Every Meal Delivery

A Database Systems Project

Hector Velasco

Computer Science 342: Database Systems
Prof. H. Wang

11.28.2010

Table of Contents

Phase I

Fact-Finding
Information Gathering
and
Conceptual Database Design

Part 1: Fact-Finding Techniques, and Information Gathering

1.1 Description of Fact-Finding Techniques

Fact-finding techniques include but are not limited to five common methods. These methods include:

- **Examining documentation** of the business model being studied.
  - Businesses commonly carry with them many forms of documentation including forms, reports, and various other files. This information is invaluable to anyone researching the current business structure.

- **Interviewing** people who run the business.
  - Gathering information from people who work within the business first-hand is also extremely helpful. An individual's perspective on how the job runs may give clues on how to better organize the database to suit the business's needs.

- **Observing the Enterprise in Operation** through objective analysis.
  - Either personally participating, or watching from a third hand perspective, may help in developing a model translated from real time activity within the business. Be aware, however, the observer's presence may affect how business is regularly run.

- **Research** the application and problem.
  - Good research may enable one insight on how others have solved similar problems. As a source for reference it may be good to look up information found inside computer trade journals, reference books, and the Internet.

- **Questionnaires** through use of surveys can also be made.
  - Questions can be asked of target audiences. Through responds to questions made one can assess the needs of the organization. There are two forms of Questionnaires: Free-format, and Fixed-format. Free-format allows the users more options in responding to questions posed, where as fixed-format offers an initial set number of available choices as answers.

1.2 Techniques Used

The business model being used is devised from scratch. Many of the techniques used deal mostly with research, though there are similar models already available. The business model for this database is not completely unheard of. Even though there are other businesses that offer similar services, this one is much more unique in scope. In the future however, if a more representative example of a real-world model applies, then any and all of the already mentioned fact-finding methods may apply.

1.3 Introduction to the Enterprise/Organization

Every Meal is a food delivery business able to be operated from any location near fast food restaurant locations. After customers register with the possibility of premium membership status, they can choose to order from an online menu listing items from many popular food dining locations such as Taco Bell, McDonald's, Starbucks, Panda Express, and other similar venues. After completing an order from a combination of any of the available restaurants, the customer's order is then delivered to any location of choice. Customers who pay for a premium membership qualify for cheaper delivery rates.

1.4 Structure of the Enterprise/Organization

For this business model employees handle all customer orders, and deliveries. Employees are dispatchers, and delivery drivers. Though there is a manager supervising everything that goes on, she/he can assume either the role of dispatcher or delivery driver at any moment. The role of dispatcher will involve not only setting up the customer's order, but also selling the customer a premium membership offering cheaper delivery costs should the customer like to purchase one. After a dispatcher enters the order, and assigns it to a driver, a driver goes to all required food locations, purchases the requested customer items, and delivers the completed order to the any one of the customer's addresses. At any moment the driver will also carry along a mobile device to keep him/herself updated on orders that need to be completed.

1.5 Itemized Descriptions of Major Objects

At any time, depending on the schedule, any employee can be either a driver or a dispatcher, but never both at the same time. If an employee works as a driver, the delivery mileage for the employee will be noted, but should an employee work as a dispatcher, he/she will serve to take incoming orders.

Customers who call in will be registered into the system by the dispatcher if the customer is new, or wanting to apply for a premium membership. The date, membership status, and price of premium membership status are saved as a registration relation between the customers, and dispatchers. All customer objects will have not only full customer name

information, but also current membership status. Because the customer may have multiple addresses, all addresses for customer objects will also be saved.

Every order will be placed by a customer, entered into the system by a dispatcher, and delivered by a driver. Each order will contain a date, delivery status (complete, or in progress), a delivery fee, customer delivery destination, and customer member status. Delivery drivers will also have a relation log not only for the time in which they receive the order, but in which they deliver the order as well.

Orders will include any number of items offered by franchises of popular restaurants. This project will only involve the option of having orders with some items from franchises of four popular restaurants (Carl's Jr, Starbucks Coffee Company, Taco Bell, and Panda Express), but it can be expanded to include any number of restaurant objects having any number of franchise objects offering any number of item objects available for inclusion in orders.

Restaurants will only carry a name, but they will have franchise objects with individual address, phone, and easy reference location number. These franchise objects will offer any number of items at different prices. Some item objects will even be lower group cost combo items referring back to individual items.

1.6 Data Views and Operations for User Groups

Views available will differ for each person depending on the type of person category. Employees, for example, will have two views available depending on the position: one for a driver employee, and one for a dispatcher. Even though the manager can at any time take the role of either driver or dispatcher, he/she will also have access to full statistical views on every employee, customer, item purchases, etc. The manager will also be able to add, or edit the menu items list, and the restaurants from which they come. Customers will only have access to view available items for order from a web page menu.

Drivers will be able to view a personal listing of all orders delivered, and orders yet to be delivered with statistics on how many miles were run for any day. They'll also be able mark off orders as they're completed.

Dispatcher view will deal mainly with order setup. It will allow dispatchers to enter new customers and customer orders with customer locations, to assign orders to drivers, and to designate to drivers from which franchises the items are to be bought. Each dispatcher will have access to a personal account view of every customer order he/she has ever entered into the system.

Part 2: Conceptual Database Design

Conceptual Modeling

Conceptual modeling is the next step in the database construction process. In this step we formulate what is known as the conceptual data model, which is independent of all implementation details of the final database. In later sections there will be more physical considerations in implementing the project. An entity relationship, or ER model, will also be panned out. The ER model is a non-technical graphical representation of the database without any ambiguities.

2.1 Entity Set Descriptions

Entity: Employee

Description:

The employee entity is used as a generalized superclass for two employee subcategories: driver, and dispatcher. All employees, including the manager, can be objects of this class of entity. Only the employee's full name, setup as a composite attribute necessary for basic identification purposes, is included in this entity.

Candidate Keys: EmployeeID
Primary Key: EmployeeID
Strong/Weak Entity: Strong
Fields to be indexed: EmployeeID

Attributes and Details

Name	EmployeeID	Name	eFirst	eMidInitial	eLast
Description	Employee ID number	Employee name	Employee's first name	Employee's middle initial	Employee's last name
Domain/Type	Unsigned Integer	String	String	String	String
Value Range	0 … 2^{32}	Any	Any	A…Z	Any
Default Value	None	None	None	None	None
Nullable?	No	No	No	Yes	No
Unique?	Yes	No	No	No	No
Single or multiple value	Single	Single	Single	Single	Single
Simple or composite	Simple	Composite	Simple	Simple	Simple

Entity: MileageLog

Description:

The MileageLog entity is a mileage log for delivery drivers. Its primary function is to hold the start, and end driving times as well as the start, and end car mileage for the sake of calculating the miles covered by a driver during the day.

Candidate Keys: SDate
Primary Key: SDate
Strong/Weak Entity: Weak
Fields to be indexed: SDate

Attributes and Details

Name	SDate	EDate	StartMileage	EndMileage
Description	Driving start date, and time	Driving end date, and time	Car mileage at start of day	Car mileage at end of day
Domain/Type	TIMESTAMP	TIMESTAMP	Unsigned Integer	Unsigned Integer
Value Range	Date/Time	Date/Time	$0 \ldots 2^{32}$	$0 \ldots 2^{32}$
Default Value	None	None	None	None
Nullable?	No	Yes	No	Yes
Unique?	Yes	Yes	No	No
Single or multiple value	Single	Single	Single	Single
Simple or composite	Simple	Simple	Simple	Simple

Entity: LogTable

Description:

This is a log that keeps track of when customer information is either updated, or deleted.

Candidate Keys: LogID
Primary Key: LogID
Strong/Weak Entity: Weak
Fields to be indexed: LogID

Attributes and Details

Name	ItemID	OldVal	NewVal
Description	Log ID number	Old Customer Record	New Customer Record
Domain/Type	Unsigned Integer	String	String
Value Range	$0 \ldots 2^{32}$	Any	Any
Default Value	None	None	None
Nullable?	No	No	Yes
Unique?	Yes	No	No
Single or multiple value	Single	Single	Single
Simple or composite	Simple	Composite	Composite

Entity: Customer

Description:

The entity Customer holds all basic customer information as well as the customer's current membership status. The customer's full name is setup as a composite attribute necessary for basic identification purposes.

Candidate Keys: CustomerID
Primary Key: CustomerID
Strong/Weak Entity: Strong
Fields to be indexed: CustomerID

Attributes and Details

Name	CustomerID	Name	cFirst	cMidInitial	cLast	Phone	MemberStatus
Description	Customer ID number	Customer name	Customer first name	Customer middle initial	Customer last name	Phone number	Premium Status Flag
Domain/Type	Unsigned Integer	String	String	String	String	Unsigned Integer	Unsigned Integer
Value Range	0 … 2^{32}	Any	Any	A…Z	Any	0 to 9999999999	0, 1, or 2
Default Value	None	None	None	None	None	None	0
Nullable?	No	No	No	Yes	No	No	No
Unique?	Yes	No	No	No	No	No	No
Single or multiple value	Single	Single	Single	Single	Single	Single	Single
Simple or composite	Simple	Composite	Simple	Simple	Simple	Simple	Single

Entity: CstmrAddress

Description:

The CstmrAddress entity is a customer address book. It serves as a way of allowing the customer to have more than one address to which orders can be delivered. With it a customer can have any number of addresses.

Candidate Keys: AddressID
Primary Key: AddressID
Strong/Weak Entity: Weak
Fields to be indexed: AddressID

Attributes and Details

Name	AddressID	Street	Apt	City	Zip
Description	Address ID number	Number with /street name	Apartment number	City name	Zip code
Domain/Type	Unsigned Integer	String	String	String	Unsigned Integer
Value Range	0 … 2^{32}	Any	Any	A…Z	0 - 99999
Default Value	None	None	None	None	None
Nullable?	No	No	Yes	No	No
Unique?	Yes	No	No	No	No
Single or multiple value	Single	Single	Single	Single	Single
Simple or composite	Simple	Simple	Simple	Simple	Simple

Entity: Order

Description:

The order entity holds the basic information important for each order. Each order contains:

- The date on which the dispatcher entered the order
- The current status of the order showing if the order's been delivered, or if it's still to be delivered
- The delivery charge (not counting the cost of items ordered)
- The destination address referenced from the customer's address book
- A copy of the customer's membership status at the time of the order

Candidate Keys: OrderNumber
Primary Key: OrderNumber
Strong/Weak Entity: Weak
Fields to be indexed: OrderNumber

Attributes and Details

Name	OrderNumber	oDate	OrderStatus	DeliveryCharge	Destination	oMemberStatus
Description	Order ID number	Order entry date	Order's current status	Delivery cost	Customer Address	Membership status log
Domain/ Type	Unsigned Integer	TIMESTAMP	Integer	Float	Unsigned Integer	Boolean
Value Range	$0 \ldots 2^{32}$	Date/Time	0-2	> 0.00	$0 \ldots 2^{32}$	0 or 1
Default Value	None	None	0	None	None	0
Nullable?	No	No	No	Yes	No	No
Unique?	Yes	No	No	No	No	No
Single or multiple value	Single	Single	Single	Single	Single	Single
Simple or composite	Simple	Simple	Simple	Simple	Simple	Single

Entity: Restaurant

Description:

Restaurant is an entity name holder for a general restaurant name, be it the name of a corporate chain of franchises, or a single name for a restaurant belonging to a single owner. It's a weak entity, because it requires at least one franchise entity object to exist.

Candidate Keys: RstID
Primary Key: RstID
Strong/Weak Entity: Weak
Fields to be indexed: RstID

Attributes and Details

Name	**RstID**	**rName**
Description	Restaurant ID Number	Restaurant Name
Domain/Type	Unsigned Integer	String
Value Range	0 … 2^{32}	Any
Default Value	None	None
Nullable?	No	No
Unique?	Yes	No
Single or multiple value	Single	Single
Simple or composite	Simple	Simple

Entity: Franchise

Description:

Franchise is an entity that denotes a single location of a restaurant chain. It contains a simple numbered reference location tag to quickly denote the restaurant location within that specific restaurant chain (ex. Numbers 1 through 5 for the five Taco Bells in the area). Address, and phone number for the franchise location is also contained in this entity.

Candidate Keys: FID
Primary Key: FID
Strong/Weak Entity: Weak
Fields to be indexed: FID

Attributes and Details

Name	FID	Address	fStreet	fCity	fZip	fPhone
Description	Franchise ID number	Franchise address	Number with /street name	City name	Zip code	Phone number
Domain/Type	Unsigned Integer	String	String	String	Unsigned Integer	Unsigned Integer
Value Range	0 … 2^{32}	Any	Any	A…Z	0 - 99999	0 to 9999999999
Default Value	None	None	None	None	None	None
Nullable?	No	No	No	No	No	No
Unique?	Yes	Yes	No	No	No	No
Single or multiple value	Single	Single	Single	Single	Single	Single
Simple or composite	Simple	Composite	Simple	Simple	Simple	Simple

Entity: Item

Description:

Items which can be ordered are contained within the Item entity class. This class contains the name of the item, and the item's price.

Candidate Keys: ItemID
Primary Key: ItemID
Strong/Weak Entity: Weak
Fields to be indexed: ItemID

Attributes and Details

Name	ItemID	iName	Price
Description	Item ID number	Item Name	Item price
Domain/Type	Unsigned Integer	String	Float
Value Range	0 … 2^{32}	Any	> 0.00
Default Value	None	None	None
Nullable?	No	No	Yes
Unique?	Yes	No	No
Single or multiple value	Single	Single	Single
Simple or composite	Simple	Simple	Simple

Entity: Combo

Description:

Combo items are types of Items composed of other items. This class contains no descriptive attributes.

Candidate Keys: Combo_ID
Primary Key: Combo_ID
Strong/Weak Entity: Weak
Fields to be indexed: Combo_ID

Attributes and Details

Name	Combo_ID
Description	Combo ID number
Domain/Type	Unsigned Integer
Value Range	0 … 2^{32}
Default Value	None
Nullable?	No
Unique?	Yes
Single or multiple value	Single
Simple or composite	Simple

2.2 Relationship Set Descriptions

Relationship: Registers

Description:

Registers is the relationship between the dispatcher, and customer entity types. It's a binary relationship noting that each dispatcher employee can register one or more customers, and that each customer deals with multiple dispatchers with member status changes made over time. It has two descriptive fields: Date and SellMembership. The registration date, or member status update date is saved in the Date field. SellMembership is an indicator for whether the customer chose to pay for a premium membership upon first entry into the system or update to his/her account.

Entity Sets Involved: Employee, and Customer
Mapping Cardinality: Many to Many
Participation Constraint: Optional

Descriptive Fields

Name	rDate	SellMembership
Description	Registration date	Membership sale status
Domain/Type	TIMESTAMP	Integer
Value Range	Date/Time	0-2
Default Value	None	0
Nullable?	No	No
Unique?	No	No
Single or multiple value	Single	Single
Simple or composite	Simple	Simple

Relationship: Logs

Description:

Logs is the relationship between the customer, or more specifically the customer's information, and the log tracker keeping track of what changes are made to a Customer's records.

Entity Sets Involved: LogTable, and Customer
Mapping Cardinality: Many to One
Participation Constraint: Optional

No Descriptive Fields

Relationship: Places

Description:

Places is a binary relationship between the customer, and the order made. It denotes the fact that any one customer can place any number of orders.

Entity Sets Involved: Customer, and Order
Mapping Cardinality: One to Many
Participation Constraint: Mandatory

No Descriptive Fields

Relationship: Enters

Description:

Enters is a binary relationship between the dispatcher employee, and the order setup. One dispatcher can setup any number of orders.

Entity Sets Involved: Employee, and Order
Mapping Cardinality: One to Many
Participation Constraint: Mandatory

No Descriptive Fields

Relationship: Delivers

Description:

Delivers is the binary relationship between a delivery driver, and the order being delivered. At any time one driver may deliver any number of orders. The relationship has two time-based descriptive fields. DOReceiptTime is the time the driver receives the order, and DeliveryTime is the time the driver delivers the order.

Entity Sets Involved: Employee, and Order
Mapping Cardinality: One to Many
Participation Constraint: Optional

Descriptive Fields

Name	DOReceiptTime	DeliveryTime
Description	Driver order receive time	Driver order delivery time
Domain/Type	TIMESTAMP	TIMESTAMP
Value Range	Date/Time	Date/Time
Default Value	None	None
Nullable?	No	Yes
Unique?	No	No
Single or multiple value	Single	Single
Simple or composite	Simple	Simple

Relationship: Includes

Description:

Includes is a ternary relation between entities Franchise, either of 2 potential kinds of the same Item entity, and Order. Included combo items contain at least two other items, so they are included differently. Sometimes a customer order may only involve purchasing a premium membership item instead of food. This relation also includes two descriptive fields: Quantity and Price for when an order does involve food items. The number of a certain kind of item purchased is saved in the Quantity field, and a copy of the individual item price at the time the order was made is saved in the Price field.

Entity Sets Involved: Item, Franchise and Order
Mapping Cardinality: Many to Many
Participation Constraint: Optional

Descriptive Fields

Name	Quantity	oPrice
Description	Number of items of a single type	Item price at order time
Domain/Type	Unsigned Integer	Float
Value Range	0 … 2^{32}	> 0.00
Default Value	None	None
Nullable?	No	Yes
Unique?	No	No
Single or multiple value	Single	Single
Simple or composite	Simple	Simple

Relationship: Contains

Description:

Contains is the many to many (2 or more) relationship between a combo item, and at least two or more subitems composing the combo.

Entity Sets Involved: Combo, Item
Mapping Cardinality: Many to Many
Participation Constraint: Optional

No Descriptive Fields

Relationship: Offers

Description:

Offers is a binary relationship between a franchise, and the items available for sale at the franchise.

Entity Sets Involved: Franchise, and Item
Mapping Cardinality: Many to Many
Participation Constraint: Mandatory

No Descriptive Fields

2.3 Relationship Entity Sets

Aggregation Relationships

Relationship: Customer has CstmrAddress

Entity Sets Involved: Customer and CstmrAddress
Mapping Cardinality: One to Many
Description:
This relation represents a customer having any number of addresses.

Relationship: Order has CstmrAddress

Entity Sets Involved: Order and CstmrAddress
Mapping Cardinality: Many to One
Description:
This relation represents a delivery destination location for any number of orders.

Relationship: Employee has MileageLog

Entity Sets Involved: Employee and MileageLog
Mapping Cardinality: One to Many
Description:
This relation represents an employee driver having many daily mileage logs.

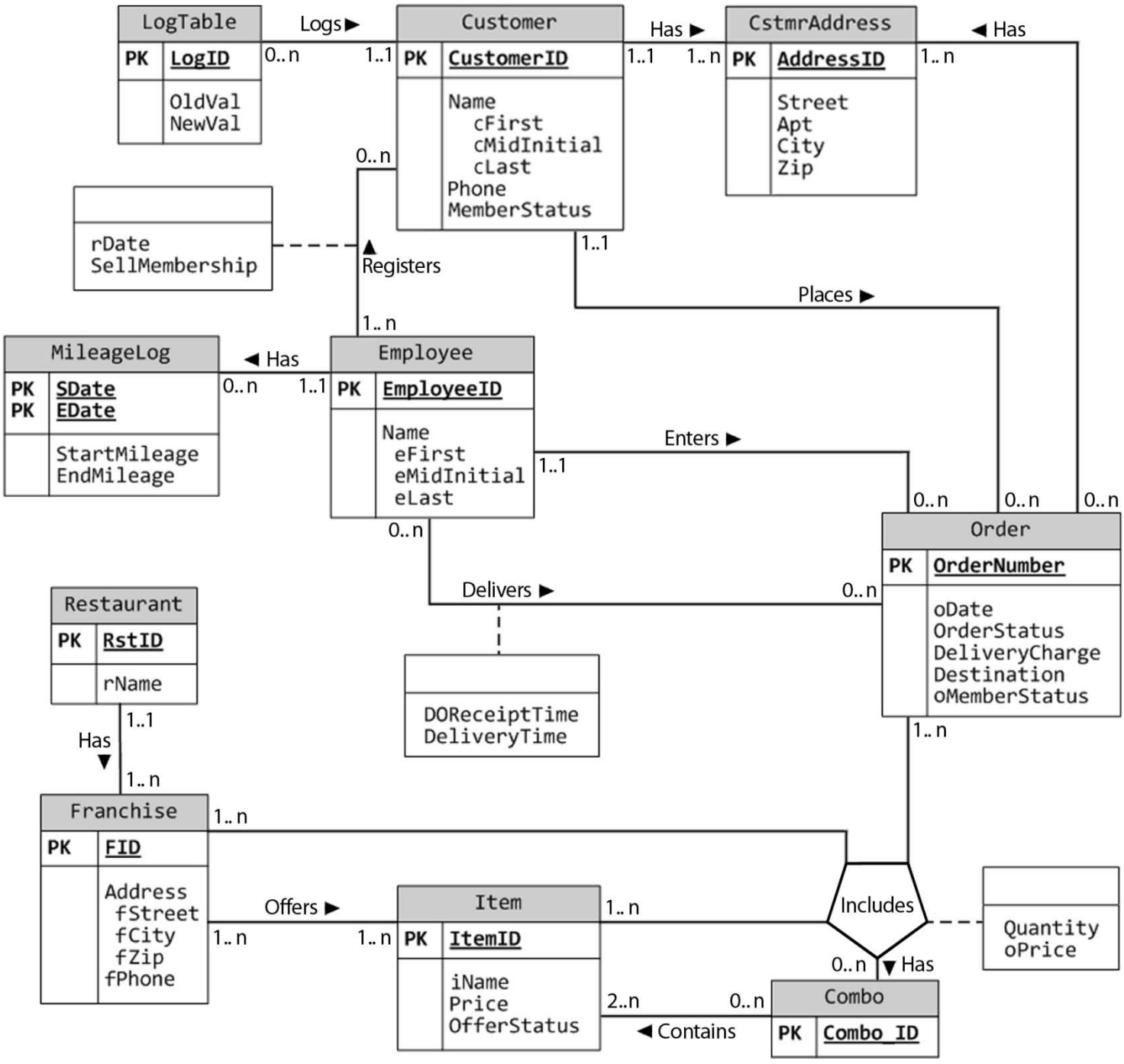
Relationship: Restaurant has Franchise

Entity Sets Involved: Restaurant and Franchise
Mapping Cardinality: One to Many
Description:
This relation represents a restaurant company having any number of franchises.

Relationship: Includes has Combo

Relationship\Entity Involved: Includes and Combo
Mapping Cardinality: Zero to Many
Description:
This relation represents combos being included in orders sometimes.

2.4 The Entity-Relationship Diagram



Phase II

From E-R (Conceptual) Model to Relational (Logical) Model

Part 1: The E-R Model, and the Relational Model

1.1 Descriptions of the Relational and Entity Relationship Models

The relational model is a mathematical concept of a relation. In physical terms the relational model is represented as a table. It was originally proposed by E.F. Codd in his 1970 seminal paper 'A relational model of data for large shared data banks'. Although set-oriented models had been presented previously, Codd's model is principally also set theory, and predicate logic. It allows a high degree of data independence, and provides substantial grounds for dealing with data semantics, consistency, and redundancy problems, which enables the expansion of set-oriented data manipulation languages.

At the time however, the relational model was insufficiently clear. To compensate, the authoritative reference for the entity-relationship (E-R) model was presented in 1976 in American computer scientist Dr. Peter Pin-Shan Chen's paper titled "The Entity-Relationship Model—Toward a Unified View of Data", a now widely accepted technique for database methodology. The E-R model presented in the paper is a non-technical top-down visual approach to database modeling free of ambiguities. Important entity data is identified and relationships between entities are clearly represented in the model.

1.2 Comparison of the Two Different Models

There are a few differences between the relational and E-R models. For one, the E-R model is a graphic diagrammatic representation of the conceptual database, where as the relational model is a textual representation which uses tables to represent its data. In the E-R model entities are represented graphically, with attributes listed very clear for each entity. Since the E-R model's purpose is to remove all ambiguities, it is kept visual for easy understanding. The E-R model uses a top-down approach for database design that identifies entities, and relationships between data. Attribute information to hold about the entities and relationships with any constraints on the entities, relationships, and attributes is represented in the model. The benefit of having such a model is that it allows both the designer and client to have an intuitive representation of the nature of the data, and how it is used by the enterprise.

