the correct amount of money per transactions .
- **Management:** This type of users can perform the same type of operations as General User but has the ability to override transactions and give discounts. For the most part the user spend most of his day taking care of business logistics and operations like determine whether extra help is needed or not based on gross sales reports. Management is not limited to daily reports, they may pull weekly or monthly reports as well.
- **Administrators:** The Pizza Parlor owner can help perform all the daily duties of the two users groups mentioned but he is mainly interested in business operation costs to profit ratios by viewing reports and introducing or removing promotions to stimulate revenue.

15 Menu and Display

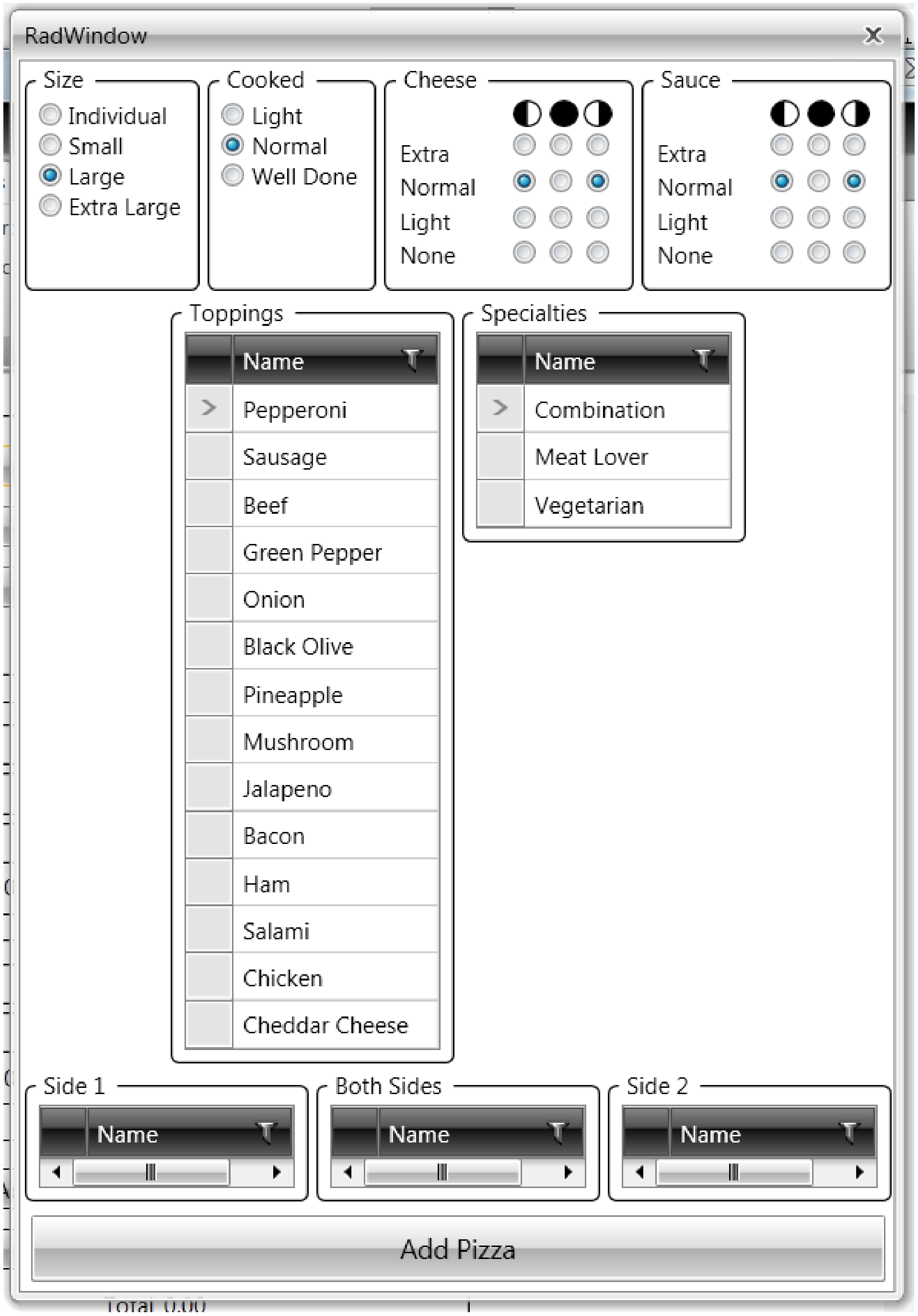
This is the main menu the user will first see before selecting the operation mode restricted by the user permissions.



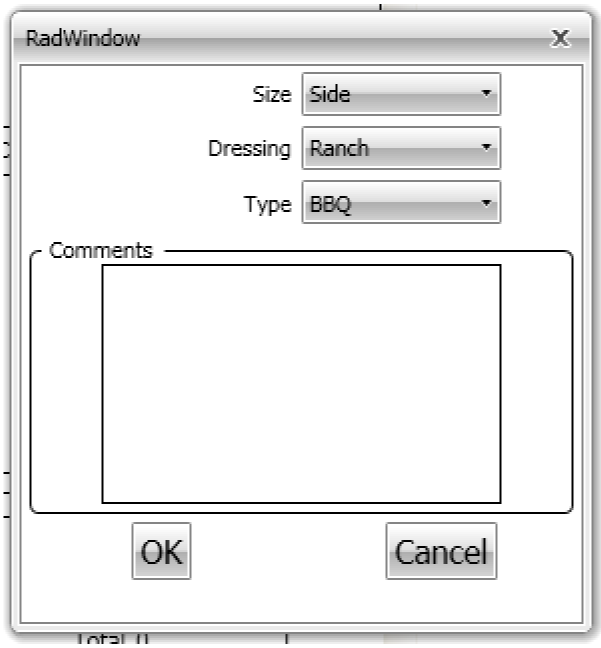
This screen shot will appear for users who select to place a 'Walk-in' or 'Phone' order.



Selecting 'Add' Pizza, from the Order menu, will produce a pop-up menu that allows you to compose all aspects of the pizza.



Like the Pizza menu, a pop-up menu appears when 'Add' is selected under Salads.



16 Code Implementation

Coding the project was challenging due to the compound use of languages , programming tools, and platforms like C#, SQL, Oracle, Microsoft (MS) Visual Studio, and MS Windows Presentation Foundation with Telerik RadControls. For questions regarding coding and debugging we turned to online resources to help with issues with triggers, stored procedures, stored functions, and exception handling. Finding code syntax was the best tool we found to code through database. The most important aspect of the project was to collaborate as a team and stay focused on the big picture. As group members we participated in daily meetings before and after class periods where we discuss concepts, new and existing issues, anomaly resolutions, and member assignments in order to complete the project in a timely manner. Fact finding and discussing ways to create the best database for a Pizza Parlor was an exciting and informative phase of our project but it was the coding that prove to be a fun an rewarding challenge. Working in an environment where different members have different ideas and everyone has to agree on one thing to move the project forward allowed us to build relationships, refine communications techniques, and improve are time management skills. This environment of having to work in a team with specific instructions and existing platform, like Oracle, prove to be a challenge and real world like work experience.