This is where the relational model differs. Relational database design extends from the E-R model to the more detailed relational model. In the relational model, database relations are represented as tables, with relation attributes represented as columns in the tables. Each record in a relation is called a tuple, and each tuple is a row in the table of the represented relation. The relational model may make each entity, and relation between each entity into a table, and have the attributes for each clearly labeled within the table. However, the relationships between any of these tables aren't as clear to see as they are in the E-R model.

1.3 Conversion from E-R Model to Relational Model

To create a useful database the original conceptual database model must be converted to a physical form which can used by a database management system so an application can be made to use the data for whatever purpose an enterprise may have in store. To do this the visual E-R model is translated to a relational model before being physically coded. Generally, converting the E-R model to a relational model involves making each entity in the E-R model a relational table with columns in the table representing each attribute of the entity converted. However, there are various possible translation methods used within this general technique to handle converting various entity types, and relations.

1.3.1 Entity Type, and Attribute Conversion Issues

Strong Entities

In strong entity types a relation is made composed of attributes with composite attributes broken into simpler component attributes. This includes whatever attributes are designated as primary key in the relation.

Weak Entities

Weak entity types also require making a relation composed of attributes with composite attributes broken into simpler component attributes. However, one attribute in the relation must refer to a primary key from another relation on which the weak entity depends. This key is to be the foreign key attribute which also serves as primary key for the relation, perhaps even combined with another partial key attribute, if any.

Multi-valued Attributes

For multi-valued attributes in any entity types a new relation is made composed of the attribute, or component attributes (in case of multi-valued composite attributes) along with a foreign reference to the entity to which the multi-valued attribute belongs. This foreign key attribute acts as primary key for the relation.

1.3.2 Relationship Conversion Issues

The following lists the types of possible relationships available between entity relations with ways to translate each relationship type into the relational model.

1:1 Relationship Types

1. **Foreign Key Approach**: take the primary key attribute from one entity, and add it as a foreign key attribute in the entity to which the relationship is held.

2. **Merged Relation Approach**: if the both entities have total participation in the relationship with one another, it may be best to merge all the attributes together into one relation.

3. **Cross-reference, or Relationship Relation Approach:** commonly referred to as a relationship relation, this method involves creating an entirely separate relation from the other two entity relations to hold the foreign key attributes referencing each entity relation. Together, the foreign key attributes combine to form a primary key.

1:N Relationship Types

To translate 1:N relationships, it is necessary to take the foreign key approach, whereby the primary keys from the entity on the 1 side of the relationship is placed as a foreign key attribute to the N side entity relation. If too few tuples participate in the N side entity relation, it may be a better idea to setup a relationship relation between the two entities, thus avoiding the use of excessive NULLs.

M:N Relationship Types

M:N relationships are formed through use of the relationship relation method. If there are descriptive attributes in the relationship between the entities, the attributes are added as extra simple attributes of the relationship relation between the two entity relations.

Relationships Involving N Entity Types

To translate this kind of relationship a relation is made composed of N foreign key attributes from the primary key attributes from the N entities. Any descriptive attributes part of the N-ary relationship are also included.

Recursive Relationship Types

In order to translate recursive relationships either foreign key or relationship relation methods can be used with the condition that the entity refer back to itself. In the case of the foreign key method, the extra foreign key attribute is added to the same entity relation containing the primary key attribute used as a foreign key. Using the relationship relation method would entail using a separate relation containing both primary, and foreign fields from the same entity.

Relationships Involving Categories (Union Types)

This situation involves multiple superclass entities all to one subclass entity. Translating this situation requires having a foreign key field in a relation for each superclass referring to the primary key field in a relation for the subclass entity.

'HasA' Relationship Types

'HasA' relationships can be translated using the same methods used to convert 1:N relationships.

'IsA' Superclass/Subclass Relationship Types

1. **Multiple relations – Superclass and subclasses.** Use the primary key attribute in a relation for the superclass entity as a primary key attribute in a relation for each subclass entity. This option works for all specialization types (total or partial, disjoint or overlapping).

2. **Multiple relations – Subclass relations only.** Create a relation for each subclass that includes not only its own attributes, but also those of the superclass. The primary key from the superclass entity which is included in the superclass entity's attributes included with the subclass entity's attributes is the relation's primary key. This option only works for cases where every entity in the superclass belongs to at least one of the subclasses.

3. **Single Relation with one type attribute.** Make a single relation containing the attributes from the superclass, and every subclass together with the primary key from the superclass as the primary key for the relation. Then add an extra discriminating attribute used to label each tuple in the relation according to the subclass in which it belongs. This option is aimed at cases where the subclasses are disjoint, and it has the potential to generate many NULL values.

4. **Single Relation with multiple type attributes.** Make a single relation containing the attributes from the superclass, and every subclass using the primary key from the superclass as the relation's primary key. Then add a set of multiple extra Boolean type discriminating attributes used to indicate the subclass to which each tuple belongs. This method works both for overlapping, and disjoint subclass specialization.

1.3.3 Constraints and DBMS Constraint Enforcement

Integrity constraints ensure that data entered into the database is accurate. They do this by setting rules over what can be entered into a record's attribute fields. There are two integrity rules that apply to all instances of a database. These rules are entity integrity, and referential integrity. Entity integrity constraints apply to primary keys in that a standard is set whereby no primary key may contain a null value. Unique and primary keys are constrained as well. When a foreign key exists in a relation, the foreign key value must match the primary key value of some tuple in its home relation. Otherwise, the foreign key must be wholly null. This is what's referred to as referential integrity. Other forms of constraint may also be enforced depending on the DBMS being run, and the business rules specified by the owners of the database system.

Part 2: E-R Database to Relational Database Schema

2.1 Entity Relations

Relation: Employee (EmployeeID, eFirst, eMidInitial, eLast)

Attributes and Domain Details

EmployeeID	unsigned integer (0 to $2^{32} - 1$), unique, not NULL
eFirst	character string, only numbers, and letters, beginning with a character, not unique, not NULL
eMidInitial	single letter character, not unique
eLast	character string, only numbers, and letters, beginning with a character, not unique, not NULL

Constraints

-Primary Key-	EmpoyeeID acts as the primary key which is unique, and not NULL.

Relation: MileageLog (EmployeeID, SDate, EDate, StartMileage, EndMileage)

Attributes and Domain Details

EmployeeID	unsigned integer (0 to $2^{32} - 1$), unique, not NULL
SDate	TIMESTAMP, unique, not NULL
EDate	TIMESTAMP, unique when not NULL
StartMileage	unsigned integer (0 to $2^{32} - 1$), not unique, not NULL
EndMileage	unsigned integer (0 to $2^{32} - 1$), not unique

Constraints

-Primary Key-	SDate, and EDate combine together to make a unique non-NULL primary key.
-Foreign Key-	The primary key EmployeeID from the relation Employee acts a foreign key in this relation.

Relation: LogTable (LogID, OldVal, NewVal)

Attributes and Domain Details

LogID	unsigned integer (0 to $2^{32} - 1$), unique, not NULL
OldVal	character string, numbers, letters,not unique, not NULL
NewVal	character string, numbers, letters,not unique

Constraints

-Primary Key-	LogID acts as the primary key which is unique, and not NULL.

Relation: Customer (CustomerID, cFirst, cMidInitial, cLast, Phone, MemberStatus)

Attributes and Domain Details

CustomerID	unsigned integer (0 to $2^{32} - 1$), unique, not NULL
cFirst	character string, only numbers, and letters, beginning with a character, not unique, not NULL
cMidInitial	single letter character, not unique
cLast	character string, only numbers, and letters, beginning with a character, not unique, not NULL
Phone	unsigned ten digit integer, not unique, not NULL
MemberStatus	unsigned Boolean, not unique, not NULL

Constraints

-Primary Key-	CustomerID acts as the primary key which is unique, and not NULL.

Relation: CstmrAddress (AddressID, Street, Apt, City, Zip, CustomerID)

Attributes and Domain Details

AddressID	unsigned integer (0 to $2^{32} - 1$), unique, not NULL
Street	character string, only numbers, and letters, beginning with a number, not unique, not NULL
Apt	character string, only numbers, and letters at most 5 digits long, not unique
City	character string, only letters, not unique, not NULL
Zip	unsigned five digit integer, not unique, not NULL
CustomerID	unsigned integer (0 to $2^{32} - 1$), unique, not NULL

Constraints

-Primary Key-	AddressID acts as the primary key which is unique, and not NULL.
-Foreign Key-	The primary key CustomerID from the relation Customer acts a foreign key in this relation.

Relation: Order (OrderNumber, oDate, OrderStatus, DeliveryCharge, oMemberStatus, Destination, EmployeeID, CustomerID)

Attributes and Domain Details

OrderNumber	unsigned integer (0 to $2^{32} - 1$), unique, not NULL
oDate	TIMESTAMP, not unique, not NULL
OrderStatus	unsigned integers 0, 1, or 2, not unique, not NULL
DeliveryCharge	float greater than zero, not unique
Destination	unsigned integer (0 to $2^{32} - 1$), unique, not NULL
oMemberStatus	unsigned Boolean, not unique, not NULL
EmployeeID	unsigned integer (0 to $2^{32} - 1$), unique, not NULL
CustomerID	unsigned integer (0 to $2^{32} - 1$), unique, not NULL

Constraints

-Primary Key-	OrderNumber acts as the primary key which is unique, and not NULL.
-Foreign Key-	The primary keys AddressID from the relation CstmrAddress, EmployeeID from the relation Employee, and CustomerID from the relation Customer act as foreign keys in this relation. EmployeeID and CustomerID keep the same names for foreign keys in this relation. AddressID is the foreign key Destination in this relation.
-Business Rule-	For each order the value of MemberStatus is copied from MemberStatus in the relation Customer, but it is not a foreign key in this relation.

Relation: Restaurant (RstID, rName)

Attributes and Domain Details

RstID	unsigned integer (0 to $2^{32} - 1$), unique, not NULL
rName	character string, only numbers, letters, and blank spaces, beginning with a character, not unique, not NULL

Constraints

-Primary Key-	RstID acts as the primary key which is unique, and not NULL.

Relation: Franchise (FID, fStreet, fCity, fZip, fPhone, RstID)

Attributes and Domain Details

FID	unsigned integer (0 to $2^{32} - 1$), unique, not NULL
fStreet	character string, only numbers, and letters, beginning with a number, not unique, not NULL
fCity	character string, only letters, not unique, not NULL
fZip	unsigned five digit integer, not unique, not NULL
fPhone	unsigned ten digit integer, not unique, not NULL
RstID	unsigned integer (0 to $2^{32} - 1$), unique, not NULL

Constraints

-Primary Key-	FID acts as the primary key which is unique, and not NULL.
-Foreign Key-	The primary key RstID from relation Restaurant acts as a foreign key in this relation by the same name.

Relation: Item (ItemID, iName, Price)

Attributes and Domain Details

ItemID	unsigned integer (0 to $2^{32} - 1$), unique, not NULL
iName	character string, numbers, letters, and dash '-' beginning with a character, not unique, not NULL
Price	float greater than zero, not unique

Constraints

-Primary Key-	ItemID acts as the primary key which is unique, and not NULL.

Relation: Combo (ComboID, Item_CR, OrderNum)

Attributes and Domain Details

ComboID	unsigned integer (0 to $2^{32} - 1$), unique, not NULL
Item_CR	unsigned integer (0 to $2^{32} - 1$), unique, not NULL
OrderNum	unsigned integer (0 to $2^{32} - 1$), unique, not NULL

Constraints

-Primary Key-	ComboID acts as the primary key which is unique, and not NULL.
-Foreign Key-	The foreign keys OrderNumber, and ItemId from relation Includes are included in this entity, also as foreign keys where OrderNum refers to OrderNumber, and Item_CR refers to ItemID.

2.2 Relationship Relations

Relation: Registers (EmployeeID, CustomerID, rDate, SellMembership)

Attributes and Domain Details

EmployeeID	unsigned integer (0 to $2^{32} - 1$), unique, not NULL
CustomerID	unsigned integer (0 to $2^{32} - 1$), unique, not NULL
rDate	TIMESTAMP, not unique, not NULL
SellMembership	unsigned integers 0, 1, or 2, not unique, not NULL

Constraints

-Primary Key-	EmployeeID from relation Employee, CustomerID from relation Customer, and TIMESTAMP Date from this relation combine together to make a primary key for this relation.
-Foreign Key-	The primary keys EmployeeID from the relation Employee and CustomerID from the relation Customer act as foreign keys in this relation. EmployeeID and CustomerID keep the same names for foreign keys in this relation.
-Business Rule-	Should the customer want to change membership status, the date of the change is recorded.

Relation: Delivers (EmployeeID, OrderNumber, DOReceiptTime, DeliveryTime)

Attributes and Domain Details

EmployeeID	unsigned integer (0 to $2^{32} - 1$), unique, not NULL
OrderNumber	unsigned integer (0 to $2^{32} - 1$), unique, not NULL
DOReceiptTime	TIMESTAMP, not unique, not NULL
DeliveryTime	TIMESTAMP, not unique

Constraints

-Primary Key-	EmployeeID from relation Employee, and OrderNumber from relation Order combine together to make a primary key for this relation.
-Foreign Key-	The primary keys EmployeeID from the relation Employee and OrderNumber from relation Order act as foreign keys in this relation. EmployeeID and OrderNumber keep the same names for foreign keys in this relation.

Relation: Offers (FID, ItemID)

Attributes and Domain Details

FID	unsigned integer (0 to $2^{32} - 1$), unique, not NULL
ItemID	unsigned integer (0 to $2^{32} - 1$), unique, not NULL

Constraints

-Primary Key-	FID from relation Franchise, and ItemID from relation Item combine together to make a primary key for this relation.
-Foreign Key-	The primary keys FID from the relation Franchise and ItemID from relation Item act as foreign keys in this relation. FID and ItemID keep the same names for foreign keys in this relation.

Relation: Includes (OrderNumber, ItemID, FID, Quantity, oPrice)

Attributes and Domain Details

OrderNumber	unsigned integer (0 to $2^{32} - 1$), unique, not NULL
ItemID	unsigned integer (0 to $2^{32} - 1$), not unique, not NULL
FID	unsigned integer (0 to $2^{32} - 1$), unique, not NULL
Quantity	unsigned integer at most 5 digits long, not unique, not NULL
oPrice	float greater than zero, not unique

Constraints

-Primary Key-	OrderNumber from relation Order and ItemID from relation Item combine to make the primary key for this relation.
-Foreign Key-	The primary keys OrderNumber from the relation Order, ItemID from the relation Item, and FID from relation Franchise act as foreign keys in this relation. OrderNumber, ItemID, and FID keep the same names for foreign keys in this relation.
-Business Rule-	All items from the same restaurant chain will be delivered from one of the franchises belonging in the chain. Price is a copy of Price from relation Item at the time the order is made.

Relation: Contains (Container, CmboNum, ItemID)

Attributes and Domain Details

Container	unsigned integer (0 to $2^{32} - 1$), unique, not NULL
CmboNum	unsigned integer (0 to $2^{32} - 1$), unique, not NULL
ItemID	unsigned integer (0 to $2^{32} - 1$), unique, not NULL

Constraints

-Primary Key-	Container is the primary key for this relation.
-Foreign Key-	The primary key ItemID from the relation Item acts as a foreign key in this relation. Foreign key CmboNum is a foreign key from entity Combo referring to Combo_ID.

Part 3: Relation Instances

3.1 Entity Relation Instances

Relation: Employee (EmployeeID, eFirst, eMidInitial, eLast)

EmployeeID	eFirst	eMidInitial	eLast
1	Stanley	L	Marsh
2	Brazuk		Pierce
3	Marsha		Bentley
4	Fred	S	Armstrong
5	Marisol	M	Stensos
6	Larry		Guytez
7	Peter	P	Patrone
8	Curtis	B	Whicks
9	Wei	H	Kai
10	Tiffany		Cooper
11	Allison		Jester

Relation: MileageLog (EmployeeID, SDate, EDate, StartMileage, EndMileage)

EmployeeID	SDate	EDate	StartMileage	EndMileage
2	10-OCT-06 10.05.42.437000	10-OCT-06 12.30.25.245960	128,050	128,065
2	10-OCT-06 13.05.50.498241	10-OCT-06 17.10.10.097899	128,067	128,098
2	10-OCT-07 12.08.13.128752	10-OCT-07 17.04.55.549876	129,005	129,018
5	10-OCT-04 10.03.05.050000	10-OCT-04 15.00.59.587769	95,745	95,761
6	10-OCT-01 07.01.20.197564	10-OCT-01 12.05.32.318425	145,876	145,905
6	10-OCT-01 13.02.45.448949	10-OCT-01 15.59.58.579147	145,914	145,941
8	10-OCT-01 07.05.37.368718	10-OCT-01 12.00.45.448147	70,754	70,773
8	10-OCT-05 06.55.45.447845	10-OCT-05 11.58.18.178127	70,810	70,843
11	10-OCT-02 06.58.15.148742	10-OCT-02 12.10.05.047482	160,424	160,443
11	10-OCT-02 12.45.13.126341	10-OCT-02 16.01.41.405612	160,445	160,467
11	10-OCT-07 12.05.04.035911	10-OCT-07 17.00.40.397381	160,530	160,563

Relation: LogTable (LogID, OldVal, NewVal)

LogID	OldVal	NewVal
1	15 Pete K Mandelov 8052145789 2	15 Jeremy F Buckley 1457124547 0
2	15 Jeremy F Buckley 457124 0	15 Kimcheck N Santi 4555621155 2
3	15 Kimcheck N Santi 4555621155 2	
4	13 Mendle H Barddrin 5452453521 2	13 Fran H Barddrin 5452453521 2
5	13 Fran H Barddrin 5452453521 2	
6	16 Kyle F Parsecks 5423527895 1	16 Kyle M Pendleton 5423545895 1
7	16 Kyle M Pendleton 5423545895 1	
8	18 Sammy O Bammy 8461543621 2	18 Sammy N Kresnevitch 4512300212 0
9	18 Sammy N Kresnevitch 4512300212 0	
10	43 Rod N Bison 7874541265 1	43 Red N Cheem 7874541455 0
11	55 Can T Pryle 8574925154 2	55 Blanka B Aaahk 2135468457 1
12	43 Red N Cheem 7874541455 0	
13	55 Blanka B Aaahk 2135468457 1	71 Mayor Hagar 7485968525 2
14	71 Mayor Hagar 7485968525 2	
15	31 Seargent f Jack 5454478415 2	

Relation: Customer (CustomerID, cFirst, cMidInitial, cLast, Phone, MemberStatus)

CustomerID	cFirst	cMidInitial	cLast	Phone	MemberStatus
1	Polly	G	Fredericks	6613314574	0
2	Rachel		Larson	6615893212	0
3	Henry	K	Ricks	6617589212	1
4	Saul	M	Steinbeck	6612059354	1
5	Daniel		Simpson	8053652241	1
6	Matt		Groening	6618614124	0
7	Karen	F	Chu	8057584790	1
8	Ryan	L	Scott	8057583787	0
9	Jeff		Green	6615894512	1
10	Manmeet		Chunta	6613584124	0
11	Greg	A	Ford	8059683641	0

Relation: CstmrAddress (AddressID, Street, Apt, City, Zip, CustomerID)

AddressID	Street	Apt	City	Zip	CustomerID
1	11421 Old Town Rd	A	Bakersfield	93312	1
2	118 Harvest Creek Rd		Bakersfield	93306	2
3	Gate 3, Elk Hill Rd		Taft	93268	2
4	423 Beyers St		Arvin	93203	3
5	12321 Dorsey Ct	3C	Bakersfield	93305	4
6	1775 Balvanera Ave		Lamont	93241	5
7	2321 Hialeah Park Ln	319	Bakersfield	93305	6
8	22314 Verdelho Ave		Bakersfield	93311	7
9	10017 Great Country Dr		Bakersfield	93306	7
10	905 Mayacamas Dr		Taft	93268	8
11	3421 Allene Way	G	Bakersfield	93302	9
12	5843 Red River Dr		Lamont	93241	10
13	342 Treasure Island St		Bakersfield	93383	5
14	44321 Abbott Dr	21	Bakersfield	93309	8
15	253 Alki Ct		Bakersfield	93306	3

Relation: Order (OrderNumber, oDate, OrderStatus, DeliveryCharge, oMemberStatus, Destination, EmployeeID, CustomerID)

OrderNumber	oDate	OrderStatus	DeliveryCharge	oMemberStatus	Destination	EmployeeID	CustomerID
1	10-SEP-30 07.05.41.407000	0	10.00	0	12	1	10
2	10-SEP-30 07.10.45.448241	2	10.00	0	7	7	6
3	10-SEP-30 08.08.13.128752	1	14.00	1	15	3	3
4	10-SEP-30 09.15.05.050000	2	20.00	0	10	8	8
5	10-SEP-30 09.45.20.197564	2	10.00	0	1	4	2
6	10-SEP-30 12.02.45.448949	0	7.00	1	1	9	1
7	10-OCT-01 07.04.37.368718	1	7.00	1	8	10	7
8	10-OCT-01 08.15.45.447845	1	10.00	0	5	1	4
9	10-OCT-01 09.05.15.148742	2	10.00	0	14	5	8
10	10-OCT-01 11.10.13.126341	2	21.00	1	11	3	9
11	10-OCT-01 14.05.04.035911	1	14.00	1	9	10	7

Relation: Restaurant (RstID, rName)

RstID	rName
1	Carls Jr
2	Starbucks Coffee
3	Taco Bell
4	Panda Express
5	McDonalds
6	Wendys
7	Jack in the Box
8	In and Out
9	Burger King
10	Del Taco
11	Pollo Loco

Relation: Franchise (FID, fStreet, fCity, fZip, fPhone, RstID)

FID	fStreet	fCity	fZip	fPhone	RstID
1	9500 Brimhall Rd Ste A	Bakersfield	93312	6615874859	1
2	9801 Hageman Rd	Bakersfield	93312	6615875199	2
3	9640 Hageman Rd.	Bakersfield	93312	6612134574	3
4	9200 Rosedale Highway 300	Bakersfield	93312	6615872316	4
5	4520 Coffee Road	Bakersfield	93312	6615879085	1
6	9200 Rosedale Hwy	Bakersfield	93312	6615873661	2
7	5121 Olive Drive	Bakersfield	93308	6613937718	3
8	5120 Stockdale Hwy Space A	Bakersfield	93309	6613232033	4
9	9000 Ming Ave Ste Q	Bakersfield	93311	6616652396	1
10	4420 Coffee Road Unit B	Bakersfield	93308	6615873661	2
11	3799 Rosedale Hwy	Bakersfield	93308	6613253862	3
12	5041 Gosford Rd F1	Bakersfield	93313	6616640391	4
13	5520 Stockdale Hwy	Bakersfield	93383	6613229857	1
14	13133 Rosedale Hwy	Bakersfield	93314	6618292651	2
15	3300 Buena Vista Rd	Bakersfield	93311	6616638131	3

Relation: Item (ItemID, iName, Price)

ItemID	iName	Price
1	Fries-Small	1.49
2	Fries-Medium	1.79
3	Fries-Large	1.89
4	Fries-CrissCut	1.99
5	Fries-Chili Cheese	2.99
6	Onion Rings	1.99
7	Fried Zucchini	1.99
8	Side Salad	1.79
9	Fish & Chips	4.99
10	Coffee Channel Islands Roasting Co	1.29
11	DASANI Water	1.49
12	Milk	1.09
13	Orange Juice	1.39
14	Shake-Oreo Cookie	2.99
15	Shake-Chocolate	2.99

Relation: Combo (ComboID, Item_CR, OrderNum)

ComboID	Item_CR	OrderNum
1	90	1
2	243	2
3	243	2
4	243	2
5	72	3
6	74	3
7	78	4
8	291	5
9	83	6
10	290	6
11	290	6
12	290	6
13	289	9
14	289	9
15	285	11

3.2 Relationship Relation Instances

Relation: Registers (EmployeeID, CustomerID, rDate, SellMembership)

EmployeeID	CustomerID	rDate	SellMembership
11	1	10-SEP-30 12.02.45.448949	0
3	2	10-SEP-30 09.45.20.197564	0
3	3	10-SEP-30 08.08.13.128752	0
11	4	10-OCT-01 08.15.45.447845	1
3	5	10-SEP-30 07.02.22.216201	0
5	6	10-SEP-30 07.10.45.448241	0
3	7	10-OCT-01 07.04.37.368718	0
11	8	10-SEP-30 09.15.05.050000	0
11	9	10-OCT-01 11.10.13.126341	1
7	10	10-SEP-30 07.05.41.407000	0
9	11	10-OCT-01 14.21.10.095610	0
10	12	10-OCT-01 15.22.37.365721	1
9	13	10-OCT-01 15.24.03.033121	0
10	14	10-OCT-01 15.27.00.000001	0
10	15	10-OCT-01 16.29.05.407000	0
11	16	10-OCT-01 16.35.31.311011	0
9	17	10-OCT-01 16.41.39.387421	0
3	18	10-OCT-01 15.22.37.365721	0
8	3	10-SEP-30 07.05.41.407000	1
3	7	10-OCT-01 14.05.04.035911	1
5	13	10-OCT-02 07.22.11.110001	1
5	19	10-OCT-02 07.35.32.315421	1
1	20	10-OCT-02 07.38.15.150005	0
1	21	10-OCT-02 07.55.57.567501	0
4	22	10-OCT-02 08.02.23.229127	0
1	23	10-OCT-02 08.32.03.031231	0
4	10	10-OCT-02 08.41.54.538124	1
10	24	10-OCT-02 10.00.01.006157	0
10	2	10-OCT-02 10.11.00.599131	1
3	15	10-OCT-02 10.43.36.360121	1
7	25	10-OCT-02 10.50.14.140022	1
7	26	10-OCT-02 14.17.24.237253	0
7	27	10-OCT-02 18.01.43.430505	0
7	28	10-OCT-02 18.03.06.055241	0
4	29	10-OCT-03 07.20.52.514443	0
1	30	10-OCT-03 07.39.31.310202	1
3	31	10-OCT-03 07.57.43.424290	0
7	18	10-OCT-03 08.30.45.447845	1
4	32	10-OCT-03 09.57.13.126341	0
10	33	10-OCT-03 10.11.05.407000	1
9	34	10-OCT-03 10.13.31.312011	0
3	35	10-OCT-03 10.51.13.128752	0
7	36	10-OCT-03 11.02.37.368718	0
9	37	10-OCT-03 11.09.30.300029	0
4	22	10-OCT-03 11.49.31.310008	1
6	5	10-OCT-03 11.55.25.248242	1
3	38	10-OCT-03 13.40.24.241212	0
9	39	10-OCT-03 14.33.01.010101	0
10	40	10-OCT-03 15.01.46.463636	1
10	41	10-OCT-03 15.19.59.580001	0
3	42	10-OCT-03 15.38.14.143215	1
6	43	10-OCT-03 15.47.33.325214	0
7	16	10-OCT-04 07.33.47.467512	1
1	3	10-OCT-04 08.02.51.510515	1
2	44	10-OCT-04 09.11.55.548625	0
8	45	10-OCT-04 10.33.39.386512	0
7	46	10-OCT-04 11.17.03.029898	0
1	47	10-OCT-04 11.38.01.010101	0
2	48	10-OCT-04 12.13.12.119999	1
11	49	10-OCT-04 13.04.35.347757	0
9	50	10-OCT-04 13.06.36.361231	0
4	37	10-OCT-04 15.45.00.597531	1
5	51	10-OCT-04 18.10.11.110120	0

Relation: Delivers (EmployeeID, OrderNumber, DOReceiptTime, DeliveryTime)

EmployeeID	OrderNumber	DOReceiptTime	DeliveryTime
11	1	10-SEP-30 12.02.45.448949	10-SEP-30 13.40.12.120101
2	2	10-SEP-30 09.45.20.197564	10-SEP-30 10.57.09.085729
5	3	10-SEP-30 08.08.13.128752	10-SEP-30 09.00.57.564134
11	4	10-OCT-01 08.15.45.447845	10-OCT-01 08.50.10.099456
6	5	10-SEP-30 07.02.22.216201	10-SEP-30 07.30.21.211201
8	6	10-SEP-30 07.10.45.448241	10-SEP-30 08.20.08.075751
6	7	10-OCT-01 07.04.37.368718	10-OCT-01 08.05.12.124253
11	8	10-SEP-30 09.15.05.050000	10-SEP-30 09.45.00.000005
11	9	10-OCT-01 11.10.13.126341	10-OCT-01 11.30.06.057865
2	10	10-SEP-30 07.05.41.407000	10-SEP-30 07.32.01.009000
9	11	10-OCT-01 14.21.10.095610	10-OCT-01 15.10.03.033410
10	12	10-OCT-01 15.22.37.365721	10-OCT-01 15.16.20.195721
9	13	10-OCT-01 15.24.03.033121	10-OCT-01 16.20.01.010121
10	14	10-OCT-01 15.27.00.000001	10-OCT-01 16.35.50.495007
10	15	10-OCT-01 16.29.05.047000	10-OCT-01 17.05.04.040005
11	16	10-OCT-01 16.35.31.311011	10-OCT-01 17.10.02.024334
8	17	10-OCT-01 16.41.39.387421	10-OCT-01 16.59.58.575713
6	18	10-OCT-01 15.22.37.365721	10-OCT-01 16.15.14.137312
5	19	10-SEP-30 07.05.41.407000	10-SEP-30 07.57.01.005029
2	20	10-OCT-01 14.05.04.035911	10-OCT-01 14.45.21.208217
5	21	10-OCT-02 07.22.11.110001	10-OCT-02 08.08.19.185721
5	22	10-OCT-02 07.35.32.315421	10-OCT-02 08.25.47.470576
1	23	10-OCT-02 07.38.15.150005	10-OCT-02 08.10.26.262003
1	24	10-OCT-02 07.55.57.567501	10-OCT-02 08.21.51.505555
4	25	10-OCT-02 08.02.23.229127	10-OCT-02 08.44.17.170122
1	26	10-OCT-02 08.32.03.031231	10-OCT-02 09.10.14.139452
11	27	10-OCT-02 08.41.54.538124	10-OCT-02 09.21.41.406147
6	28	10-OCT-02 10.00.01.006157	10-OCT-02 11.03.04.040482
8	29	10-OCT-02 10.11.00.599131	10-OCT-02 10.58.00.002148
6	30	10-OCT-02 10.43.36.360121	10-OCT-02 11.26.21.210431
11	31	10-OCT-02 10.50.14.140022	10-OCT-02 11.54.08.079978
11	32	10-OCT-02 14.17.24.237253	10-OCT-02 14.57.06.057274
2	33	10-OCT-02 18.01.43.430505	10-OCT-02 19.00.31.308040
9	34	10-OCT-02 18.03.06.055241	10-OCT-02 18.49.20.199483
11	35	10-OCT-03 07.20.52.514443	10-OCT-03 08.10.32.318487
6	36	10-OCT-03 07.39.31.310202	10-OCT-03 08.24.12.118274
8	37	10-OCT-03 07.57.43.424290	10-OCT-03 08.44.32.320012
6	38	10-OCT-03 08.30.45.447845	10-OCT-03 09.11.12.121314
11	39	10-OCT-03 09.57.13.126341	10-OCT-03 10.33.09.090001
11	40	10-OCT-03 10.11.05.407000	10-OCT-03 10.48.24.238467
2	41	10-OCT-03 10.13.31.312011	10-OCT-03 11.04.09.087029
9	42	10-OCT-03 10.51.13.128752	10-OCT-03 11.44.25.251349
11	43	10-OCT-03 11.02.37.368718	10-OCT-03 11.56.45.453215
9	44	10-OCT-03 11.09.30.300029	10-OCT-03 12.04.16.150650
4	45	10-OCT-03 11.49.31.310008	10-OCT-03 12.32.24.241111
6	46	10-OCT-03 11.55.25.248242	10-OCT-03 12.57.13.126720
3	47	10-OCT-03 13.40.24.241212	10-OCT-03 14.06.18.178211
7	48	10-OCT-03 14.33.01.010101	10-OCT-03 15.38.37.370593
7	49	10-OCT-03 15.01.46.463636	10-OCT-03 15.56.23.228214
4	50	10-OCT-03 15.19.59.580001	10-OCT-03 16.04.08.083512
1	51	10-OCT-03 15.38.14.143215	10-OCT-03 16.14.30.298201
3	52	10-OCT-03 15.47.33.325214	10-OCT-03 16.17.21.212121
7	53	10-OCT-04 07.33.47.467512	10-OCT-04 08.18.02.017842
4	54	10-OCT-04 08.02.51.510515	10-OCT-04 08.52.13.130647
10	55	10-OCT-04 09.11.55.548625	10-OCT-04 10.01.31.309750
9	56	10-OCT-04 10.33.39.386512	10-OCT-04 10.57.01.010000
3	57	10-OCT-04 11.17.03.029898	10-OCT-04 11.41.10.097304
7	58	10-OCT-04 11.38.01.010101	10-OCT-04 12.14.23.227520
9	59	10-OCT-04 12.13.12.119999	10-OCT-04 13.01.11.110912
4	60	10-OCT-04 13.04.35.347757	10-OCT-04 13.54.05.048721
6	61	10-OCT-04 13.06.36.361231	10-OCT-04 13.40.30.301867
3	62	10-OCT-04 15.45.00.597531	10-OCT-04 16.16.59.587261
9	63	10-OCT-04 18.10.11.110120	10-OCT-04 18.47.03.028171

Relation: Offers (FID,ItemID)

FID	ItemID
11	1
3	2
3	3
11	4
3	5
5	6
3	7
11	8
11	9
7	10
9	11
10	12
9	13
10	14
10	15
11	16
9	17
3	18
8	3
3	7
5	13
5	19
1	20
1	21
4	22
1	23
4	10
10	24
10	2
3	15
7	25
7	26
7	27
7	28
4	29
1	30
3	31
7	18
4	32
10	33
9	34
3	35
7	36
9	37
4	22
6	5
3	38
9	39
10	40
10	41
3	42
6	43
7	16
1	3
2	44
8	45
7	46
1	47
2	48
11	49
9	50
4	37
5	51

Relation: Contains (Container, CmboNum, ItemID)

OrderNumber	ComboID	ItemID
1	1	47
2	2	150
3	3	158
4	4	142
5	5	39
6	6	33
7	7	51
8	8	271
9	8	278
10	8	277
11	8	268
12	8	270
13	9	30
14	10	271
15	10	276
16	10	280
17	10	270
18	11	273
19	11	275
20	11	279
21	11	268
22	12	272
23	12	274
24	12	281
25	12	269
26	13	271
27	13	277
28	13	270
29	14	283
30	14	271
31	14	268
32	15	281
33	16	278
34	16	277
35	16	271
36	16	270
37	16	268
38	17	271
39	17	269
40	18	281
41	18	280
42	18	268
43	19	271
44	19	270
45	20	271
46	20	270
47	21	272
48	21	267
49	22	276
50	22	268
51	23	279
52	23	270
53	24	277
54	24	269
55	25	282
56	25	267
57	26	281
58	26	269
59	27	283
60	27	281
61	27	275
62	27	268
63	27	270

Relation: Includes (OrderNumber, ItemID, FID, Quantity, oPrice)

OrderNumber	ItemID	FID	Quantity	oPrice
11	1	4	1	2.49
3	2	1	2	3.99
3	3	3	1	2.99
11	4	7	1	4.99
3	5	4	2	6.49
5	6	10	1	5.49
3	7	9	2	4.79
11	8	3	3	4.39
11	9	7	1	1.99
7	10	9	4	2.49
9	11	4	1	3.99
10	12	6	2	2.99
9	13	3	1	4.99
10	14	9	1	6.49
10	15	10	1	5.49
11	16	10	1	4.79
9	17	3	2	4.39
3	18	6	4	1.99
8	3	7	4	2.49
3	7	1	1	3.99
5	13	2	1	2.99
5	19	8	1	4.99
1	20	7	1	6.49
1	21	1	1	5.49
4	22	2	1	4.79
1	23	11	2	4.39
4	10	9	2	1.99
10	24	4	1	2.49
10	2	5	2	1.49
3	15	4	1	1.79
7	25	1	1	1.99
7	26	3	1	1.39
7	27	11	2	1.69
7	28	3	2	1.89
4	29	3	1	2.49
1	30	11	2	3.99
3	31	3	1	2.99
7	18	5	2	4.99
4	32	3	2	6.49
10	33	11	2	5.49
9	34	11	1	4.79
3	35	7	1	4.39
7	36	9	1	1.99
9	37	10	1	1.49
4	22	9	1	1.79
6	5	10	1	1.99
3	38	10	1	1.39
9	39	11	3	1.69
10	40	9	5	1.89
10	41	3	1	2.49
3	42	8	4	3.99
6	43	3	1	2.99
7	16	5	4	4.99
1	3	5	5	6.49
2	44	1	2	5.49
8	45	1	2	4.79
7	46	4	1	4.39
1	47	1	1	1.99
2	48	4	1	1.49
11	49	10	1	1.79
9	50	10	4	1.99
4	37	3	1	1.39
5	51	7	1	1.69

Part 4: Non-trivial Queries in Relational Algebra, Tuple Relational, and Domain Relational Calculus Form

In this part of the project thirty queries will be expressed in relational algebra, tuple relational calculus, and domain relational calculus, ten for each formal language . Each query will be stated prior to being expressed in each section's respective format.

4.1 Queries in Relational Algebra

Special operators:

π (projection)	σ (selection)	
$\wedge$ (AND)	$\vee$ (OR)	$\cup$ (Set Union)
$\times$ (Cross Product)	$\div$ (Set Division)	
$\leftarrow$ (Relation Rename)	ρ (Formal Rename)	
$\bowtie$ (Natural Join)	$\ltimes_{\theta}$ (Left Semijoin)	
$\Im$ (Aggregate Functions)		

Query 1: List all restaurants which offer combo meals priced $3.99 or less.

$$\pi_{\text{I.RstID, I.rName}}(\sigma_{\text{C.ComboID} = \text{I.ItemID} \wedge \text{I.Price} \leq 3.99}((\text{Contains})^{\text{C}} \times (\text{Restaurant} \bowtie \text{Franchise} \bowtie \text{Offers} \bowtie \text{Item})^{\text{I}}))$$

Query 2: List customers who've placed at least two orders priced greater than $20 total.

$$\rho_{C_1}(\text{cID, oID, dCharge, iSum})\ {}_{\text{CustomerID, OrderNumber, DeliveryCharge}}\Im_{\text{SUM oPrice}}(\text{Includes} \bowtie \text{Order})$$

$$\rho_{C_2}(\text{cID, oID, dCharge, iSum})\ {}_{\text{CustomerID, OrderNumber, DeliveryCharge}}\Im_{\text{SUM oPrice}}(\text{Includes} \bowtie \text{Order})$$

$$\text{Customer} \ltimes_{\text{Customer.CustomerID} = C_1\text{.cID}}(\sigma_{\substack{C_1\text{.cID} = C_2\text{.cID} \wedge C_1\text{.oID} \neq C_2\text{.oID} \wedge \\ (C_1\text{.dCharge} + C_1\text{.iSum}) > 20 \wedge \\ (C_2\text{.dCharge} + C_2\text{.iSum}) > 20}}(C_1 \times C_2))$$

Query 3: List franchises from which more than ten customers have ordered.

$$\rho_{\text{FC}}(\text{FID, cCount})\ {}_{\text{FID}}\Im_{\text{COUNT CustomerID}}(\text{Franchise} \bowtie \text{Includes} \bowtie \text{Order})$$

$$\text{Franchise} \ltimes_{\text{Franchise.FID} = \text{FC.FID}}(\sigma_{\text{cCount} > 10}(\text{FC}))$$

Query 4: List customers who average the smallest number of items per order.

$$\rho_{oAVG}\,(\text{OrderNumber, aIQ})\ {}_{\text{OrderNumber}}\Im_{\text{AVG Quantity}}\,(\text{Includes})$$

$$\text{coAVG} \leftarrow \pi_{\text{oAVG.OrderNumber, Order.CustomerID, oAVG.aIQ}}\,(\text{Order} \bowtie \text{oAVG})$$

$$\rho_{\text{ciAVG}}\,(\text{CustomerID, cIA})\ {}_{\text{CustomerID}}\Im_{\text{AVG Quantity}}\,(\text{coAVG})$$

$$\pi_{\text{ciAVG.CustomerID}}\,(\text{ciAVG} - \pi_{C_1.\text{CustomerID}, C_1.\text{cIA}}\,(\sigma_{C_1.\text{cIA} > C_2.\text{cIA}}\,(\text{ciAVG}^{C_1} \times \text{ciAVG}^{C_2})) \bowtie \text{Customer}$$

Query 5: List customers who've been members between 5/1/2010, and 6/25/2010.

$$\text{tdRange} \leftarrow \sigma_{\substack{\text{rDate} > \text{'10-MAY-01 00.00.00.000000'} \wedge\ \text{rDate} < \text{'10-JUN-25 00.00.00.000000'} \wedge \\ (\text{SellMembership} = 1 \vee \text{SellMembership} = 2)}}\,(\text{Registers})$$

$$\text{tdRange2} \leftarrow \pi_{\substack{C_1.\text{EmployeeID}, C_1.\text{CustomerID}, C_1.\text{rDate}, \\ C_1.\text{SellMembership}}}\,(\sigma_{\substack{C_1.\text{rDate} \le \text{'10-MAY-01 00.00.00.000000'} \wedge \\ C_1.\text{SellMembership} = 2 \wedge \\ ((C_2.\text{rDate} > \text{'10-MAY-01 00.00.00.000000'} \vee \\ C_2.\text{rDate} \le C_1.\text{rDate}) \vee \\ C_2.\text{SellMembership} \ne 1) \wedge \\ C_1.\text{CustomerID} = C_2.\text{CustomerID}}}\,(\text{Registers}^{C_1} \times \text{Registers}^{C_2}))$$

$$\text{Customer} \bowtie_{\text{Customer.CustomerID} = \text{cFin.CustomerID}} (\text{tdRange} \bowtie \text{tdRange2})$$

Query 6: List customers who've never placed an order from a Taco Bell in the 93308 zip code area.

$$\text{cTB} \leftarrow \pi_{\text{CustomerID}}(\sigma_{\text{rName} = \text{'Taco Bell'} \wedge \text{fZip} = 93308}\,(\text{Restaurant} \bowtie \text{Franchise} \bowtie \text{Includes} \bowtie \text{Order}))$$

$$\text{Customer} - (\text{Customer} \bowtie \text{cTB})$$

Query 7: List franchises with the second least expensive combos.

$$\rho_{(\text{ItemID, subItem})}\,(\text{Contains})$$

$$\text{Combo} \leftarrow \text{Item} \bowtie \pi_{\text{ComboID}}\,(\text{Contains})$$

$$\text{ABL} \leftarrow \pi_{C1.\text{Price}}\,(\sigma_{C1.\text{Price} > C2.\text{Price}}\,(\text{Combo}_{C_1} \times \text{Combo}_{C_2}))$$

$$\text{AB2L} \leftarrow \pi_{A2.\text{Price}}\,(\sigma_{A1.\text{Price} > A2.\text{Price}}\,(\text{ABL}_{A_1} \times \text{ABL}_{A_2}))$$

$$\text{Franchise} \bowtie \text{Offers} \bowtie \text{Combo} \bowtie (\text{ABL} - \text{AB2L})$$

Query 8: List employees who delivered items from each franchise location in the 93312 zip code area on 7/4/2010.

$$\text{ID4d} \leftarrow \text{Includes} \bowtie \pi_{\text{EmployeeID}}\,(\sigma_{\substack{\text{DeliveryTime} < \text{'10-JUL-05 00.00.00.000000'} \\ \text{DeliveryTime} \geq \text{'10-JUL-04 00.00.00.000000'}} \wedge (\text{Delivers}))$$

$$\text{Employee} \bowtie \pi_{\text{EmployeeID}}\,(\text{ID4d} \div \pi_{\text{FID}}\,(\sigma_{\text{fZip} = 93312}\,(\text{Franchise})))$$

Query 9: List employees who've not sold memberships, or made deliveries to any customers.

$$\text{eD} \leftarrow \text{Employee} \bowtie \pi_{\text{EmployeeID}}\,(\sigma_{\text{DeliveryTime} \neq \text{NULL}}\,(\text{Delivers}))$$

$$\text{eR} \leftarrow \text{Employee} \bowtie \pi_{\text{EmployeeID}}\,(\sigma_{\text{SellMembership} = 2}\,(\text{Registers}))$$

$$\text{Employee} - (\text{eD} \cup \text{eR})$$

Query 10: List employees for whom it's taken more than 1.5 hours to deliver an order.

$$\text{Employee} \bowtie \pi_{\text{EmployeeID}}\,(\sigma_{(\text{DeliveryTime} - \text{DOReceiptTime})\ \text{Hours} > 1.5 \wedge \text{DeliveryTime} \neq \text{NULL}}\,(\text{Delivers}))$$

4.2 Queries in Tuple Relational Calculus

Special operators: $\wedge$ **(AND)** $\vee$ **(OR)**
$\neg$ **(Negation)** $\rightarrow$ **(Implication)**
$\exists$ **(Existential Quantifier)** $\forall$ **(Universal Quantifier)**

Query 1: List all restaurants which offer combo meals priced $3.99 or less.

$$\{ r \mid Restaurant(r) \wedge (\exists f)(Franchise(f) \wedge f.RstID = r.RstID \wedge (\exists i)(\exists c)(Contains(c) \wedge Item(i) \wedge c.ComboID = i.ItemID \wedge i.Price \leq 3.99 \wedge (\exists o)(Offers(o) \wedge o.FID = f.FID \wedge o.ItemID = i.ItemID)))\}$$

Query 2: List customers who've placed at least two orders with a delivery charge of more than twenty dollars.

$$\{ c \mid Customer(c) \wedge (\exists o_1)(Order(o_1) \wedge o_1.DeliveryCharge > 20.00 \wedge o_1.CustomerID = c.CustomerID \wedge (\exists o_2)(Order(o_2) \wedge o_2.DeliveryCharge > 20.00 \wedge o_2.OrderNumber \neq o_1.OrderNumber \wedge o_2.CustomerID = c.CustomerID))\}$$

Query 3: List restaurants from which every customer has ordered.

$$\{ r \mid Restaurant(r) \wedge (\forall c)(Customer(c) \rightarrow (\exists o)(Order(o) \wedge c.CustomerID = o.CustomerID \wedge (\exists iN)(Includes(iN) \wedge iN.OrderNumber = o.OrderNumber \wedge (\exists f)(Franchise(f) \wedge f.FID = iN.FID \wedge f.RstID = r.RstID)))))\}$$

Query 4: List customers who were registered by Peter Patrone.

$$
\begin{aligned}
&\{ c \mid \text{Customer}(c) \wedge (\exists e)(\text{Employee}(e) \wedge e.\text{eFirst} = \text{'Peter'} \wedge e.\text{eLast} = \text{'Patrone'} \wedge \\
&\qquad (\exists r)(\text{Registers}(r) \wedge r.\text{EmployeeID} = e.\text{EmployeeID} \wedge \\
&\qquad\quad r.\text{CustomerID} = c.\text{CustomerID} \\
&\qquad\quad) \\
&\qquad) \\
&\}
\end{aligned}
$$

Query 5: List customers who've been members between 5/1/2010, and 6/25/2010.

$$
\begin{aligned}
&\{ c \mid \text{Customer}(c) \wedge (\exists r)(\text{Registers}(r) \wedge r.\text{rDate} > \text{'5/1/2010'} \wedge r.\text{rDate} < \text{'6/25/2010'} \wedge \\
&\qquad r.\text{CustomerID} = c.\text{CustomerID} \wedge \\
&\qquad (r.\text{SellMembership} = 1 \vee r.\text{SellMembership} = 2) \\
&\qquad) \vee \\
&\quad (\exists r_2)(\text{Registers}(r_2) \wedge r_2.\text{rDate} \leq \text{'5/1/2010'} \wedge \\
&\qquad r_2.\text{CustomerID} = c.\text{CustomerID} \wedge r_2.\text{SellMembership} = 2 \wedge \\
&\qquad (\exists r_3)(\text{Registers}(r_3) \wedge r_3.\text{rDate} \leq \text{'5/1/2010'} \wedge r_2.\text{rDate} < r_3.\text{rDate} \wedge \\
&\qquad\quad r_3.\text{CustomerID} = c.\text{CustomerID} \rightarrow r_3.\text{SellMembership} \neq 1 \\
&\qquad\quad) \\
&\qquad) \\
&\}
\end{aligned}
$$

Query 6: List customers who've never placed an order from a Taco Bell in the 93308 zip code area.

$$
\begin{aligned}
&\{ c \mid \text{Customer}(c) \wedge \neg(\exists c_2)(\text{Customer}(c_2) \wedge \\
&\qquad (\exists o)(\text{Order}(o) \wedge o.\text{CustomerID} = c_2.\text{CustomerID} \wedge \\
&\qquad\quad (\exists i)(\text{Includes}(i) \wedge i.\text{OrderNumber} = o.\text{OrderNumber} \wedge \\
&\qquad\qquad (\exists f)(\text{Franchise}(f) \wedge f.\text{fZip} = 93308 \wedge f.\text{FID} = i.\text{FID} \wedge \\
&\qquad\qquad\quad (\exists r)(\text{Restaurant}(r) \wedge r.\text{RstID} = f.\text{RstID} \wedge \\
&\qquad\qquad\qquad r.\text{rName} = \text{'Taco Bell'} \wedge \\
&\qquad\qquad\qquad c.\text{CustomerID} = c_2.\text{CustomerID} \\
&\qquad\qquad\qquad) \\
&\qquad\qquad\quad) \\
&\qquad\qquad) \\
&\qquad\quad) \\
&\qquad) \\
&\}
\end{aligned}
$$

Query 7: List franchises with the second least expensive combos.

$$
\begin{aligned}
&\{ f \mid \text{Franchise}(f) \wedge (\exists c)(\text{Contains}(c) \wedge \\
&\quad (\exists i)(\text{Item}(i) \wedge c.\text{ComboID} = i.\text{ItemID} \wedge \\
&\quad\quad (\exists c_2)(\text{Contains}(c_2) \wedge \\
&\quad\quad\quad (\exists i_2)(\text{Item}(i_2) \wedge i_2.\text{ItemID} = c_2.\text{ComboID} \wedge \\
&\quad\quad\quad\quad i_2.\text{Price} < i.\text{Price} \wedge \\
&\quad\quad\quad\quad\quad \neg(\exists c_3)(\text{Contains}(c_3) \wedge \\
&\quad\quad\quad\quad\quad (\exists i_3)(\text{Item}(i_3) \wedge i_3.\text{ItemID} = c_2.\text{ComboID} \wedge \\
&\quad\quad\quad\quad\quad\quad i_3.\text{Price} < i.\text{Price} \wedge i_3.\text{Price} \neq i_2.\text{Price} \\
&\quad\quad\quad\quad\quad\quad) \\
&\quad\quad\quad\quad\quad) \wedge \\
&\quad\quad\quad\quad (\exists o)(\text{Offers}(o) \wedge o.\text{ItemID} = i.\text{ItemID} \wedge o.\text{FID} = f.\text{FID} \\
&\quad\quad\quad\quad\quad) \\
&\quad\quad\quad\quad) \\
&\quad\quad\quad) \\
&\quad\quad) \\
&\quad) \\
&\}
\end{aligned}
$$

Query 8: List employees who delivered items from each franchise location in the 93312 zip code area on 7/4/2010.

$$
\begin{aligned}
&\{ e \mid \text{Employee}(e) \wedge (\forall f)(\text{Franchise}(f) \rightarrow \\
&\quad (\exists i)(\text{Includes}(i) \wedge i.\text{FID} = f.\text{FID} \wedge f.\text{fZip} = 93312 \wedge \\
&\quad\quad (\exists o)(\text{Order}(o) \wedge o.\text{OrderNumber} = i.\text{OrderNumber} \wedge \\
&\quad\quad\quad (\exists d)(\text{Delivers}(d) \wedge d.\text{OrderNumber} = o.\text{OrderNumber} \wedge \\
&\quad\quad\quad\quad d.\text{EmployeeID} = e.\text{EmployeeID} \wedge \\
&\quad\quad\quad\quad d.\text{DeliveryTime} < \text{'7/5/2010'} \wedge \\
&\quad\quad\quad\quad d.\text{DeliveryTime} > \text{'7/3/2010'} \\
&\quad\quad\quad\quad) \\
&\quad\quad\quad) \\
&\quad\quad) \\
&\quad) \\
&\}
\end{aligned}
$$

Query 9: List employees who've not sold memberships, or made deliveries to any customers.

$$\{ e \mid \text{Employee}(e) \wedge \neg((\exists d)(\text{Delivers}(d) \wedge d.\text{DeliveryTime} \neq \text{NULL} \wedge d.\text{EmployeeID} = e.\text{EmployeeID}) \vee (\exists r)(\text{Registers}(r) \wedge r.\text{SellMembership} = 2 \wedge r.\text{EmployeeID} = e.\text{EmployeeID}))\}$$

Query 10: List employees for whom it's taken more than 1.5 hours to deliver an order.

$$\{ e \mid \text{Employee}(e) \wedge (\exists d)(\text{Delivers}(d) \wedge (d.\text{DeliveryTime} - d.\text{DOReceiptTime}) > 1.5 \text{ hours} \wedge d.\text{DeliveryTime} \neq \text{NULL} \wedge d.\text{EmployeeID} = e.\text{EmployeeID})\}$$

4.3 Queries in Domain Relational Calculus

Special operators:				
	$\wedge$	**(AND)**	$\vee$	**(OR)**
	$\neg$	**(Negation)**	$\rightarrow$	**(Implication)**
	$\exists$	**(Existential Quantifier)**	$\forall$	**(Universal Quantifier)**

Query 1: List all restaurants which offer combo meals priced $3.99 or less.

$$\{ rI, rN \mid \text{Restaurant}(rI, rN) \wedge (\exists f)(\text{Franchise}(f, \_, \_, \_, \_, \_, rI) \wedge (\exists c)(\text{Contains}(c, \_) \wedge \text{Item}(c, \_, \leq 3.99, \_) \wedge \text{Offers}(f, c)))\}$$

Query 2: List customers who've placed at least two orders with a delivery charge of more than twenty dollars.

$$\{ c, v, w, x, y, z \mid \text{Customer}(c, v, w, x, y, z) \wedge (\exists o)(\text{Order}(o, \_, \_, >20, \_, \_, \_, c) \wedge (\exists o_2)(\text{Order}(o_2, \_, \_, >20, \_, \_, \_, c) \wedge o \neq o_2))\}$$

Query 3: List restaurants from which every customer has ordered.

$$\{ r, n \mid \text{Restaurant}(r, n) \wedge (\forall c)(\text{Customer}(c, \_, \_, \_, \_, \_) \rightarrow (\exists o)(\text{Order}(o, \_, \_, \_, \_, \_, \_, c) \wedge (\exists f)(\text{Includes}(o, \_, f, \_, \_) \wedge \text{Franchise}(f, \_, \_, \_, \_, r)))) \}$$

Query 4: List customers who were registered by Peter Patrone.

$$\{ c, v, w, x, y, z \mid \text{Customer}(c, v, w, x, y, z) \wedge (\exists e)(\text{Employee}(e, \text{'Peter'}, \_, \text{'Patrone'}) \wedge (\text{Registers}(e, c, \_, \_))) \}$$

Query 5: List customers who were members anytime between 5/1/2010, and 6/25/2010.

$$\{ c, v, w, x, y, z \mid \text{Customer}(c, v, w, x, y, z) \wedge (\text{Registers}(\_, c, > \text{'5/1/2010'} \wedge < \text{'6/25/2010'}, \_, 2 \vee 1) \vee ((\exists d_1)(\text{Registers}(\_, c, d_1, 2) \wedge (\exists d_2)(\text{Registers}(\_, c, d_2, 1) \wedge d_1 \leq \text{'5/1/2010'} \wedge d_2 < d_1)))) \}$$

Query 6: List customers who've never placed an order from a Taco Bell in the 93308 zip code area.

$$\{ c, v, w, x, y, z \mid \text{Customer}(c, v, w, x, y, z) \wedge \neg(\exists o)(\text{Order}(o, \_, \_, \_, \_, \_, \_, c) \wedge (\exists f)(\text{Includes}(o, \_, f, \_, \_) \wedge (\exists r)(\text{Franchise}(f, \_, \_, 93312, r) \wedge \text{Restaurant}(r, \text{'Taco Bell'}))))) \}$$

Query 7: List franchises with the second least expensive combos.

$$\{ f, v, w, x, y, z \mid \text{Franchise}(f, v, w, x, y, z) \wedge (\exists c)\,(\exists p)(\text{Contains}(c, \_) \wedge \text{Item}(c, \_, p, \_) \wedge (\exists p_1)(\text{Item}(\_, \_, p_1, \_) \wedge p_1 < p \wedge \neg(\exists p_2)(\text{Item}(\_, \_, p_2, \_) \wedge p_2 < p \wedge p_2 \neq p_1)))\}$$

Query 8: List employees who delivered items from each franchise location in the 93312 zip code area on 7/4/2010.

$$\{ e, x, y, z \mid \text{Employee}(e, x, y, z) \wedge (\forall f)(\text{Franchise}(f, \_, \_, 93312, \_, \_) \rightarrow (\exists o)(\text{Includes}(o, \_, f, \_, \_) \wedge \text{Delivers}(e, o, \_, >'7/3/2010' \wedge <'7/5/2010')))\}$$

Query 9: List employees who've not sold memberships, or made deliveries to any customers.

$$\{ e, x, y, z \mid \text{Employee}(e, x, y, z) \wedge \neg(\exists e_1)((\text{Delivers}(e_1, \_, \_, \neq \text{NULL}) \vee \text{Registers}(e_1, \_, \_, 2)) \wedge e = e_1)\}$$

Query 10: List employees for whom it's taken more than 1.5 hours to deliver an order.

$$\{ e, x, y, z \mid \text{Employee}(e, x, y, z) \wedge (\exists s_1)(\exists s_2)(\text{Delivers}(e, \_, s_1, s_2) \wedge s_2 \neq \text{NULL} \wedge (s_1 - s_2) > 1.5 \text{ hrs})\}$$

Phase III

Logical and Physical Database Creation
With the Oracle DBMS

Part 1: The SQL*Plus Command Utility

SQL*Plus is a command line interface tool used to access data stored in an Oracle database. The application originates from a prior Oracle utility called UFI, short for "User friendly utility." Because UFI was not as robust as many programmers felt it needed to be, new features were added to the UFI program to the point where it was made quite advanced, and Oracle decided to change the name to SQL*Plus.

The great thing about this application is that it enables users to do all the things necessary to create a full-featured Database in Oracle 10g. Not only does it run SQL, and PL/SQL commands, but also operating system commands which allow users the ability to format, perform calculations on, store, and print from query results. With it, one can examine table objects, and definitions, and output query results to a text file for later review. Even though today there are more graphically involved user interfaces that allow people to perform similar functions, SQL*Plus still serves an important need for people interested in quick database access, maintenance, and manipulation.

Part 2: Schema Objects

A schema object is a collection of logical structures of data. Such objects are stored logically in tablespaces of the database, where the data of each object is physically stored, contained in datafiles which can span multiple physical locations.

Common schema objects in Oracle include tables, views, indexes, procedures, functions, database links, clusters, triggers, dimensions, sequences, and synonyms. The following is a description of each:

2.1 Table

A form of table has already been presented in this project through the relation, however, not all tables can be a form of relation. Like relations tables have columns, and rows used to manipulate, and organize data. However, unlike relations SQL tables can have nonunique repeating row values. Otherwise, for the most part tables are like relations in that they contain columns which act much like attributes, and records are stored in rows much like tuples in a relation.

2.2 Views

Views are essentially virtual tables. Like tables, views display data logically arranged in terms of rows, and columns, but views don't actually store information like tables. In fact, the only thing stored in a view is a query used to organize together information from actual tables to form a table-like structure that only shows what other tables have already stored. Views provide a great benefit in that they are more secure than tables, because they are read only. They can also display aggregate function results useful in performing day-to-day database operations.

2.3 Indexes

An index is a copy of part of a table. It's used to improve data lookup speed by not having to bring up the entire contents of a table to retrieve wanted information. In being small, they also take up significantly less space than a table. This allows them to be stored in main memory giving reason the significant improvement in lookup time. Still, even though they take up less space, they're still more space than the table alone. Also , because it's not just the table that information is written to (the index is written to as well), longer write times are required. Indexes are definitely a tradeoff considered by most people.

2.4 Procedures

Procedures are subroutines accessed by relational database applications. They're useful, because they help setup control mechanisms through which many statements for actions to be performed on the database can be made. There are many useful function-type features built into stored procedures. Not only are stored procedures good to use for data validation, but also for limiting user control access to the database for safety reasons.

2.5 Functions

Functions are pretty much the same as stored procedures except that functions be used like any other SQL expression, whereas stored procedures must be instantiated using the CALL statement. Functions also commonly use a return statement to return values.

2.6 Database Links

Database links are schema objects that enable one to access objects in another database. With a database link schema object one can even connect with database that isn't necessarily an Oracle database either. Once a database link is established all local database operations can be employed just as well on the other link database. All standard options with SQL in terms of insertions, deletions, update, etc. can be performed.

2.7 Clusters

Clusters are an optional method of data organization used in contextual optimizations that require having commonly accessed sets of table data to be stored contiguously.

2.8 Triggers

Triggers are code programmed to set off and run a procedure the moment a specified event happens. Mostly triggers are used to ensure data integrity so that if say one were to insert something in one relation, the trigger would go off alerting that business constraints require that other relation fields somewhere else must also be modified.

2.9 Dimensions

Dimensions are objects that help to categorize the properties of a business, and it's methods so as to answer important question people in their business may have regarding their own methods of running the business. An example given by Oracle on what a dimension would be good for is a retail chain management building a data warehouse to analyze sales trends of products across all of their stores. This is setup through hierarchies in columns of a table representing different levels of an organization.

2.10 Sequences

An easy way to setup autoincrementing of values for specified attributes as records are entered is through sequences. These are mostly used in primary key fields due to the requirement for such fields to be unique.

2.11 Synonyms

Synonyms are aliases used to reference tables, and table attributes for the sake of allowing users a simpler way to reference the object to which the synonym refers. It also makes it more difficult for malicious programs to attack the named object.

Part 3: Relation Schema

The main schema objects used in this project are table relations, though there is some indexing, mostly through use of indexing of primary keys. This section is essentially a snapshot of the data residing in the database.

Relation Schema Instances

HV_EMPLOYEE

```
CS342 SQL> DESC HV_EMPLOYEE

 Name              Null?    Type
 ----------------- -------- ------------
 EMPLOYEEID        NOT NULL NUMBER(8)
 EFIRST            NOT NULL VARCHAR2(20)
 EMIDINITIAL                CHAR(1)
 ELAST             NOT NULL VARCHAR2(40)

CS342 SQL> SELECT * FROM HV_EMPLOYEE;

EMPLOYEEID EFIRST               E ELAST
---------- -------------------- - ----------------------------------------
         1 Stanley              L Marsh
         2 Brazuk                 Pierce
         3 Marsha                 Bentley
         4 Fred                 S Armstrong
         5 Marisol              M Stensos
         6 Larry                  Guytez
         7 Peter                P Patrone
         8 Curtis               B Whicks
         9 Wei                  H Kai
        10 Tiffany                Cooper
        11 Allison                Jester

11 rows selected.
```

HV_MILEAGELOG

```
CS342 SQL> DESC HV_MILEAGELOG
 Name                    Null?    Type
 ----------------------- -------- ---------------
 EMPLOYEEID              NOT NULL NUMBER(8)
 SDATE                   NOT NULL TIMESTAMP(0)
 EDATE                            TIMESTAMP(0)
 STARTMILEAGE            NOT NULL NUMBER(6)
 ENDMILEAGE                       NUMBER(6)

CS342 SQL> SELECT * FROM HV_MILEAGELOG;

EMPLOYEEID SDATE                     EDATE                     STARTMILEAGE ENDMILEAGE
---------- ------------------------- ------------------------- ------------ ----------
         2 02-JUL-10 07.50.00 AM     02-JUL-10 02.10.00 PM            94032      94073
         4 02-JUL-10 09.35.00 AM     02-JUL-10 05.52.00 PM           119003     119041
         6 03-JUL-10 07.13.00 AM     03-JUL-10 05.21.00 PM            56213      56233
         4 03-JUL-10 09.09.00 AM     03-JUL-10 01.54.00 PM           119053     119118
         8 03-JUL-10 05.48.00 PM     03-JUL-10 08.31.00 PM            81423      81463
         1 04-JUL-10 08.21.00 AM     04-JUL-10 01.15.00 PM            76324      76397
        10 04-JUL-10 01.04.00 PM     04-JUL-10 06.45.00 PM           155221     155318
         9 04-JUL-10 04.33.00 PM     04-JUL-10 07.18.00 PM           121351     121396
        11 05-JUL-10 07.07.00 AM                                     147214
         9 05-JUL-10 10.15.00 AM                                     121384
         7 05-JUL-10 11.10.00 AM                                      61352

11 rows selected.
```

HV_CUSTOMER

```
CS342 SQL> DESC HV_CUSTOMER
 Name              Null?    Type
 ----------------- -------- ------------
 CUSTOMERID        NOT NULL NUMBER(8)
 CFIRST            NOT NULL VARCHAR2(20)
 CMIDINITIAL                CHAR(1)
 CLAST             NOT NULL VARCHAR2(40)
 PHONE             NOT NULL NUMBER(10)
 MEMBERSTATUS      NOT NULL NUMBER(1)

CS342 SQL> SELECT * FROM HV_CUSTOMER;

CUSTOMERID CFIRST   C CLAST             PHONE MEMBERSTATUS
---------- -------- - ------------ ---------- ------------
         1 Polly    G Fredericks   6613314574            0
         2 Rachel     Larson       6615893212            0
         3 Henry    K Ricks        6617589212            1
         4 Saul     M Steinbeck    6612059354            1
         5 Daniel     Simpson      8053652241            1
         6 Matt       Groening     6618614124            0
         7 Karen    F Chu          8057584790            1
         8 Ryan     L Scott        8057583787            0
         9 Jeff       Green        6615894512            1
        10 Manmeet    Chunta       6613584124            0
        11 Greg     A Ford         8059683641            0

11 rows selected.
```

HV_CSTMRADDRESS

```
CS342 SQL> DESC HV_CSTMRADDRESS
 Name              Null?    Type
 ----------------- -------- ------------
 ADDRESSID         NOT NULL NUMBER(8)
 STREET            NOT NULL VARCHAR2(60)
 APT                        VARCHAR2(10)
 CITY              NOT NULL VARCHAR2(20)
 ZIP               NOT NULL NUMBER(5)
 CUSTOMERID        NOT NULL NUMBER(8)

CS342 SQL> SELECT * FROM HV_CSTMRADDRESS;

 ADDRESSID STREET                   APT        CITY               ZIP CUSTOMERID
---------- ------------------------ ---------- ----------- ---------- ----------
         1 11421 Old Town Rd        A          Bakersfield      93312          1
         2 118 Harvest Creek Rd                Bakersfield      93311          2
         3 Elk Hill Rd              Gate 3     Taft             93268          2
         4 253 Alki Ct                         Bakersfield      93314          3
         5 12321 Dorsey Ct          3C         Bakersfield      93311          4
         6 1775 Balvanera Ave                  Lamont           93241          5
         7 2321 Hialeah Park Ln     319        Bakersfield      93309          6
         8 22314 Verdelho Ave                  Bakersfield      93312          7
         9 10017 Great Country Dr              Bakersfield      93306          8
        10 905 Mayacamas Dr                    Taft             93268          8
        11 3421 Allene Way          G          Bakersfield      93383          9
        12 5843 Red River Dr                   Lamont           93241          4
        13 342 Treasure Island St              Bakersfield      93308         10
        14 44321 Abbott Dr          21         Bakersfield      93308          5
        15 423 Beyers St                       Lamont           93241          3
        16 784 Tanner Michael Dr               Bakersfield      93312         11
        17 321 Skye Dr              B          Bakersfield      93309          4
        18 562 Linda Vista Dr                  Bakersfield      93308          8
        19 119 Ray St                          Bakersfield      93313          2

19 rows selected.
```

HV_ORDER

```
CS342 SQL> DESC HV_ORDER
 Name              Null?    Type
 ----------------- -------- ------------
 ORDERNUMBER       NOT NULL NUMBER(8)
 ODATE             NOT NULL TIMESTAMP(0)
 ORDERSTATUS       NOT NULL NUMBER(1)
 DELIVERYCHARGE             NUMBER(38,2)
 OMEMBERSTATUS     NOT NULL NUMBER(1)
 DESTINATION       NOT NULL NUMBER(8)
 EMPLOYEEID        NOT NULL NUMBER(8)
 CUSTOMERID        NOT NULL NUMBER(8)
```

```
CS342 SQL> SELECT * FROM HV_ORDER;

ORDERNUMBER ODATE                 ORDERSTATUS DELIVERYCHARGE OMEMBERSTATUS DESTINATION EMPLOYEEID CUSTOMERID
----------- --------------------- ----------- -------------- ------------- ----------- ---------- ----------
          1 02-JUL-10 07.50.00 AM           2             14             0           1          1          1
          2 02-JUL-10 09.30.00 AM           2              7             0           2          3          2
          3 02-JUL-10 10.45.00 AM           2             10             2           4          1          3
          4 02-JUL-10 01.20.00 PM           2              5             2           5          1          4
          5 02-JUL-10 02.24.00 PM           2              5             2           6          1          5
          6 02-JUL-10 04.09.00 PM           2             14             0           7          3          6
          7 03-JUL-10 07.05.00 AM           2              5             2           8          7          7
          8 03-JUL-10 08.58.00 AM           2              7             0           9          5          8
          9 03-JUL-10 11.21.00 AM           2             21             0          10          5          8
         10 03-JUL-10 03.11.00 PM           2              5             2          11          7          9
         11 03-JUL-10 05.45.00 PM           2              5             2          12          2          4
         12 03-JUL-10 07.13.00 PM           2              5             2          13          2         10
         13 04-JUL-10 08.10.00 AM           2              5             2          14          2          5
         14 04-JUL-10 08.47.00 AM           2              5             2          15         11          3
         15 04-JUL-10 10.05.00 AM           2             14             0          16         11         11
         16 04-JUL-10 12.05.00 PM           2              5             2          17          2          4
         17 04-JUL-10 01.01.00 PM           2              7             0          18          4          8
         18 04-JUL-10 02.50.00 PM           2              5             2          19         11          2
         19 04-JUL-10 04.22.00 PM           2             14             0           1          2          1
         20 04-JUL-10 05.05.00 PM           2             20             2           8          4          7
         21 04-JUL-10 06.04.00 PM           0              0             2           4          4          3
         22 05-JUL-10 07.07.00 AM           2              5             2          16          9         11
         23 05-JUL-10 08.17.00 AM           2              5             2          12          8          4
         24 05-JUL-10 10.00.00 AM           2              7             0          10          5          8
         25 05-JUL-10 11.05.00 AM           1             10             2          13          9         10
         26 05-JUL-10 11.07.00 AM           2              7             0           7          5          6
         27 05-JUL-10 11.20.00 AM           1              5             2          14          5          5
         28 05-JUL-10 11.24.00 AM           1             14             0           1          8          1
         29 05-JUL-10 11.35.00 AM           1              5             2          11          9          9
         30 05-JUL-10 11.39.00 AM           1             10             2           3          5          2

30 rows selected.
```

HV_RESTAURANT

```
CS342 SQL> DESC HV_RESTAURANT;
 Name              Null?    Type
 ----------------- -------- ------------
 RSTID             NOT NULL NUMBER(8)
 RNAME             NOT NULL VARCHAR2(30)

CS342 SQL> SELECT * FROM HV_RESTAURANT;
     RSTID RNAME
---------- ------------------------------
         1 Every Meal Delivery
         2 Carls Jr
         3 Starbucks Coffee
         4 Taco Bell
         5 Panda Express
         6 McDonalds
         7 Wendys
         8 Jack in the Box
         9 In and Out
        10 Burger King
        11 Del Taco
        12 Pollo Loco

12 rows selected.
```

HV_FRANCHISE

```
CS342 SQL> DESC HV_FRANCHISE
 Name              Null?    Type
 ----------------- -------- ------------
 FID               NOT NULL NUMBER(8)
 FSTREET           NOT NULL VARCHAR2(60)
 FCITY             NOT NULL VARCHAR2(20)
 FZIP              NOT NULL NUMBER(5)
 FPHONE            NOT NULL NUMBER(10)
 RSTID             NOT NULL NUMBER(8)

CS342 SQL> SELECT * FROM HV_FRANCHISE;

       FID FSTREET                           FCITY                FZIP     FPHONE      RSTID
---------- ---------------------------------- -------------- ---------- ---------- ----------
         1 9001 Stockdale Hwy                 Bakersfield         93311 6616542782          1
         2 9500 Brimhall Rd Ste A             Bakersfield         93312 6615874859          2
         3 9801 Hageman Rd                    Bakersfield         93312 6615875199          3
         4 9640 Hageman Rd.                   Bakersfield         93312 6612134574          4
         5 9200 Rosedale Highway 300          Bakersfield         93312 6615872316          5
         6 4520 Coffee Road                   Bakersfield         93312 6615879085          2
         7 9200 Rosedale Hwy                  Bakersfield         93312 6615873661          3
         8 5121 Olive Drive                   Bakersfield         93308 6613937718          4
         9 5120 Stockdale Hwy Space A         Bakersfield         93309 6613232033          5
        10 9000 Ming Ave Ste Q                Bakersfield         93311 6616652396          2
        11 4420 Coffee Road Unit B            Bakersfield         93308 6615873661          3
        12 3799 Rosedale Hwy                  Bakersfield         93308 6613253862          4
        13 5041 Gosford Rd F1                 Bakersfield         93313 6616640391          5
        14 5520 Stockdale Hwy                 Bakersfield         93383 6613229857          2
        15 13133 Rosedale Hwy                 Bakersfield         93314 6618292651          3
        16 3300 Buena Vista Rd                Bakersfield         93311 6616638131          4
        17 1400 Brundage Lane Space 101       Bakersfield         93304 6616380748          5
        18 3501 Panama Ln                     Bakersfield         93313 6618338414          2
        19 420A Weedpatch highway             Bakersfield         93307 6613636961          3
        20 1117 Kern Street                   Taft                93268 6617631177          4

20 rows selected.
```

HV_ITEM

```
CS342 SQL> DESC HV_ITEM
 Name              Null?    Type
 ----------------- -------- ------------
 ITEMID            NOT NULL NUMBER(8)
 INAME             NOT NULL VARCHAR2(60)
 PRICE                      NUMBER(38,2)

CS342 SQL> SELECT * FROM HV_ITEM;

    ITEMID INAME                                                        PRICE
---------- ------------------------------------------------------------ ----------
         1 Membership                                                       20
         2 Fries-Small                                                    1.49
         3 Fries-Medium                                                   1.79
         4 Fries-Large                                                    1.89
         5 Fries-CrissCut                                                 1.99
         6 Fries-Chili Cheese                                             2.99
         7 Onion Rings                                                    1.99
         8 Fried Zucchini                                                 1.99
         9 Side Salad                                                     1.79
        10 Fish & Chips                                                   4.99
        11 Coffee Channel Islands Roasting Co                             1.29
        12 DASANI Water                                                   1.49
        13 Milk                                                           1.09
        14 Orange Juice                                                   1.39
        15 Shake-Oreo Cookie                                              2.99
        16 Shake-Chocolate                                                2.99
        17 Shake-Vanilla                                                  2.99
```

ITEMID	INAME	PRICE
18	Shake-Strawberry	2.99
19	Chocolate Chip Cookie	.99
20	Strawberry Cheesecake	1.89
21	Chocolate Cake	1.69
22	Chicken Strips-3 Piece	3.19
23	Chicken Strips-5 Piece	4.79
24	Salad-Original	4.79
25	Salad-Hawaiian Grilled Chicken	4.99
26	Salad-Cranberry Apple Walnut	4.99
27	Salad-Southwest	4.99
28	Soda-Small-Coca Cola Classic	1.39
29	Soda-Medium-Coca Cola Classic	1.69
30	Soda-Large-Coca Cola Classic	1.89
31	Soda-Small-Diet Coke	1.39
32	Soda-Medium-Diet Coke	1.69
33	Soda-Large-Diet Coke	1.89
34	Soda-Small-Barqs Root Beer	1.39
35	Soda-Medium-Barqs Root Beer	1.69
36	Soda-Large-Barqs Root Beer	1.89
37	Soda-Small-Dr Pepper	1.39
38	Soda-Medium-Dr Pepper	1.69
39	Soda-Large-Dr Pepper	1.89
40	Soda-Small-Sprite	1.39
41	Soda-Medium-Sprite	1.69
42	Soda-Large-Sprite	1.89
43	Soda-Small-Minute Maid Orange	1.39
44	Soda-Medium-Minute Maid Orange	1.69
45	Soda-Large-Minute Maid Orange	1.89
46	Soda-Small-Rasberry Nestea	1.39
47	Soda-Medium-Rasberry Nestea	1.69
48	Soda-Large-Rasberry Nestea	1.89
49	Soda-Small-Iced Tea	1.39
50	Soda-Medium-Iced Tea	1.69
51	Soda-Large-Iced Tea	1.89
52	Burger-Famous Star with Cheese	2.49
53	Burger-Super Star with Cheese	3.99
54	Burger-Western Bacon Cheeseburger	2.99
55	Burger-Double Western Bacon Cheeseburger	4.09
56	Burger-Teriyaki Burger	2.69
57	Burger-Jalapeno Burger	2.99
58	Burger-Philly CheeseSteak Burger	3.49
59	Burger-The Big Carl	2.69
60	Burger-Original Six Dollar	3.99
61	Burger-Guacamole Bacon Six Dollar	4.99
62	Burger-Western Bacon Six Dollar	4.99
63	Burger-Portobello Mushroom Six Dollar	4.89
64	Burger-Jalapeno Six Dollar	4.69
65	Burger-Low Carb Six Dollar	3.99
66	Burger-Teriyaki Chicken Sandwich	3.89
67	Burger-Charbroiled Chicken Club	4.39
68	Burger-Charbroiled Santa Fe Chicken	4.19
69	Burger-Charbroiled BBQ Chicken	3.99
70	Burger-Bacon Swiss Crispy Chicken	4.49
71	Burger-Carls Catch Fish Sandwich	3.79
72	Combo-Famous Star with Cheese	4.99
73	Combo-Super Star with Cheese	6.49
74	Combo-Western Bacon Cheeseburger	5.49
75	Combo-Double Western Bacon Cheeseburger	6.59
76	Combo-Teriyaki Burger	5.19
77	Combo-Jalapeno Burger	5.49
78	Combo-Philly CheeseSteak Burger	5.99
79	Combo-The Big Carl	4.99
80	Combo-Original Six Dollar	6.49
81	Combo-Guacamole Bacon	7.49
82	Combo-Western Bacon Six Dollar	7.49
83	Combo-Portobello Mushroom Six Dollar	7.39
84	Combo-Jalapeno Six Dollar	7.19
85	Combo-Low Carb Six Dollar	6.49
86	Combo-Teriyaki Chicken Sandwich	6.39

ITEMID	INAME	PRICE
87	Combo-Charbroiled Chicken Club	6.89
88	Combo-Charbroiled Santa Fe Chicken	6.69
89	Combo-Charbroiled BBQ Chicken	6.49
90	Combo-Bacon Swiss Crispy Chicken	6.99
91	Combo-Carls Catch Fish Sandwich	6.29
92	Pumpkin Spice Latte-Grande	4.15
93	Pumpkin Spice Latte-Venti	4.65
94	White Chocolate Mocha-Grande	3.95
95	White Chocolate Mocha-Venti	4.45
96	Caramel Macchiato-Grande	3.85
97	Caramel Macchiato-Venti	4.25
98	Caffe Latte-Grande	3.35
99	Caffe Latte-Venti	3.6
100	Toffee Mocha-Grande	4.15
101	Toffee Mocha-Venti	4.65
102	Skinny Vanilla Latte-Grande	3.75
103	Skinny Vanilla Latte-Venti	4
104	Chai Tea Latte-Grande	3.55
105	Chai Tea Latte-Venti	3.95
106	Pike Place Roast-Grande	1.85
107	Pike Place Roast-Venti	2
108	Frapuccino-Strawberries N Creme-Grande	3.95
109	Frapuccino-Strawberries N Creme-Venti	4.6
110	Frapuccino-Caramel-Grande	3.95
111	Frapuccino-Caramel-Venti	4.6
112	Frapuccino-Mocha-Grande	3.95
113	Frapuccino-Mocha-Venti	4.6
114	Tea-Shaken Iced Passion Tea Lemonade-Grande	2.8
115	Tea-Shaken Iced Passion Tea Lemonade-Venti	3.15
116	Tea-Shaken Iced Black Tea-Grande	1.95
117	Tea-Shaken Iced Black Tea-Venti	2.3
118	Tea-Iced Chai Tea Latte-Grande	3.55
119	Tea-Iced Chai Tea Latte-Venti	3.95
120	Iced-Coffee w Milk-Grande	2.35
121	Iced-Coffee w Milk-Venti	2.65
122	Iced-Coffee-Grande	2.35
123	Iced-Coffee-Venti	2.65
124	Iced-Caffe Latte-Grande	3.35
125	Iced-Caffe Latte-Venti	3.6
126	Iced-Caramel Macchiato-Grande	3.85
127	Iced-Caramel Macchiato-Venti	4.25
128	Starbucks Perfect Oatmeal	2.45
129	Banana Walnut Bread	1.95
130	Artisan Ham Breakfast Sandwich	3.25
131	Reduced Fat Cinnamon Swirl Coffee Cake	1.95
132	Turkey and Swiss Sandwich	5.75
133	Roasted Tomato & Mozzarella Panini	5.45
134	Protein Plate	4.95
135	Petite Vanilla Bean Scone	2.25
136	Soda-Small-Pepsi	1.19
137	Soda-Medium-Pepsi	1.39
138	Soda-Large-Pepsi	1.59
139	Soda-XLarge-Pepsi	1.79
140	Soda-Small-Diet Pepsi	1.19
141	Soda-Medium-Diet Pepsi	1.39
142	Soda-Large-Diet Pepsi	1.59
143	Soda-XLarge-Diet Pepsi	1.79
144	Soda-Small-Sierra Mist	1.19
145	Soda-Medium-Sierra Mist	1.39
146	Soda-Large-Sierra Mist	1.59
147	Soda-XLarge-Sierra Mist	1.79
148	Soda-Small-Mountain Dew	1.19
149	Soda-Medium-Mountain Dew	1.39
150	Soda-Large-Mountain Dew	1.59
151	Soda-XLarge-Mountain Dew	1.79
152	Soda-Small-Mountain Dew Baja Blast	1.19
153	Soda-Medium-Mountain Dew Baja Blast	1.39
154	Soda-Large-Mountain Dew Baja Blast	1.59
155	Soda-XLarge-Mountain Dew Baja Blast	1.79

ITEMID	INAME	PRICE
156	Soda-Small-MUG Rootbeer	1.19
157	Soda-Medium-MUG Rootbeer	1.39
158	Soda-Large-MUG Rootbeer	1.59
159	Soda-XLarge-MUG Rootbeer	1.79
160	Soda-Small-Lemonade	1.19
161	Soda-Medium-Lemonade	1.39
162	Soda-Large-Lemonade	1.59
163	Soda-XLarge-Lemonade	1.79
164	Soda-Small-Brisk Rasberry Tea	1.19
165	Soda-Medium-Brisk Rasberry Tea	1.39
166	Soda-Large-Brisk Rasberry Tea	1.59
167	Soda-XLarge-Brisk Rasberry Tea	1.79
168	Soda-Small-Dr Pepper	1.19
169	Soda-Medium-Dr Pepper	1.39
170	Soda-Large-Dr Pepper	1.59
171	Soda-XLarge-Dr Pepper	1.79
172	Soda-Small-Fruit Punch	1.19
173	Soda-Medium-Fruit Punch	1.39
174	Soda-Large-Fruit Punch	1.59
175	Soda-XLarge-Fruit Punch	1.79
176	Strawberry Frutista Freeze	1.99
177	Mango Strawberry Frutista Freeze	1.99
178	Cheesy Nachos	.89
179	Crunchy Taco	.89
180	Cinnamon Twists	.89
181	Cheese Rollup	.89
182	Crispy Potato Soft Taco	.89
183	Soft Taco	.99
184	Bean Burrito	.99
185	Beefy 5 Layer Burrito	.99
186	Chicken Burrito	.99
187	Caramel Apple Empanada	.99
188	Chicken FlatBread Sandwich	.99
189	Pintos and Cheese or Rice	.99
190	Cheesy Fiesta Potatoes	1.19
191	Nachos BellGrande	2.99
192	Nachos Supreme	1.99
193	Nachos	.99
194	Volcano Nachos	3.49
195	Mexican Pizza	2.89
196	MexiMelt	1.69
197	Chicken Quesadilla	2.59
198	Steak Quesadilla	2.59
199	Cheese Quesadilla	1.89
200	Tostada	1.29
201	Chicken Grilled Taquitos	1.99
202	Steak Grilled Taquitos	1.99
203	Crunchwrap Supreme	2.39
204	Chipotle Steak Taco Salad	4.99
205	Chicken Ranch Taco Salad	4.99
206	Fiesta Taco Salad	4.39
207	Baja Chalupa	1.89
208	Supreme Chalupa	1.89
209	Nacho Cheese Chalupa	1.89
210	Baja Gordita	1.89
211	Supreme Gordita	1.89
212	Nacho Cheese Gordita	1.89
213	Chicken Baja Chalupa	2.59
214	Chicken Supreme Chalupa	2.59
215	Steak Baja Chalupa	2.59
216	Steak Supreme Chalupa	2.59
217	Chicken Ranchero Taco	1.89
218	Grilled Steak Taco	1.89
219	Double Decker Taco	1.29
220	Double Decker Taco Supreme	1.69
221	Crunchy Taco Supreme	1.29
222	Soft Taco Supreme	1.29
223	Volcano Taco	1.19
224	Chicken Soft Taco	1.29

ITEMID	INAME	PRICE
225	Grilled Stuft Burrito	2.39
226	Chicken Grilled Stuft Burrito	3.39
227	Steak Grilled Stuft Burrito	3.39
228	Burrito Supreme	2.29
229	Chicken Burrito Supreme	2.99
230	Steak Burrito Supreme	2.99
231	7 Layer Burrito	1.99
232	Enchirito	1.99
233	Volcano Burrito	2.99
234	Half Pound Nacho Crunch Burrito	1.99
235	Half Pound Cheesy Potato Burrito	1.99
236	Half Pound Beef Combo Burrito	1.99
237	Half Pound Cheesy Bean and Rice Burrito	1.49
238	Cheesy Double Beef Burrito	1.49
239	Combo 1-Burrito Supreme	4.49
240	Combo 2-Grilled Stuft Beef Burrito	4.39
241	Combo 2-Grilled Stuft Chicken Burrito	5.59
242	Combo 2-Grilled Stuft Steak Burrito	5.59
243	Combo 3- 3 Crunchy Tacos Supreme	4.59
244	Combo 3-3 Soft Tacos Supreme	4.59
245	Combo 4-Mexican Pizza & 2 Crunchy Tacos Supreme	5.89
246	Combo 5-Nachos BellGrande & Crunchy Taco Supreme	4.99
247	Combo 6-2 Beef Chalupas (Supreme or Baja) & Crunchy Taco	5.39
248	Combo 6-2 Chicken or Steak Chalupas & Crunchy Taco	6.49
249	Combo 7-Chicken or Steak Quesadilla	4.49
250	Combo 8-3 Crunchy or Soft Tacos	3.99
251	Combo 9-Crunchwrap Supreme & Crunchy Taco	4.29
252	Small-Sierra Mist	1.4
253	Medium-Sierra Mist	1.65
254	Large-Sierra Mist	1.85
255	Small-Pepsi	1.4
256	Medium-Pepsi	1.65
257	Large-Pepsi	1.85
258	Small-Mountain Dew	1.4
259	Medium-Mountain Dew	1.65
260	Large-Mountain Dew	1.85
261	Small-Diet Pepsi	1.4
262	Medium-Diet Pepsi	1.65
263	Large-Diet Pepsi	1.85
264	Small-Dr Pepper	1.4
265	Medium-Dr Pepper	1.65
266	Large-Dr Pepper	1.85
267	side-Mixed Veggies	0
268	side-Fried Rice	0
269	side-Steamed Rice	0
270	side-Chow Mein	0
271	entree-Orange Chicken	0
272	entree-Kung Pao Chicken	0
273	entree-Crispy Shrimp	0
274	entree-Beijing Beef	0
275	entree-String Bean Chicken Breast	0
276	entree-Sweet and Sour Pork	0
277	entree-Mandarin Chicken	0
278	entree-Broccoli Beef	0
279	entree-Chicken Egg Roll	0
280	entree-2 Veggie Spring Rolls	0
281	entree-Honey Walnut Shrimp	0
282	entree-Mushroom Chicken	0
283	entree-SweetFire Chicken Breast	0
284	Combo-Single Entree	3
285	Combo-Large Entree	9
286	Combo-Single Side	2
287	Combo-Large Side	3
288	Combo-Panda Bowl	4.99
289	Combo-Two Entree Plate	5.99
290	Combo-Three Entree Plate	7.24
291	Combo-Panda Feast	29

291 rows selected.

HV_REGISTERS

```
CS342 SQL> DESC HV_REGISTERS;
 Name              Null?    Type
 ----------------- -------- ------------
 EMPLOYEEID        NOT NULL NUMBER(8)
 CUSTOMERID        NOT NULL NUMBER(8)
 RDATE             NOT NULL TIMESTAMP(0)
 SELLMEMBERSHIP    NOT NULL NUMBER(1)

CS342 SQL> SELECT * FROM HV_REGISTERS;
EMPLOYEEID CUSTOMERID RDATE                        SELLMEMBERSHIP
---------- ---------- ------------------------- --------------
         1          1 02-JUL-10 07.50.00 AM                   0
         3          2 02-JUL-10 09.30.00 AM                   0
         1          3 02-JUL-10 10.45.00 AM                   2
         1          4 02-JUL-10 01.20.00 PM                   2
         1          5 02-JUL-10 02.24.00 PM                   2
         3          6 02-JUL-10 04.09.00 PM                   0
         7          7 03-JUL-10 07.05.00 AM                   2
         5          8 03-JUL-10 08.58.00 AM                   0
         7          9 03-JUL-10 03.11.00 PM                   2
         2         10 03-JUL-10 07.13.00 PM                   2
        11         11 04-JUL-10 10.05.00 AM                   0
        11          2 04-JUL-10 02.50.00 PM                   2
         9         11 05-JUL-10 07.07.00 AM                   2
```

HV_DELIVERS

```
CS342 SQL> DESC HV_DELIVERS
 Name              Null?    Type
 ----------------- -------- ------------
 EMPLOYEEID        NOT NULL NUMBER(8)
 ORDERNUMBER       NOT NULL NUMBER(8)
 DORECEIPTTIME     NOT NULL TIMESTAMP(0)
 DELIVERYTIME               TIMESTAMP(0)

CS342 SQL> SELECT * FROM HV_DELIVERS;
EMPLOYEEID ORDERNUMBER DORECEIPTTIME               DELIVERYTIME
---------- ----------- ------------------------- --------------------------
         2           1 02-JUL-10 07.58.00 AM     02-JUL-10 09.01.00 AM
         4           2 02-JUL-10 09.38.00 AM     02-JUL-10 10.15.00 AM
         2           3 02-JUL-10 10.48.00 AM     02-JUL-10 11.50.00 AM
         2           4 02-JUL-10 01.22.00 PM     02-JUL-10 02.10.00 PM
         4           5 02-JUL-10 02.25.00 PM     02-JUL-10 03.52.00 PM
         4           6 02-JUL-10 04.10.00 PM     02-JUL-10 05.40.00 PM
         6           7 03-JUL-10 07.08.00 AM     03-JUL-10 08.01.00 AM
         4           8 03-JUL-10 09.01.00 AM     03-JUL-10 09.49.00 AM
         4           9 03-JUL-10 11.22.00 AM     03-JUL-10 01.15.00 PM
         6          10 03-JUL-10 03.14.00 PM     03-JUL-10 05.13.00 PM
         8          11 03-JUL-10 05.47.00 PM     03-JUL-10 06.58.00 PM
         8          12 03-JUL-10 07.14.00 PM     03-JUL-10 08.05.00 PM
         1          13 04-JUL-10 08.16.00 AM     04-JUL-10 09.05.00 AM
         1          14 04-JUL-10 08.48.00 AM     04-JUL-10 11.02.00 AM
         9          15 04-JUL-10 10.08.00 AM     04-JUL-10 11.58.00 AM
         1          16 04-JUL-10 12.11.00 PM     04-JUL-10 01.05.00 PM
        10          17 04-JUL-10 01.04.00 PM     04-JUL-10 01.55.00 PM
        10          18 04-JUL-10 02.57.00 PM     04-JUL-10 03.49.00 PM
         9          19 04-JUL-10 04.30.00 PM     04-JUL-10 05.13.00 PM
         9          20 04-JUL-10 05.12.00 PM     04-JUL-10 07.00.00 PM
        10          21 04-JUL-10 06.40.00 PM
        11          22 05-JUL-10 07.04.00 AM     05-JUL-10 08.01.00 AM
        11          23 05-JUL-10 08.21.00 AM     05-JUL-10 09.15.00 AM
         9          24 05-JUL-10 10.12.00 AM     05-JUL-10 11.35.00 AM
         7          25 05-JUL-10 11.07.00 AM
        11          26 05-JUL-10 11.09.00 AM     05-JUL-10 11.40.00 AM
         7          27 05-JUL-10 11.21.00 AM
        11          28 05-JUL-10 11.27.00 AM
         7          29 05-JUL-10 11.36.00 AM
         9          30 05-JUL-10 11.41.00 AM

30 rows selected.
```

HV_OFFERS

```
CS342 SQL> DESC HV_OFFERS
 Name              Null?    Type
 ----------------- -------- ------------
 FID               NOT NULL NUMBER(8)
 ITEMID            NOT NULL NUMBER(8)

CS342 SQL> SELECT * FROM HV_OFFERS;                (HV_OFFERS Instance Continued Here … )

       FID     ITEMID                                    FID     ITEMID
---------- ----------                               ---------- ----------
         1          1                                      2         59
         2          2                                      2         60
         2          3                                      2         61
         2          4                                      2         62
         2          5                                      2         63
         2          6                                      2         64
         2          7                                      2         65
         2          8                                      2         66
         2          9                                      2         67
         2         10                                      2         68
         2         11                                      2         69
         2         12                                      2         70
         2         13                                      2         71
         2         14                                      2         72
         2         15                                      2         73
         2         16                                      2         74
         2         17                                      2         75
         2         18                                      2         76
         2         19                                      2         77
         2         20                                      2         78
         2         21                                      2         79
         2         22                                      2         80
         2         23                                      2         81
         2         24                                      2         82
         2         25                                      2         83
         2         26                                      2         84
         2         27                                      2         85
         2         28                                      2         86
         2         29                                      2         87
         2         30                                      2         88
         2         31                                      2         89
         2         32                                      2         90
         2         33                                      2         91
         2         34                                      3         92
         2         35                                      3         93
         2         36                                      3         94
         2         37                                      3         95
         2         38                                      3         96
         2         39                                      3         97
         2         40                                      3         98
         2         41                                      3         99
         2         42                                      3        100
         2         43                                      3        101
         2         44                                      3        102
         2         45                                      3        103
         2         46                                      3        104
         2         47                                      3        105
         2         48                                      3        106
         2         49                                      3        107
         2         50                                      3        108
         2         51                                      3        109
         2         52                                      3        110
         2         53                                      3        111
         2         54                                      3        112
         2         55                                      3        113
         2         56                                      3        114
         2         57                                      3        115
         2         58                                      3        116
```

FID	ITEMID
3	117
3	118
3	119
3	120
3	121
3	122
3	123
3	124
3	125
3	126
3	127
3	128
3	129
3	130
3	131
3	132
3	133
3	134
3	135
4	136
4	137
4	138
4	139
4	140
4	141
4	142
4	143
4	144
4	145
4	146
4	147
4	148
4	149
4	150
4	151
4	152
4	153
4	154
4	155
4	156
4	157
4	158
4	159
4	160
4	161
4	162
4	163
4	164
4	165
4	166
4	167
4	168
4	169
4	170
4	171
4	172
4	173
4	174
4	175
4	176
4	177
4	178
4	179
4	180
4	181
4	182
4	183
4	184
4	185

FID	ITEMID
4	186
4	187
4	188
4	189
4	190
4	191
4	192
4	193
4	194
4	195
4	196
4	197
4	198
4	199
4	200
4	201
4	202
4	203
4	204
4	205
4	206
4	207
4	208
4	209
4	210
4	211
4	212
4	213
4	214
4	215
4	216
4	217
4	218
4	219
4	220
4	221
4	222
4	223
4	224
4	225
4	226
4	227
4	228
4	229
4	230
4	231
4	232
4	233
4	234
4	235
4	236
4	237
4	238
4	239
4	240
4	241
4	242
4	243
4	244
4	245
4	246
4	247
4	248
4	249
4	250
4	251
5	252
5	253
5	254

FID	ITEMID
5	255
5	256
5	257
5	258
5	259
5	260
5	261
5	262
5	263
5	264
5	265
5	266
5	267
5	268
5	269
5	270
5	271
5	272
5	273
5	274
5	275
5	276
5	277
5	278
5	279
5	280
5	281
5	282
5	283
5	284
5	285
5	286
5	287
5	288
5	289
5	290
5	291
6	2
6	3
6	4
6	5
6	6
6	7
6	8
6	9
6	10
6	11
6	12
6	13
6	14
6	15
6	16
6	17
6	18
6	19
6	20
6	21
6	22
6	23
6	24
6	25
6	26
6	27
6	28
6	29
6	30
6	31
6	32
6	33

FID	ITEMID
6	34
6	35
6	36
6	37
6	38
6	39
6	40
6	41
6	42
6	43
6	44
6	45
6	46
6	47
6	48
6	49
6	50
6	51
6	52
6	53
6	54
6	55
6	56
6	57
6	58
6	59
6	60
6	61
6	62
6	63
6	64
6	65
6	66
6	67
6	68
6	69
6	70
6	71
6	72
6	73
6	74
6	75
6	76
6	77
6	78
6	79
6	80
6	81
6	82
6	83
6	84
6	85
6	86
6	87
6	88
6	89
6	90
6	91
7	92
7	93
7	94
7	95
7	96
7	97
7	98
7	99
7	100
7	101
7	102

FID	ITEMID
7	103
7	104
7	105
7	106
7	107
7	108
7	109
7	110
7	111
7	112
7	113
7	114
7	115
7	116
7	117
7	118
7	119
7	120
7	121
7	122
7	123
7	124
7	125
7	126
7	127
7	128
7	129
7	130
7	131
7	132
7	133
7	134
7	135
8	136
8	137
8	138
8	139
8	140
8	141
8	142
8	143
8	144
8	145
8	146
8	147
8	148
8	149
8	150
8	151
8	152
8	153
8	154
8	155
8	156
8	157
8	158
8	159
8	160
8	161
8	162
8	163
8	164
8	165
8	166
8	167
8	168
8	169
8	170
8	171

FID	ITEMID
8	172
8	173
8	174
8	175
8	176
8	177
8	178
8	179
8	180
8	181
8	182
8	183
8	184
8	185
8	186
8	187
8	188
8	189
8	190
8	191
8	192
8	193
8	194
8	195
8	196
8	197
8	198
8	199
8	200
8	201
8	202
8	203
8	204
8	205
8	206
8	207
8	208
8	209
8	210
8	211
8	212
8	213
8	214
8	215
8	216
8	217
8	218
8	219
8	220
8	221
8	222
8	223
8	224
8	225
8	226
8	227
8	228
8	229
8	230
8	231
8	232
8	233
8	234
8	235
8	236
8	237
8	238
8	239
8	240

FID	ITEMID
8	241
8	242
8	243
8	244
8	245
8	246
8	247
8	248
8	249
8	250
8	251
9	252
9	253
9	254
9	255
9	256
9	257
9	258
9	259
9	260
9	261
9	262
9	263
9	264
9	265
9	266
9	267
9	268
9	269
9	270
9	271
9	272
9	273
9	274
9	275
9	276
9	277
9	278
9	279
9	280
9	281
9	282
9	283
9	284
9	285
9	286
9	287
9	288
9	289
9	290
9	291
10	2
10	3
10	4
10	5
10	6
10	7
10	8
10	9
10	10
10	11
10	12
10	13
10	14
10	15
10	16
10	17
10	18
10	19

FID	ITEMID
10	20
10	21
10	22
10	23
10	24
10	25
10	26
10	27
10	28
10	29
10	30
10	31
10	32
10	33
10	34
10	35
10	36
10	37
10	38
10	39
10	40
10	41
10	42
10	43
10	44
10	45
10	46
10	47
10	48
10	49
10	50
10	51
10	52
10	53
10	54
10	55
10	56
10	57
10	58
10	59
10	60
10	61
10	62
10	63
10	64
10	65
10	66
10	67
10	68
10	69
10	70
10	71
10	72
10	73
10	74
10	75
10	76
10	77
10	78
10	79
10	80
10	81
10	82
10	83
10	84
10	85
10	86
10	87
10	88

FID	ITEMID
10	89
10	90
10	91
11	92
11	93
11	94
11	95
11	96
11	97
11	98
11	99
11	100
11	101
11	102
11	103
11	104
11	105
11	106
11	107
11	108
11	109
11	110
11	111
11	112
11	113
11	114
11	115
11	116
11	117
11	118
11	119
11	120
11	121
11	122
11	123
11	124
11	125
11	126
11	127
11	128
11	129
11	130
11	131
11	132
11	133
11	134
11	135
12	136
12	137
12	138
12	139
12	140
12	141
12	142
12	143
12	144
12	145
12	146
12	147
12	148
12	149
12	150
12	151
12	152
12	153
12	154
12	155
12	156
12	157

FID	ITEMID
12	158
12	159
12	160
12	161
12	162
12	163
12	164
12	165
12	166
12	167
12	168
12	169
12	170
12	171
12	172
12	173
12	174
12	175
12	176
12	177
12	178
12	179
12	180
12	181
12	182
12	183
12	184
12	185
12	186
12	187
12	188
12	189
12	190
12	191
12	192
12	193
12	194
12	195
12	196
12	197
12	198
12	199
12	200
12	201
12	202
12	203
12	204
12	205
12	206
12	207
12	208
12	209
12	210
12	211
12	212
12	213
12	214
12	215
12	216
12	217
12	218
12	219
12	220
12	221
12	222
12	223
12	224
12	225
12	226

FID	ITEMID
12	227
12	228
12	229
12	230
12	231
12	232
12	233
12	234
12	235
12	236
12	237
12	238
12	239
12	240
12	241
12	242
12	243
12	244
12	245
12	246
12	247
12	248
12	249
12	250
12	251
13	252
13	253
13	254
13	255
13	256
13	257
13	258
13	259
13	260
13	261
13	262
13	263
13	264
13	265
13	266
13	267
13	268
13	269
13	270
13	271
13	272
13	273
13	274
13	275
13	276
13	277
13	278
13	279
13	280
13	281
13	282
13	283
13	284
13	285
13	286
13	287
13	288
13	289
13	290
13	291
14	2
14	3
14	4
14	5

FID	ITEMID
14	6
14	7
14	8
14	9
14	10
14	11
14	12
14	13
14	14
14	15
14	16
14	17
14	18
14	19
14	20
14	21
14	22
14	23
14	24
14	25
14	26
14	27
14	28
14	29
14	30
14	31
14	32
14	33
14	34
14	35
14	36
14	37
14	38
14	39
14	40
14	41
14	42
14	43
14	44
14	45
14	46
14	47
14	48
14	49
14	50
14	51
14	52
14	53
14	54
14	55
14	56
14	57
14	58
14	59
14	60
14	61
14	62
14	63
14	64
14	65
14	66
14	67
14	68
14	69
14	70
14	71
14	72
14	73
14	74

FID	ITEMID
14	75
14	76
14	77
14	78
14	79
14	80
14	81
14	82
14	83
14	84
14	85
14	86
14	87
14	88
14	89
14	90
14	91
15	92
15	93
15	94
15	95
15	96
15	97
15	98
15	99
15	100
15	101
15	102
15	103
15	104
15	105
15	106
15	107
15	108
15	109
15	110
15	111
15	112
15	113
15	114
15	115
15	116
15	117
15	118
15	119
15	120
15	121
15	122
15	123
15	124
15	125
15	126
15	127
15	128
15	129
15	130
15	131
15	132
15	133
15	134
15	135
16	136
16	137
16	138
16	139
16	140
16	141
16	142
16	143

FID	ITEMID
16	144
16	145
16	146
16	147
16	148
16	149
16	150
16	151
16	152
16	153
16	154
16	155
16	156
16	157
16	158
16	159
16	160
16	161
16	162
16	163
16	164
16	165
16	166
16	167
16	168
16	169
16	170
16	171
16	172
16	173
16	174
16	175
16	176
16	177
16	178
16	179
16	180
16	181
16	182
16	183
16	184
16	185
16	186
16	187
16	188
16	189
16	190
16	191
16	192
16	193
16	194
16	195
16	196
16	197
16	198
16	199
16	200
16	201
16	202
16	203
16	204
16	205
16	206
16	207
16	208
16	209
16	210
16	211
16	212

FID	ITEMID
16	213
16	214
16	215
16	216
16	217
16	218
16	219
16	220
16	221
16	222
16	223
16	224
16	225
16	226
16	227
16	228
16	229
16	230
16	231
16	232
16	233
16	234
16	235
16	236
16	237
16	238
16	239
16	240
16	241
16	242
16	243
16	244
16	245
16	246
16	247
16	248
16	249
16	250
16	251
17	252
17	253
17	254
17	255
17	256
17	257
17	258
17	259
17	260
17	261
17	262
17	263
17	264
17	265
17	266
17	267
17	268
17	269
17	270
17	271
17	272
17	273
17	274
17	275
17	276
17	277
17	278
17	279
17	280
17	281

FID	ITEMID
17	282
17	283
17	284
17	285
17	286
17	287
17	288
17	289
17	290
17	291
18	2
18	3
18	4
18	5
18	6
18	7
18	8
18	9
18	10
18	11
18	12
18	13
18	14
18	15
18	16
18	17
18	18
18	19
18	20
18	21
18	22
18	23
18	24
18	25
18	26
18	27
18	28
18	29
18	30
18	31
18	32
18	33
18	34
18	35
18	36
18	37
18	38
18	39
18	40
18	41
18	42
18	43
18	44
18	45
18	46
18	47
18	48
18	49
18	50
18	51
18	52
18	53
18	54
18	55
18	56
18	57
18	58
18	59
18	60

FID	ITEMID
18	61
18	62
18	63
18	64
18	65
18	66
18	67
18	68
18	69
18	70
18	71
18	72
18	73
18	74
18	75
18	76
18	77
18	78
18	79
18	80
18	81
18	82
18	83
18	84
18	85
18	86
18	87
18	88
18	89
18	90
18	91
19	92
19	93
19	94
19	95
19	96
19	97
19	98
19	99
19	100
19	101
19	102
19	103
19	104
19	105
19	106
19	107
19	108
19	109
19	110
19	111
19	112
19	113
19	114
19	115
19	116
19	117
19	118
19	119
19	120
19	121
19	122
19	123
19	124
19	125
19	126
19	127
19	128
19	129

FID	ITEMID
19	130
19	131
19	132
19	133
19	134
19	135
20	136
20	137
20	138
20	139
20	140
20	141
20	142
20	143
20	144
20	145
20	146
20	147
20	148
20	149
20	150
20	151
20	152
20	153
20	154
20	155
20	156
20	157
20	158
20	159
20	160
20	161
20	162
20	163
20	164
20	165
20	166
20	167
20	168
20	169
20	170
20	171
20	172
20	173
20	174
20	175
20	176
20	177
20	178
20	179
20	180
20	181
20	182
20	183
20	184
20	185
20	186
20	187
20	188
20	189
20	190
20	191
20	192
20	193
20	194
20	195
20	196
20	197
20	198

```
       FID     ITEMID
---------- ----------
        20        199
        20        200
        20        201
        20        202
        20        203
        20        204
        20        205
        20        206
        20        207
        20        208
        20        209
        20        210
        20        211
        20        212
        20        213
        20        214
        20        215
        20        216
        20        217
        20        218
        20        219
        20        220
        20        221
        20        222
        20        223
        20        224
        20        225
        20        226
        20        227
        20        228
        20        229
        20        230
        20        231
        20        232
        20        233
        20        234
        20        235
        20        236
        20        237
        20        238
        20        239
        20        240
        20        241
        20        242
        20        243
        20        244
        20        245
        20        246
        20        247
        20        248
        20        249
        20        250
        20        251

1411 rows selected.
```

HV_INCLUDES

```
CS342 SQL> DESC HV_INCLUDES
 Name              Null?    Type
 ----------------- -------- ------------
 ORDERNUMBER       NOT NULL NUMBER(8)
 ITEMID            NOT NULL NUMBER(8)
 FID               NOT NULL NUMBER(8)
 QUANTITY          NOT NULL NUMBER(8)
 OPRICE            NOT NULL NUMBER(38,2)

CS342 SQL> SELECT * FROM HV_INCLUDES;

ORDERNUMBER     ITEMID        FID   QUANTITY     OPRICE
----------- ---------- ---------- ---------- ----------
          1         90          2          1       6.99
          1          7          2          1       1.99
          1         19          2          2        .99
          1         93          7          1       4.65
          2        243         16          3       4.59
          2        177         16          2        176
          3          1          1          1         20
          3         72          2          1       4.99
          3         74          2          1       5.49
          3         96         15          2       3.85
          4          1          1          1         20
          4         26         10          1       4.99
          4         50         10          1       1.69
          4         78         10          1       5.99
          5          1          1          1         20
          5        291          9          1         29
          6         83         14          1       7.39
          6        290          9          3       7.24
          7          1          1          1         20
          7        104          3          1       3.55
          7        100          3          1       4.15
          7         97          3          1       4.25
          7         96          3          1       3.85
          8         52         14         10       2.49
          9        289         13          2       5.99
          9        102         15          1       3.75
          9         95         15          1       4.45
          9        206         20          1       4.39
         10          1          1          1         20
         10         52         14          5       2.49
         10          3         14          5       1.79
         11          1          1          1         20
         11        291          9          1         29
         11        285          9          1          9
         12          1          1          1         20
         12        107         11          1          2
         12        105         11          1       3.95
         12        110         11          1       3.95
         12        128         11          1       2.45
         12        127         11          1       4.25
         13          1          1          1         20
         13        132         11          2       5.75
         13        114         11          1        2.8
         13        102         11          1       3.75
         14          1          1          1         20
         14         25         19          1       4.99
         14         31         19          1       1.39
         15        289          5          1       5.99
         15        288          5          1       4.99
         15         97          7          4       4.25
         16        288          9          8       4.99
         17         93         11          2       4.15
         17        100         11          1       4.15
         17        113         11          4        4.6
         17        129         11          5       1.29
```

```
ORDERNUMBER     ITEMID        FID   QUANTITY     OPRICE
----------- ---------- ---------- ---------- ----------
         18          1          1          1         20
         18        291         13          1         29
         19         69          6          1       3.99
         19         60          6          1       3.99
         19        108          3          4       3.95
         19        125          3          2        3.6
         20         85          2          1       6.49
         20        290          5          1       7.24
         20        288          5          1       4.99
         20        123          3          1       2.65
         20        201          4          6       1.99
         22          1          1          1         20
         22         99          7          1        3.6
         22        107          7          3          2
         23         93         19          1       4.65
         23        100         19          2       4.15
         24        188         20          5        .99
         24        183         20          7        .99
         24        179         20         50        .89
         24        184         20          4        .99
         24        193         20         10        .99
         25        111         11          1         11
         25        248          8          1       6.49
         26        289          9          2       5.99
         27        246         12          1       4.99
         27        242         12          1       5.59
         27        249         12          1       4.49
         28         88          2          1       6.69
         28         84          2          1       7.19
         28        289          5          2       5.99
         29         23         14          1       4.79
         30         29         13          1         29
         30        251         20          1       4.29

88 rows selected.
```

HV_COMBO

```
CS342 SQL> DESC HV_COMBO
 Name              Null?    Type
 ----------------- -------- ------------
 COMBO_ID          NOT NULL NUMBER(8)
 ITEM_CR           NOT NULL NUMBER(8)
 ORDERNUM          NOT NULL NUMBER(8)

CS342 SQL> SELECT * FROM HV_COMBO;

  COMBO_ID    ITEM_CR   ORDERNUM
---------- ---------- ----------
         1         90          1
         2        243          2
         3        243          2
         4        243          2
         5         72          3
         6         74          3
         7         78          4
         8        291          5
         9         83          6
        10        290          6
        11        290          6
        12        290          6
        13        289          9
        14        289          9
        15        285         11
        16        291         11
        17        288         15
        18        289         15
        19        288         16
```

```
  COMBO_ID    ITEM_CR   ORDERNUM
---------- ---------- ----------
        20        288         16
        21        288         16
        22        288         16
        23        288         16
        24        288         16
        25        288         16
        26        288         16
        27        291         18
        28         85         20
        29        288         20
        30        290         20
        31        248         25
        32        289         26
        33        289         26
        34        242         27
        35        246         27
        36        249         27
        37         84         28
        38         88         28
        39        289         28
        40        289         28
        41        251         30

41 rows selected.
```

HV_CONTAINS

```
CS342 SQL> DESC HV_CONTAINS
 Name              Null?    Type
 ----------------- -------- ------------
 CONTAINER         NOT NULL NUMBER(8)
 CMBONUM           NOT NULL NUMBER(8)
 ITEMID            NOT NULL NUMBER(8)

CS342 SQL> SELECT * FROM HV_CONTAINS;

 CONTAINER    CMBONUM     ITEMID
---------- ---------- ----------
         1          1         47
         2          2        150
         3          3        158
         4          4        142
         5          5         39
         6          6         33
         7          7         51
         8          8        271
         9          8        278
        10          8        277
        11          8        268
        12          8        270
        13          9         30
        14         10        271
        15         10        276
        16         10        280
        17         10        270
        18         11        273
        19         11        275
        20         11        279
        21         11        268
        22         12        272
        23         12        274
        24         12        281
        25         12        269
        26         13        271
        27         13        277
        28         13        270
        29         14        283
        30         14        271
```

CONTAINER	CMBONUM	ITEMID
31	14	268
32	15	281
33	16	278
34	16	277
35	16	271
36	16	270
37	16	268
38	17	271
39	17	269
40	18	281
41	18	280
42	18	268
43	19	271
44	19	270
45	20	271
46	20	270
47	21	272
48	21	267
49	22	276
50	22	268
51	23	279
52	23	270
53	24	277
54	24	269
55	25	282
56	25	267
57	26	281
58	26	269
59	27	283
60	27	281
61	27	275
62	27	268
63	27	270
64	28	42
65	29	271
66	29	270
67	30	273
68	30	275
69	30	278
70	30	269
71	31	215
72	31	214
73	31	166
74	32	278
75	32	277
76	32	268
77	33	272
78	33	271
79	33	270
80	34	142
81	35	146
82	36	197
83	36	174
84	37	42
85	38	48
86	39	271
87	39	278
88	39	267
89	40	276
90	40	279
91	40	267
92	41	150

92 rows selected.

Part 4: Queries in SQL

In this part of phase III queries will be expressed using the SQL (structured query language) tailored for the database relation instances shown in the previous phase. The result of the queries will also be listed.

Query 1: List all restaurants which offer combo meals priced $4.99 or less.

```
select distinct r.rstid, r.rname
from      hv_restaurant r inner join
          hv_franchise f on r.rstid = f.rstid inner join
          hv_offers o on f.fid = o.fid inner join
          hv_item i on o.itemid = i.itemid
where   (i.price <= 4.99) and (i.iname like 'Combo%')
order by r.rstid
```

Result:

```
CS342 SQL> @Query1.sql

     RSTID RNAME
---------- ------------------------------
         2 Carls Jr
         4 Taco Bell
         5 Panda Express
```

Query 2: List customers who've placed at least two orders priced greater than $40 total.

```
select distinct c.customerid, c.cfirst,c.cmidinitial, c.clast

from hv_customer c,

 (select p.ordernumber, sum(p.deliverycharge+t.itemstotal) as ordertotal
  from hv_order p,
      (select o.ordernumber, sum(i.quantity * i.oprice) as itemstotal
                from hv_order o inner join hv_includes i
                on o.ordernumber = i.ordernumber
                group by o.ordernumber) t
 where (p.ordernumber = t.ordernumber) and ((p.deliverycharge+t.itemstotal) > 40)
 group by p.ordernumber) oone,

(select p.ordernumber, sum(p.deliverycharge+t.itemstotal) as ordertotal
  from hv_order p,
      (select o.ordernumber, sum(i.quantity * i.oprice) as itemstotal
                from hv_order o inner join hv_includes i
                on o.ordernumber = i.ordernumber
                group by o.ordernumber) t
 where (p.ordernumber = t.ordernumber) and ((p.deliverycharge+t.itemstotal) > 40)
 group by p.ordernumber) otwo,

hv_order link1, hv_order link2

where (link1.ordernumber = oone.ordernumber) and (link2.ordernumber = otwo.ordernumber) and
        (link1.ordernumber <> link2.ordernumber) and
        (link1.customerid = c.customerid) and (link2.customerid = c.customerid)

order by c.customerid;
```

Result:

```
CS342 SQL> @Query2.sql

CUSTOMERID CFIRST   C CLAST
---------- -------- - ------------
         2 Rachel     Larson
         4 Saul     M Steinbeck
         5 Daniel     Simpson
         7 Karen    F Chu
         8 Ryan     L Scott
```

Query 3: List restaurants from which more than 7 different customers have ordered.

```
select r.*
from hv_restaurant r,
        (select rdc.rstid
         from(select distinct o.customerid, r.rstid
                 from   hv_restaurant r inner join
                          hv_franchise f on r.rstid = f.rstid inner join
                          hv_includes i on f.fid = i.fid inner join
                          hv_order o on i.ordernumber = o.ordernumber
                 )rdc
         group by (rdc.rstid)
         having count (rdc.customerid)>7
         )r8
where r.rstid = r8.rstid and r.rname <> 'Every Meal Delivery';
```

Result:

```
CS342 SQL> @Query3

     RSTID RNAME
---------- ------------------------------
         5 Panda Express
         3 Starbucks Coffee
```

Query 4: List customers who average the smallest number of items per order.

```
select cs.*
from hv_customer cs

minus

select cb.*
from hv_customer cb,

(select distinct c.customerid, avg(do.total) as itemavg
        from hv_order c,
                (select distinct i.ordernumber, count(quantity) as total
                 from    hv_includes i
                 group by i.ordernumber)do,
         hv_order link
        where (c.customerid = link.customerid) and (do.ordernumber = link.ordernumber)
        group by (c.customerid)) c1,

(select distinct c.customerid, avg(do.total) as itemavg
        from hv_order c,
                (select distinct i.ordernumber, count(quantity) as total
                 from    hv_includes i
                 group by i.ordernumber)do,
         hv_order link
        where (c.customerid = link.customerid) and (do.ordernumber = link.ordernumber)
        group by (c.customerid))c2

where (c1.itemavg< c2.itemavg) and (c2.customerid = cb.customerid);
```

Result:

```
CS342 SQL> @Query4.sql

CUSTOMERID CFIRST   C CLAST             PHONE MEMBERSTATUS
---------- -------- - ------------ ---------- ------------
         6 Matt       Groening     6618614124            0
```

Query 5: List customers who've been members between 7/3/2010, and 7/04/2010.

```
select c.*
from hv_customer c,
(
select td.* from
(     select r.*
      from hv_registers r
      where (r.sellmembership = 1) or (r.sellmembership = 2)  )                              td
inner join
(select r1.*
 from hv_registers r1, hv_registers r2
 where (r1.rdate <= (timestamp '2010-07-03 00:00:00')) and (r1.sellmembership = 2) and
      ((r2.rdate >  (timestamp '2010-07-03 00:00:00')  or  r2.rdate <= r1.rdate)  or
        r2.sellmembership <> 1) and (r1.customerid = r2.customerid) )                        td2
on (td.employeeid = td2.employeeid and
td.customerid = td2.customerid)
)td3
where td3.customerid = c.customerid;
```

Result:

```
CS342 SQL> @Query5.sql

CUSTOMERID CFIRST   C CLAST            PHONE MEMBERSTATUS
---------- -------- - ------------ ---------- ------------
         3 Henry    K Ricks        6617589212            1
         4 Saul     M Steinbeck    6612059354            1
         5 Daniel     Simpson      8053652241            1
```

Query 6: List customers who've never placed an order from a Taco Bell in the 93308 zip code area.

```
select n.*
from hv_customer n
where not exists (
      select o.customerid
        from hv_restaurant r inner join
             hv_franchise f on r.rstid = f.rstid inner join
             hv_includes i on f.fid = i.fid inner join
             hv_order o on i.ordernumber = o.ordernumber
      where (f.fzip = 93308) and (r.rname = 'taco bell') and (n.customerid = o.customerid)
                );
```

Result:

```
CS342 SQL> @Query6.sql

CUSTOMERID CFIRST    C CLAST              PHONE MEMBERSTATUS
---------- --------- - ------------- ---------- ------------
         1 Polly     G Fredericks    6613314574            0
         2 Rachel      Larson        6615893212            0
         3 Henry     K Ricks         6617589212            1
         4 Saul      M Steinbeck     6612059354            1
         6 Matt        Groening      6618614124            0
         7 Karen     F Chu           8057584790            1
         8 Ryan      L Scott         8057583787            0
         9 Jeff        Green         6615894512            1
        11 Greg      A Ford          8059683641            0
```

Query 7: List franchises with the second least expensive combos.

```
select  f.*
from        hv_franchise f inner join
            hv_offers m on f.fid = m.fid inner join
            hv_item i on m.itemid = i.itemid,
(select i.*
from   hv_item i
where (i.iname like 'combo%') and exists (
            select i2.*
            from hv_item i2
               where (i2.iname like 'combo%') and
                  (i2.price < i.price) and not exists (
                  select i3.* from hv_item i3 where (i3.iname like 'combo%')
                  and i3.price < i.price and i3.price <> i2.price
                                    )
                        )
) cc
where i.itemid = cc.itemid;
```

Result:

```
CS342 SQL> @Query7.sql

       FID FSTREET                        FCITY               FZIP     FPHONE      RSTID
---------- ------------------------------ -------------- ---------- ---------- ----------
         5 9200 Rosedale Highway 300      Bakersfield         93312 6615872316          5
         5 9200 Rosedale Highway 300      Bakersfield         93312 6615872316          5
         9 5120 Stockdale Hwy Space A     Bakersfield         93309 6613232033          5
         9 5120 Stockdale Hwy Space A     Bakersfield         93309 6613232033          5
        13 5041 Gosford Rd F1             Bakersfield         93313 6616640391          5
        13 5041 Gosford Rd F1             Bakersfield         93313 6616640391          5
        17 1400 Brundage Lane Space 101   Bakersfield         93304 6616380748          5
        17 1400 Brundage Lane Space 101   Bakersfield         93304 6616380748          5

8 rows selected.
```

Query 8: List employees who delivered items from each franchise location in the 93312 zip code area on 7/4/2010.

```
select      d.employeeid
from        hv_includes i inner join
             hv_franchise f on i.fid = f.fid inner join
             hv_order o on i.ordernumber = o.ordernumber inner join
             hv_delivers d on o.ordernumber = d.ordernumber
where       (f.fzip = 93312) and (d.deliverytime > to_date('3-jul-2010', 'dd-mon-yyyy'))
   and      (d.deliverytime < to_date('5-jul-2010', 'dd-mon-yyyy'))
```

Result:

```
CS342 SQL> @Query8

EMPLOYEEID
----------
         9
         9
         9
         9
         6
         6
         6
         6
         9
         9
         9
         9
         9
         9
         9
         9

16 rows selected.
```

Query 9: List employees who've not sold memberships, or made deliveries to any customers.

```
select      e.*
from              hv_registers r inner join
                  hv_employee e on r.employeeid = e.employeeid inner join
                  hv_delivers d on e.employeeid = d.employeeid
 where not exists (select e1.*
           from   hv_employee e1
           where  (e1.employeeid = r.employeeid and r.sellmembership = 2 or
                     e1.employeeid = d.employeeid and d.deliverytime <> null)
           and    e.employeeid = e1.employeeid);
```

Result:

```
CS342 SQL> @Query9.sql

EMPLOYEEID EFIRST               E ELAST
---------- -------------------- - ----------------------------------------
         1 Stanley              L Marsh
         1 Stanley              L Marsh
         1 Stanley              L Marsh
        11 Allison                Jester
        11 Allison                Jester
        11 Allison                Jester
        11 Allison                Jester

7 rows selected.
```

Query 10: List employees for whom it's taken more than 1.5 hours to deliver an order.

```
select  e.*
from    hv_delivers e
where   e.deliverytime <> null and substr((e.deliverytime-e.doreceipttime),
     instr((e.deliverytime-e.doreceipttime),' ')+1,2) > 1.5;
```

No Result

Command: Create a new table from an existing table using CREATE TABLE …. AS SELECT…

The following command creates a table from every Carl's Jr. Location in the database.

```
Create Table HV_CJTEST
AS (SELECT F.FID, R.RNAME, F.FSTREET, F.FCITY, F.FZIP, F.FPHONE
FROM HV_FRANCHISE F INNER JOIN HV_RESTAURANT R ON F.RSTID = R.RSTID
WHERE (R.RNAME = 'Carls Jr'))
```

Result:

```
CS342 SQL> @Command1

Table created.

CS342 SQL> select * from HV_CJTEST;

       FID RNAME        FSTREET                   FCITY                FZIP     FPHONE
---------- ------------ ------------------------- -------------- ---------- ----------
         2 Carls Jr     9500 Brimhall Rd Ste A    Bakersfield         93312 6615874859
         6 Carls Jr     4520 Coffee Road          Bakersfield         93312 6615879085
        10 Carls Jr     9000 Ming Ave Ste Q       Bakersfield         93311 6616652396
        14 Carls Jr     5520 Stockdale Hwy        Bakersfield         93383 6613229857
        18 Carls Jr     3501 Panama Ln            Bakersfield         93313 6618338414
```

Part 5: Loading Record Data to the DB

5.1 Descriptions of Data Loading Methods

Many options are available for one to load data into a database. The most common available one for loading records into a schema through Oracle SQL*Plus is the insert statement. The syntax for loading a new record into a relation is as follows:

```
INSERT INTO <table_name>VALUES(<comma_separated_value_list>);
```

though the insert statement can also be run tailored to the specific business needs of an organization through use of extra statements passed along while inserting a record for the sake of either security, or more simply data format. Because command record input can get quite tedious, more efficient methods of data loading come back to program applications tailored to complete such tasks.

5.2 The Java DataLoader Program

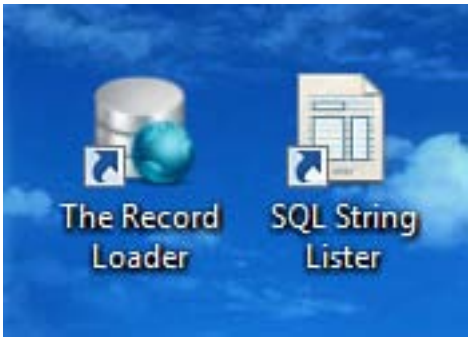
Though it's possible to go on the web to buy expensive applications not necessarily optimized to meet the needs of a person's specific database tasks, sometimes the best option is making data entry a custom endeavor. The Java DataLoader program is one example of such a feat. The program takes text files setup in a specified format to note the records it's to enter. After setting up a file delimited with pipe characters between records ready for entry into the database, a user can use java to run the program. One thing to note though is that the type of the fields must also be specified to the program so it knows how to organize the stream of information to send into Oracle. In its current state the program is quite useful for the task it was made to perform, and it was even recommended to students that it be built on for good practice, and functionality.

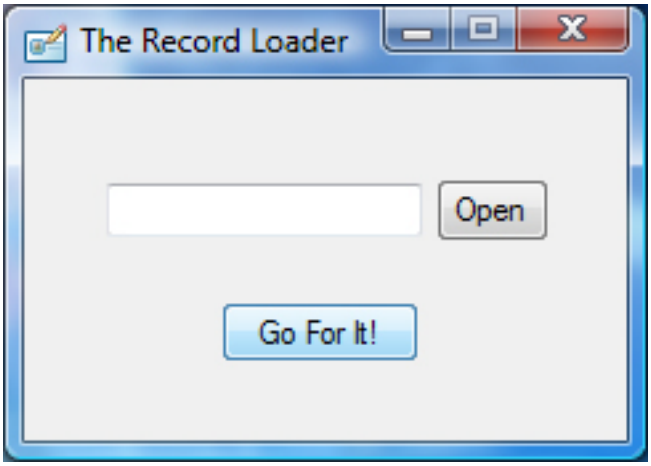
5.3 The C # Record Loader Program

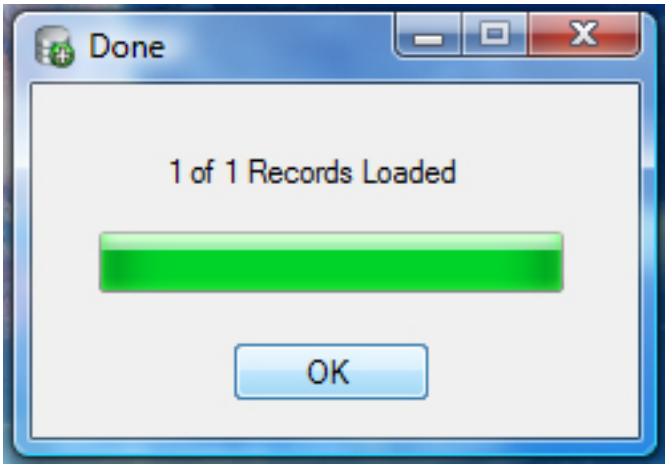
Though building on a java program is an enticing proposition, I decided to go with Microsoft C# to build a data loader. The main reason I did so was to start gaining familiarity with programming C# forms for this project. Originally, I'd made the C # application a simple one record insert console command line connection tester to understand the process of setting up a data provider in a database application, but I found the GUI to be the more ease of use option.

Much like the Java-based data loader application, the foundation of the program was made to take in a delimited record file of some sort of form to stream into the database all at once. I went ahead, and fashioned a template to work with files of comma separated values type. I figured I could use Microsoft Excel to easily enter the relational data into

tables, save them to .csv, then modify the files with a simple header to specify the attribute types of the relations being streamed off into the database. The original console database provider code I adapted to the windows form to enable data insertion. Other than that, it was then a matter of application aesthetics. I also managed to put together a separate similar application to show the file stream out to a Windows message box so I could make sure no mistakes were made in the data entry process prior to the data export. The following are a few simple screenshots of the record loader, and the string format display application.







I'll be using this similar technology in the implementation of this project.

Phase IV

The Oracle DBMS PL/SQL Component

Part 1: Common Features in Oracle PL/SQL and MS Trans-SQL

Advanced database centered programming languages are commonly included with many modern database management systems. They help database administrators manage a database with more flexible options than those offered by simple query statements alone. Two languages from two of the most popular DBMSs now offered are Microsoft Transact-SQL (T-SQL) from the Microsoft SQL Server DBMS, and PL/SQL from the Oracle DBMS.

1.1 PL/SQL, and MS Transact-SQL Components

Both PL/SQL, and MS Transact-SQL have common features, and functionality each in its own respective DBMS. Common DBMS system schema objects are offered in both languages. Components such as indexes, tables, views, database links, built-in procedures, built-in functions, triggers, packages, and synonyms are each able to be implemented in each language, but each using different syntax to do so. Historically, the Oracle DBMS tends to have offered advanced component features first, with MS SQL server following along in years after. To this day, MS SQL server's MS T-SQL still doesn't have the level of capability PL/SQL does, though it has come quite close. Such schema objects as sequences for example aren't available to MS T-SQL, but a similar option can be setup through a custom user built-in function which uses properties built into MS SQL server to simulate an Oracle sequence. Still, today both languages now offer much of the same functionality.

1.2 Purpose and Benefits of Stored Subprograms

Languages such Oracle PL/SQL, and MS Transact-SQL support the ability to setup stored subprograms for the sake of enabling a database administrator all the benefits that come with imperative programming, but within the database environment. Going much beyond the ability of simple queries, updates, or deletes, entire batch processes can be run to perform a combination of multiple standard SQL commands. Stored subprograms enable a common industry professional recommendation to never allow users direct access to the database. Stored subprograms act as a protective layer between the user's input, and the table data in the database. Also, even though it's possible to setup programs using dynamic SQL at the user front end alone, there are great performance benefits to be gained from using a program residing in the DBMS. Making programs at the user front-end requires making lengthy connection calls to pull data from the database to perform operations on the data, after which perhaps sending the information back. Stored subprograms avoid the slow overhead in such a process by allowing all procedural operations to be run on the server itself.

Part 2: Oracle PL/SQL Programs

PL/SQL code is grouped together into structures named as blocks. Even though a grouped structure can be named, it can also be run on its own as what is called an anonymous block. Given a name, such a block is appropriately called a named block.

2.1 PL/SQL Code Block Overview

A typical PL/SQL program exhibits the following structure:

```
declare
     <declarations section>
begin
     <executable commands>
exception
     <exception handling>
end;
```

Though not always, normally the structure of a PL/SQL program contains a Declarations, Execution Commands, and Exceptions section. The Declarations section starts after the declare keyword, and ends with the begin keyword, followed by the Executable Commands section started after the begin keyword, ending with the exception or end keyword should the program not have an exception. If the program does contain an exceptions section, the section starts with the exception keyword, and ends with the end keyword.

The declarations section is the section of code used primarily to define variables, and cursors to be used in the program. Variable datatypes encompass everything from standard scalar types such as integer, character, and timestamp to composite types such as records, tables, and arrays, to large objects data (LOBs). Variables can have constant values, or they can inherit their datatypes from query results.

The Executable Commands section of a program manipulates variables, and cursors setup in the Declarations section. A program block's work begins in the Executable Commands section. It may contain execute commands for any cursors declared, or common conditional logic control statements such as if, else, elsif, case switches, for loops, and/or while loops.

2.1.1 Control Statements

Control statements in PL/SQL are used to implement standard conditional logic operations found in most programming languages. The following are a few general examples of such statements, and their formats.

if, else, elseif

```
if <some condition>
   then <some command>
elseif <some condition>
   then <some command>
else <some command>
end if;
```

Simple Loop

```
loop                  --<--Simple Loop Starts Here
    <loop operations>
exit when <some condition>
end;
```

WHILE Loop

```
while <some condition>
    loop
         <loop operations>
end loop;
```

FOR Loop

```
for i in <Low Value> .. <High Value> --i is a variable
    <loop operations>
end loop;
```

CASE Statements

```
case i     --<--i is a variable
   when <some condition 1>  then <some operation>
   when <some condition 2>  then <some operation>
   when <some condition 3>  then <some operation>
   .
   .
   .
   when <some condition n>  then <some operation>
  else  <some operation>
end case;
```

2.1.2 Cursors

Cursors are a datatype to which SQL select statements can be assigned, and through which information can be manipulated. They're setup in the Declarations section, and initiated in the Executable Commands section of a program block.

Cursor Definition Syntax in a Program Declarations Section

```
declare

cursor   <cursor name>  is  <SQL Query>;
…
```

For Example:

```
declare

cursor cname is select s_attr from sample_table;
…
```

Cursors come with four attributes which can referenced during the Executable Commands section of the program in various contexts that allows checking current cursor status. The four attributes are as follows:

%FOUND	A record can be fetched from the cursor.
%NOTFOUND	No more records can be fetched from the cursor.
%ISOPEN	The cursor has been opened.
%ROWCOUNT	The number of rows fetched from the cursor so far.

The following shows a sample cursor named `e_cursor` with its attribute being used inside the Executable Commands section to setup the conditions for continuing a while loop:

```
…
begin

     while e_cursor%FOUND
          loop
               …
     end loop;
…
end;
```

A common use for a cursor is to retrieve rows resulting from the cursor SQL definition so that the data retrieved can be manipulated in some fashion during program operation. This is mainly done through use of three cursor

implementation statements: open, fetch, and close. The following is sample generic code demonstrating each cursor operation statement during use:

```
declare
   e_variable varchar2;
   cursor e_cursor is select s_attr from sample_table
                      where s_attr2 = 'match';
begin
   open  e_cursor;
   fetch e_cursor into e_variable;
   if c1%NOTFOUND then
      e_variable := 'Not Found';
   end if;
   close e_cursor;
end;
```

As demonstrated in the sample, the open command opens the cursor so the query declared for the cursor is executed, and records to be returned are identified. The query result records, however, aren't actually returned until the fetch command is used. In the case of this code block, the result is returned into the variable `e_variable`. After going through the `if` statement, and the cursor data is no longer needed, the `close` command is called to close the cursor.

2.1.3 Exception Handling

An optional Exception Handling section can be used to address system-related exceptions (errors) encountered during Execution Commands section operations. Upon encountering an error in the Execution Commands section, control of the program is transferred to the operations setup in the Exception Handling section. A sample Exception Handling section format is as follows:

```
declare
   …    <declare setup here>
begin
   …   <program execution commands run here>
exception
   raise_application_error(1234,'Custom error.');
end;
```

2.2 Stored Procedures

Stored procedures are groups of SQL statements brought together to form logical units aimed at performing specified tasks. They encapsulate operation, or query sets executed on a database server. In layman's terms a stored procedure is a simple program stored in an Oracle server, but in its most complex state, a stored procedure can be formed with any combined group of SQL, PL/SQL, and Java statements all in a named block of code which enables one to move code used to enforce business rules from an application to a database.

Stored Procedure Syntax

phrases in [….] are optional

vertical bars such as | represent dividers between available options

```
create [or replace] procedure [schema.] procedure_name
           ([argument_1 [in|out|in out] [nocopy] datatype],
            [argument_2 [in|out|in out] [nocopy] datatype],
             ..................................................... ,
            [argument_n [in|out|in out] [nocopy] datatype])
           [authid {current_user | definer }]
{ is | as }
{ PL/SQL_Subprogram_Body | language }
};
```

`create [or replace]` – if only the `create` keyword is used, the system creates a new procedure, and will not re-create currently existing procedures going by the same name. Including the `replace` keyword enables replacing existing procedures should they exist while still keeping the same previously granted object privileges on the same procedure.

`procedure` – standard keyword letting the system know a procedure is being defined

`[schema.]` – specifies the database schema that will contain the procedure If this phrase is not included, the procedure is created for the current database schema.

`procedure_name` – the custom name provided to identify, and call the procedure

`(...,...,...)` – all the arguments to be set for the procedure are contained in between two paranthesis. Each argument is delimited by a comma.

`argument_n` – the custom name given to one of any *n* number of arguments setup for the procedure.

[in|out|in out] – if `in` keyword alone is specified then a value must be given for the specified argument when the procedure is called. If the `out` keyword is specified then the procedure will pass a value back to its calling environment after the procedure has completed its execution. If the `in out` keyword is specified, then a value must be supplied with the argument, and a value will be returned with the procedure has completed its execution. Should none of these I/O keywords be specified, the default effect is setup the same as if the `in` keyword were used.

[nocopy] – instructs the database to pass the specified argument as fast as possible. Though it's not always guaranteed to do so, this clause can help improve performance when transferring large values like records, index-by tables, or a varray to an `out` or an `in out` parameter. `in` parameter values are always passed as `nocopy.`

datatype – the type of the argument to be sent which can be of any datatype supported by PL/SQL. Length, precision, or scale of the datatype are not specified, because such properties are taken from the argument being passed to the procedure.

authid current_user – creates the procedure with invoker rights, indicating that the procedure runs with the privileges of `current_user.`

authid definer – creates the procedure with definer-rights, indicating that the procedure runs with the privileges of the owner of the schema in which the procedure is located. When no `authid` of any form is specified `authid definer` becomes the default permission setting.

{ is | as } – both `is` and `as` are synonyms. Either one is used once to initiate the Declarations section of the procedure code block.

PL/SQL_Subprogram_Body – the procedure's program body setup in PL/SQL. PL/SQL code blocks within procedures can include any data manipulation language statements, however, they can't include data definition language statements (such as create table).

language – instead of having PL/SQL in the procedure code block, exclusively either java, or c language code can be used, thus enabling the ability to build a procedure employing the advanced features offered by each of those programming languages.

2.3 Stored Functions

Stored procedures, and stored functions are similar in some of their structure, but not completely. Stored procedures, and stored functions differ in how they operate. While a procedure does not have a return value sent back to its caller, a function does. A function can also be referenced directly in queries. There are also advanced handling properties available to functions that procedures don't have.

Stored Function Syntax

- phrases in [….] are optional
- vertical bars such as | represent dividers between available options
- `. . .` N means preceding [….] can be set any N number of times

```
create [or replace] function [schema.] function_name
           ([argument_1 [in|out|in out] [nocopy] datatype],
            [argument_2 [in|out|in out] [nocopy] datatype],
             ..................................................... ,
            [argument_n [in|out|in out] [nocopy] datatype])
           return datatype
           [ authid {current_user | definer } |
            deterministic| parallel_enable
           ] ... N
{ { aggregate | pipelined } using [schema.] implementation_type
| [pipelined] { is | as }
 { PL/SQL_Subprogram_Body | language }
 };
```

`create [or replace]` – if only the `create` keyword is used, the system creates a new function, and will not re-create currently existing functions going by the same name. Including the `replace` keyword enables replacing existing functions should they exist while still keeping the same previously granted object privileges on the same function.

`function` – standard keyword letting the system know a function is being defined

`[schema.]` – specifies the database schema that will contain the function. If this phrase is not included, the function is created for the current database schema.

`function_name` – the custom name provided to identify, and call the function

`(...,...,...)` – all the arguments to be set for the function are contained in between two paranthesis. Each argument is delimited by a comma.

argument_n – the custom name given to one of any n number of arguments setup for the function.

[in|out|in out] – if `in` keyword alone is specified then a value must be given for the specified argument when the function is called. If the `out` keyword is specified then the function will set the argument value back to its calling environment after the function has completed its execution. If the `in out` keyword is specified, then a value must be supplied with the argument. A value for the argument will set during function execution, and returned when the function has completed its execution. Should none of these I/O keywords be specified, the default effect is setup the same as if the `in` keyword were used.

[nocopy] – instructs the database to pass the specified argument as fast as possible. Though it's not always guaranteed to do so, this clause can help improve performance when transferring large values like records, index-by tables, or a varray to an `out` or an `in out` parameter. `in` parameter values are always passed as `nocopy.`

datatype – the type of the argument to be sent which can be of any datatype supported by PL/SQL. Length, precision, or scale of the datatype are not specified, because such properties are taken from the argument being passed to the function.

return *datatype* – return statement specifying the datatype of the value being returned which can be of any datatype supported by PL/SQL. Length, precision, or scale are not specified, because such datatype properties are set by the environment from which the function is called. Boolean parameters, or returns are not allowed. Instead, it is encouraged to use numbers 0, or 1, or strings specifying either 'true', or 'false'.

authid current_user – creates the function with invoker rights, indicating that the function runs with the privileges of `current_user.`

authid definer – creates the function with definer-rights, indicating that the function runs with the privileges of the owner of the schema in which the function is located. When no `authid` of any form is specified `authid definer` becomes the default permission setting.

deterministic – a clause that tells the system to return the same result for the specified function whenever the function is called with the same argument values it was called with previously.

parallel_enable – an optimization setup letting the DBMS know the function can be run from a parallel execution server of a parallel query operation.

aggregate using [*schema*.] – a specifier used to set the function up as an aggregate function, used to evaluate a group of rows, and return one row. This enables the function to be made usable with the SQL `having`, and `order by` clauses. This specifier also enables the function to act as an analytic function (one that works on a query result set). If **[*schema*.]** is not specified, the database assumes the implementation type is the definer's own schema.

pipelined *implementation_type* – instructs the Oracle Database to iteratively return the results of a table function. Table functions are queried with the `table` keyword prior to the name of the function in the FROM part of the query.
Example:

```
select * from table(function_name(...))
```

The database then returns rows as they are produced by the function. If *implementation_type* is not specified then single elements of the function's return collection are returned, instead of the entire collection as a single value. If *implementation_type* is specified then an interface for the `start`, `fetch`, and `close` operations can be predefined (very useful for functions setup in external programming languages such as C++, and Java).

{ is | as } – `is` and `as` are synonyms. Either one is used once to declare the function body.

PL/SQL_Subprogram_Body – the function's program body setup in PL/SQL.

language – just as within the built in procedure, instead of having PL/SQL in the function code block, exclusively either java, or c language code can be used, thus enabling the ability to build a function employing advanced features offered by each of those programming languages.

2.4 Stored Packages

A stored package is a class-styled group of related procedures, stored functions, and other program objects saved together as a single unit in a database. A single group of variables defined within a package can be used by multiple procedures, or functions also within the same package. Commands can also be setup to run within packages anytime the package is called, despite any package function, or procedure members also run in the same call.

Stored Package Syntax

- phrases in [….] are optional
- vertical bars such as | represent dividers between available options

```
create [or replace] package [schema.] package_name
[authid {definer | current_user} ]
{ is | as }
package specification;
```

`create [or replace]` – same as before if only the `create` keyword is used, the system creates a new package, and will not re-create currently existing packages going by the same name. Including the `replace` keyword enables replacing existing packages should they exist while still keeping the same previously granted object privileges on the same package.

`package` – standard keyword letting the system know a package is being defined

`[schema.]` – specifies the database schema that will contain the package. If this phrase is not included, the package is created for the current database schema.

`package_name` – the custom name provided to identify, and call the package

`authid current_user` – creates the package with invoker rights, indicating that the package runs with the privileges of `current_user.`

`authid definer` – creates the package with definer-rights, indicating that the package runs with the privileges of the owner of the schema in which the package is located. When no `authid` of any form is specified `authid definer` becomes the default permission setting.

`{ is | as }` – `is` and `as` are also synonyms when used in packages. Either one is used once to declare the body of the package.

`package specification` – also much like before, the function's program body setup in PL/SQL. Instead of having PL/SQL in the function code block, exclusively either java, or c language code can be used, thus enabling the ability to build a function employing advanced features offered by each of those programming languages.

2.5 Triggers

A trigger is a definition for an action the database takes when some database-related event occurs. Like procedures, triggers can be used to enforce complex business rules. However, triggers are unique in that they also help to automatically audit changes to data, and the execution of triggers happens automatically so the user need not be aware. Triggering events can include `inserts`, `updates`, `deletes`, specific column updates, data definition language commands, and other database related events such as shutdowns, or logins. They're a great supplement to referential integrity, though it's always recommended that declarative referential integrity be relied on first, and foremost.

In order to run operations on data before, and after the trigger event, two keywords are used to reference data. These are the `new`, and `old` keywords, where `old` is used to refer to data prior to the trigger event, and `new` is used to refer to data after the trigger event. Syntax for these keywords differs depending on the context in which they're used.

`new`, and `old` Keywords Used in a Conditional Statement

```
...
while (new.s_attr < old.s_attr)
    begin
         <loop operations>
    end;
...
```

`new`, and `old` Keywords Used in an Insert Statement to Insert Old and New Values

```
...
insert into s_table (attr1, attr2) values
                    (:old.s_attr1, :new.s_attr2);
...
```

Trigger Syntax

- phrases in [….] are optional
- vertical bars such as | represent dividers between available options

```
create [or replace] trigger [schema.] trigger_name
{ before | after | instead of }
{ dml_event_clause
| { ddl_event [or ddl_event] ...
  | database_event [or database_event] ...
  }
  on { [schema.] schema | database }
}
[when ( condition ) ]
{ pl/sql_block | call_procedure_statement };
```

`create [or replace]` – similarly, once again, if only the `create` keyword is used, the system creates a new trigger, and will not re-create currently existing triggers going by the same name. Including the `replace` keyword enables replacing an existing trigger of the same name without having to drop it first.

`trigger` – standard keyword letting the system know a trigger is being defined

`[`*`schema`*`.]` – specifies the database schema that will contain the trigger. If this phrase is not included, the trigger is created for the current database schema.

`trigger_name` – the custom name provided to identify, and call the trigger

`before` – causes the database to fire the trigger before the database executes the event causing the trigger to fire. If it's a row trigger, the trigger fires right before each row is changed. `before` triggers cannot be specified on a view, or object view. The `:old` value cannot be written to, only the `:new` value.

`after` – database fires the trigger after the trigger event is executed. In the case of rows, the trigger is fired after each row event. `after` triggers also cannot be specified on views, or object views. Both the `:old` and `:new` values cannot be written to.

`instead of` – database fires the trigger instead of executing the trigger event. Such events are valid for data manipulation language events on views, and not valid for data definition language, or database events. `instead of` triggers take precedence over performing a data manipulation language event on an inherently updatable view containing the trigger. In such a view, but of hierarchical form, subviews do not inherit the trigger. Unlike the other two types of triggers `instead of` triggers only apply to views, never tables. Such views also allow both `:old` and `:new` values to be read, but not written to.

`dml_event_clause` – specifies the type of data manipulation language event on which the trigger is to fire. The DML events include `delete`, `insert`, and `update`. Any OR combination of the three events can be specified for a trigger to fire. The `update` event can even be set to trigger upon update of single columns in a relations, not just the entire row.

ddl_event – specifies the data definition language events on which the trigger is to fire. An OR combination of such events can be setup similar to DML events. DDL events through a PL/SQL procedure cannot be setup with triggers. Of course, DDL events include events such as create, rename, drop, etc.

database_event – specifies the database states on which a trigger can be fired. OR combinations of database states can also be specified. Common database events for setting up triggers include logon, logoff, startup, shutdown, suspend, etc.

on [*schema*.]*schema* – specifies the schema on which the trigger will be defined. Users connected as *schema* specified are able to initiate the event that sets off the trigger.

on database – specifies an entire database in which the trigger can be fired. Any database user that initiates the triggering event can fire the trigger.

when (condition) – an SQL condition which must be satisfied for the trigger to fire. `new` and `old` are not preceded by a colon (:), because they're not considered bind variables when part of conditional statements.

pl/sql_block – this is the PL/SQL block of code executed to fire the trigger.

call_procedure_statement – an optional call to a stored procedure to be used instead of inline trigger code as a PL/SQL block. Referencing columns of tables on which the trigger is defined requires specification of `:new` and `:old`.

Part 3: Oracle PL/SQL Subprograms through Example

Now that the ideas behind Oracle PL/SQL Subprograms have been covered, a few stored program examples pertaining to the Every Meal Delivery project will be described.

3.1 Inserting Records with a Stored Procedure

The following is a stored procedure named HV_NewCustomer setup to insert a new customer with all his/her basic information into the main customer table. The complete set of attributes required to enter the information for a new customer is passed to the procedure, each attribute being a separate argument. Upon receiving the information, the procedure runs a simple insert command.

```
procedure HV_NewCustomer (aCID in number, aFN in varchar2,
                          aMI in char,    aLN  in varchar2,
                          aPH in number,  aMS in number) as
begin
      insert into HV_Customer (CustomerID, cfirst,
                               cMidInitial, cLast,
                               Phone, MemberStatus)
                       values (aCID, aFN, aMI, aLN, aPH, aMS);
end;
```

3.2 Deleting a Record with a Stored Procedure

This procedure, named HV_DelCustomer is a simple way to delete a customer from the customer relation by passing along the customer's ID number, represented through the primary key in the relation. After receiving the ID number, a command is run to delete the row matching the ID specified in the HV_Customer relation.

```
procedure HV_DelCustomer (row in number)
as
begin
     delete from HV_Customer where CustomerID = row;
end;
```

3.3 Returning the Average of the Specified Highest, or Lowest with a Stored Function

Stored function HV_CXAVG takes in two parameters: a number, and a character. The purpose of this function is to grab either the highest, or lowest (highest or lowest determined by the second argument sent) priced `ncount` (`ncount` being the first argument sent in the function) number of combo items from the HV_Item relation. The function then calculates the average price of the retrieved combo items, and returns the calculated average as the function return argument. Should there be an input error, either an exception is raised, or a string is printed informing the function caller of a wrong character having been sent.

```
function "hv_cxavg" ("ncount" in number, "drection" in char)
return number authid current_user
      --declaration section starts here
is
       tally number;      --total combo number tracker
       avrg  number;      --average number variable to be returned by function
       rnum  number;      --records to be counted

       cursor cnt is      --cursor to count the total number of combos available
                    select count(i.itemid) from hv_item i
                    where (iname like 'combo-%');

       cursor mx is       --cursor to calculate the average of the highest priced combos
       select avg(f.price)
       from (select i.*
           from (select itemid, price
                 from hv_item
                 where (iname like 'Combo-%')
                 order by price desc) i
       where rownum <= rnum) f;

       cursor mn is       --cursor to calculate the average of the lowest priced combos
       select avg(f.price)
       from (select i.*
           from (select itemid, price
                 from hv_item
                 where (iname like 'Combo-%')
                 order by price) i
       where rownum <= rnum) f;

begin --executable commands section starts here

       open cnt;
       fetch cnt into tally;    --retrieve the total number of combos available
       close cnt;

    rnum := ncount;

       if rnum > tally          --if item request # is too high set the request to max
          then rnum := tally;
       end if;

   --case statement gets lowest average if L/l was sent, highest average if H/h was sent
   case drection
   when 'l' then
           open   mn;
                  fetch  mn into avrg;
           close  mn;
```

```
   when 'L' then
           open   mn;
                 fetch  mn into avrg;
           close  mn;
   when 'h' then
           open   mx;
           fetch  mx into avrg;
           close  mx;
   when 'H' then
           open   mx;
           fetch  mx into avrg;
           close  mx;
--message is printed if the wrong highest/lowest price range specifying character is sent
   else    dbms_output.put_line ('highest or lowest incorrectly specified.');
           avrg := 0;
   end case;

  return avrg;

--in case of program error, print an error message
exception
when others then
       raise_application_error (-1234,'Danger! There's been an error.');

end;
```

3.4 A Data Manipulation Language Trigger

The following trigger titled HV_Customer_Trigger is performed before the delete or update of a customer record in the HV_Customer relation. The trigger checks for whether either a delete or update event is to be run on the HV_Customer relation. If it's an update event, the old record is concatenated together into one string, and the new record is concatenated into another string. The two strings are then inserted into the HV_LogTable relation as a new record. If it's a delete event, only the old record is concatenated, then stored as a new record in the HV_LogTable relation. The insert or delete operation is then completed.

```
trigger "hv_customer_trigger"
before
delete or update of "customerid", "cfirst", "cmidinitial",
                    "clast", "phone", "memberstatus"
on "hv_customer"
for each row
--declare

begin -- executable part starts here
if updating then
       insert into hv_logtable values (hv_ltsqnce.nextval, :old.customerid || ' ' ||
                                       :old.cfirst || ' ' || :old.cmidinitial || ' ' ||
                                       :old.clast || ' ' || :old.phone || ' ' ||
                                       :old.memberstatus,
                                       :new.customerid || ' ' || :new.cfirst || ' ' ||
                                       :new.cmidinitial || ' ' || :new.clast || ' ' ||
                                       :new.phone || ' ' || :new.memberstatus);
end if;

if deleting then
       insert into hv_logtable values (hv_ltsqnce.nextval, :old.customerid || ' ' ||
                                       :old.cfirst || ' ' || :old.cmidinitial || ' ' ||
                                       :old.clast || ' ' || :old.phone || ' ' ||
                                       :old.memberstatus, null);
end if;
null;
end;
```

Phase V

Graphical User Interface Design and Implementation

Part 1: Daily Users, and Activities

The business model on which this project is based involves one user entity of type employee, but with three different roles: Delivery Driver, Dispatcher, and Manager. Though any employee can play a part in any of these roles, not everyone will. Most employees embody the role of Dispatcher, or Delivery Driver, though never at the same time. For example, on one day employee Stan L. Marsh could be a dispatcher setting up customer orders he gets over the phone for two hours; then go out as a delivery driver for the rest of the day. Anytime Stan is doing something, it's only as part of one role at any one time. The only role that can play multiple roles at the same time is that of Manager, left to either one employee, or a small number of employees for larger size meal delivery franchises.

1.1 Dispatcher Role

Dispatchers are employees in the delivery call center who receive calls from customers over the phone. When a customer calls, an employee logged into the system with the role of dispatcher requests the customer's phone number to look up the customer's information for an order.

Should the customer not already be in the system, the dispatcher records the new customer's name, and phone, as well as any multiple number of addresses the customer may have as destinations for deliveries. If the customer wants to purchase an Every Meal Delivery membership, the dispatcher notes the purchase intent in the order to be made. Customer information as well as the customer's chosen location for delivery is established by the dispatcher to the system as part of the order when the order is being started.

After entering the customer information, the dispatcher then asks the customer the restaurant from which the customer would like to order. The customer may list any number of restaurants supported in the system, but each restaurant is entered one at a time. An example of this would be the customer first naming Taco Bell as a location from which the customer wants to order. The dispatcher then selects Taco Bell from the list of restaurants available in the system. After which, the dispatcher selects the address of the Taco Bell nearest to the customer's delivery destination location. Having the menu loaded into the system for the chosen franchise, the dispatcher selects the menu items to add to the order as the customer requests them over the phone. When the customer is done selecting the items for the specific restaurant, the dispatcher adds the item list to the order. If the customer then requests more items from a different restaurant, the order addition process is repeated for the other restaurant.

When the dispatcher is done adding the list of customer requested items, a current order and delivery driver nearest to the delivery destination area is assigned to deliver the order. The dispatcher sets the order as entered into the system, though the order isn't actually complete until it's delivered.

Beside the main function of setting up customers, and orders to be delivered, dispatchers also have the ability to look up prior customer orders made, though they're not allowed to alter them.

1.2 Delivery Driver Role

At any time during the day the majority of employees logged in as dispatchers will be able to switch the current role of dispatcher to that of driver should a request to do so be made by the manager. This can also be done either through pre-scheduling for the week by the manager, or at the beginning of the work day.

An employee assigned as delivery driver has to keep track of mileage used to deliver orders, because gas is reimbursed by the delivery business. All delivery driver employees use their own vehicles to make deliveries, and in order to keep track of mileage, drivers log in the car's initial total miles at the start of the day's delivery assignment, and the total miles at the end of the day's delivery assignment. The system then calculates each employee's total miles run for the day.

Drivers en route carry with them a mobile device with a delivery system application connected back to the main system. The program carried along with the driver lists the driver's delivery tasks to be completed, including the items to be ordered, the address locations from where the items are to be purchased, as well as the customer's delivery destination address. As the driver picks up the orders, and delivers them, the driver updates order statuses. After delivering an order the driver marks the order off as complete. The program also lists other orders in real time which may be assigned by a dispatcher while the driver is en route. Drivers go about completing as they get them in orders queued in the mobile device application.

1.3 Manager Role

Managers are an employee type that can take on both the role of driver as well as dispatcher. A manager has the ability to add/edit employees in the system, look up action reports on every employee, and add/edit restaurants, franchises, and sales items in the system. If the delivery business takes on a large enough size it may even be necessary to have more than one manager employee.

Part 2: Relations, and Subprograms

Even though the scope of this project encompasses functionality for three different entity types, only a major part of the dispatcher role will be instituted. The components of adding a customer as well as customer addresses along with setting up items in the order will employ stored procedures involving relations in the database.

2.1 The Customer Entry Process

Relations Involved

Relations involved in the customer entry process are HV_CUSTOMER, and HV_CSTMRADDRESS. Entering a new customer requires entering not only the basic customer information to be stored in the HV_CUSTOMER relation, but also any number of addresses a customer may have to be stored in the HV_CSTMRADDRESS relation.

Procedures Involved

HV_NewCustomer

```
procedure HV_NewCustomer (aFN in varchar2,   aMI in char,
                          aLN in varchar2,   aPH in number,
                          aMS in number,     aReturn out number)
as
       cursor cSeq is
       select HV_SQ_Customer.CurrVal from Dual;
begin
       insert into HV_Customer (CustomerID, cfirst, cMidInitial, cLast, Phone,
                                MemberStatus)

       values (HV_SQ_Customer.NextVal, aFN, aMI, aLN, aPH, aMS);

open cSeq; fetch cSeq into aReturn; commit; close cSeq;

END;
------------------------
```

This procedure involves sending to it a new customer's first name, middle initial, last name, phone number, member status number, and a reference variable for a return value. The main customer information passed to this procedure is stored in the HV_Customer relation. What is really noteworthy in this procedure is that it uses a sequence object named `HV_SQ_Customer` to automatically assign the new record a unique primary key. The next unique primary key value is assigned in the insert statement through use of the `HV_SQ_Customer.NextVal` statement which calls the sequence object's next value function to assign the next unique sequence value. Another important thing of note in this procedure is that it uses a sequence specific SQL statement through a cursor to return the new sequence value. The SQL statement is defined with the `HV_SQ_Customer.CurrVal` sequence function used to return the current value of the sequence. Even though the

cursor is defined to return the current sequence value prior to the next sequence value assignment, the cursor isn't called until after the next sequence number has already been assigned. This way it is ensured that the new customer entry identification number will be returned.

HV_SP_ADDADDRESS

```
PROCEDURE "HV_SP_ADDADDRESS" (aST in varchar2,    aAPT in varchar2,
                              aCity  in varchar2, aZip in number,
                              aCID in number,
                              aReturn out number)
is
       cursor aSeq is
       select HV_SQ_CSTADDRESS.CurrVal from Dual;
begin
       insert into HV_CSTMRAddress (AddressID, Street, Apt, City, Zip,
                                                                  CustomerID)
       values (HV_SQ_CSTADDRESS.NextVal, aST, aAPT, aCity, aZip, aCID);

open aSeq; fetch aSeq into aReturn; commit; close aSeq;

END;
------------------------
```

The information necessary to store a new customer's address information is passed to this procedure. This includes the street, optional apartment, city, zip code, and the customer identification to which the new address is assigned. All this information is stored in the HV_CSTMRADDRESS relation. This procedure uses the return value from the previous procedure to assign the new customer a new address. Like the previous procedure this one also uses a cursor to return the new unique address value.

2.2 The Order Setup Process

Relations Involved

The order setup process involves use of the HV_CUSTOMER, HV_CSTMRADDRESS, HV_RESTAURANT, HV_FRANCHISE, HV_OFFERS, and HV_ITEM relations. Setting up a new order involves first choosing the customer, and the customer delivery address, then choosing the restaurant, franchise location, and menu items related to the chosen franchise location.

Procedures Involved

HV_SP_GETCUSTOMERS

```
PROCEDURE "HV_SP_GETCUSTOMERS" ("RCLIST" OUT SYS_REFCURSOR)
AUTHID CURRENT_USER IS

BEGIN -- executable part starts here
       OPEN RCLIST FOR SELECT * FROM HV_CUSTOMER;

END;
-----------------------
```

This procedure is pretty straight forward in its function. It returns every record from the HV_Customer relation through use of a simple query. There are a few of these type of procedures used in the program. The thing to note about these kinds of stored procedures is that the record set for the query result is returned through use of a SYS_REFCURSOR type parameter argument. In this stored procedure the cursor return type is named RCLIST, and it's opened for return with the following syntax:

```
       OPEN RCLIST FOR <AN SQL QUERY>
```

HV_SP_GETUADDRESS

```
PROCEDURE "HV_SP_GETUADDRESS" ("ACID" IN NUMBER, "RALIST" OUT SYS_REFCURSOR)
AUTHID CURRENT_USER IS

BEGIN -- executable part starts here

       OPEN RALIST FOR SELECT * FROM HV_CSTMRADDRESS
                       WHERE HV_CSTMRADDRESS.CUSTOMERID = aCID;

END;
-----------------------
```

This procedure is much the same as the previous one, except that it's passed an HV_CUSTOMER record primary key in order to search the customer identification foreign key column in the HV_CSTMRADDRESS relation for matches so address records pertaining to the specified customer can be returned through use of a return SYS_REFCURSOR type parameter argument.

HV_SP_GETRESTAURANT

```
PROCEDURE "HV_SP_GETRESTAURANT" ("RRLIST" OUT SYS_REFCURSOR)
AUTHID CURRENT_USER IS

BEGIN -- executable part starts here

       OPEN RRLIST FOR SELECT * FROM HV_RESTAURANT;

END;
-----------------------
```

HV_SP_GETRESTAURANT is pretty straight forward in that its setup to return every restaurant record from the HV_RESTAURANT relation through use of an SQL query combined with a SYS_REFCURSOR type parameter argument.

HV_SP_GETFADDRESS

```
PROCEDURE "HV_SP_GETFADDRESS" ("ARID" IN NUMBER, "RFALIST" OUT SYS_REFCURSOR)
AUTHID CURRENT_USER IS

BEGIN -- executable part starts here

       OPEN RFALIST FOR SELECT * FROM HV_FRANCHISE
                        WHERE HV_FRANCHISE.RSTID = aRID;

END;
------------------------
```

HV_SP_GETFADDRESS functions very similar to stored procedure HV_SP_GETUADDRESS in that it's given a key from a different table to match in the foreign key column of another table. In this case, the stored procedure is passed a restaurant identification number. The procedure returns all records that have a matching restaurant number in the foreign key column of the HV_FRANCHISE relation. Again, an SQL query combined with a SYS_REFCURSOR type return parameter argument is used.

HV_SP_GETFİTEMS

```
PROCEDURE "HV_SP_GETFITEMS" ("AFID" IN NUMBER, "RILIST" OUT SYS_REFCURSOR)
AUTHID CURRENT_USER IS

BEGIN -- executable part starts here

       OPEN RILIST FOR SELECT I.*
                       FROM HV_FRANCHISE F INNER JOIN
                       HV_OFFERS O ON F.FID = O.FID INNER JOIN
                       HV_ITEM I ON O.ITEMID = I.ITEMID
                       where f.fid = AFID;
END;
------------------------
```

HV_SP_GETFITEMS incorporates much the same style of the previous stored procedures except that it also includes a bit more involved SQL query used to join three relations: HV_FRANCHISE, HV_OFFERS, and HV_ITEMS so the item list from the franchise specified by the IN parameter attribute AFID can be returned in the SYS_REFCURSOR type OUT return parameter argument RILIST.

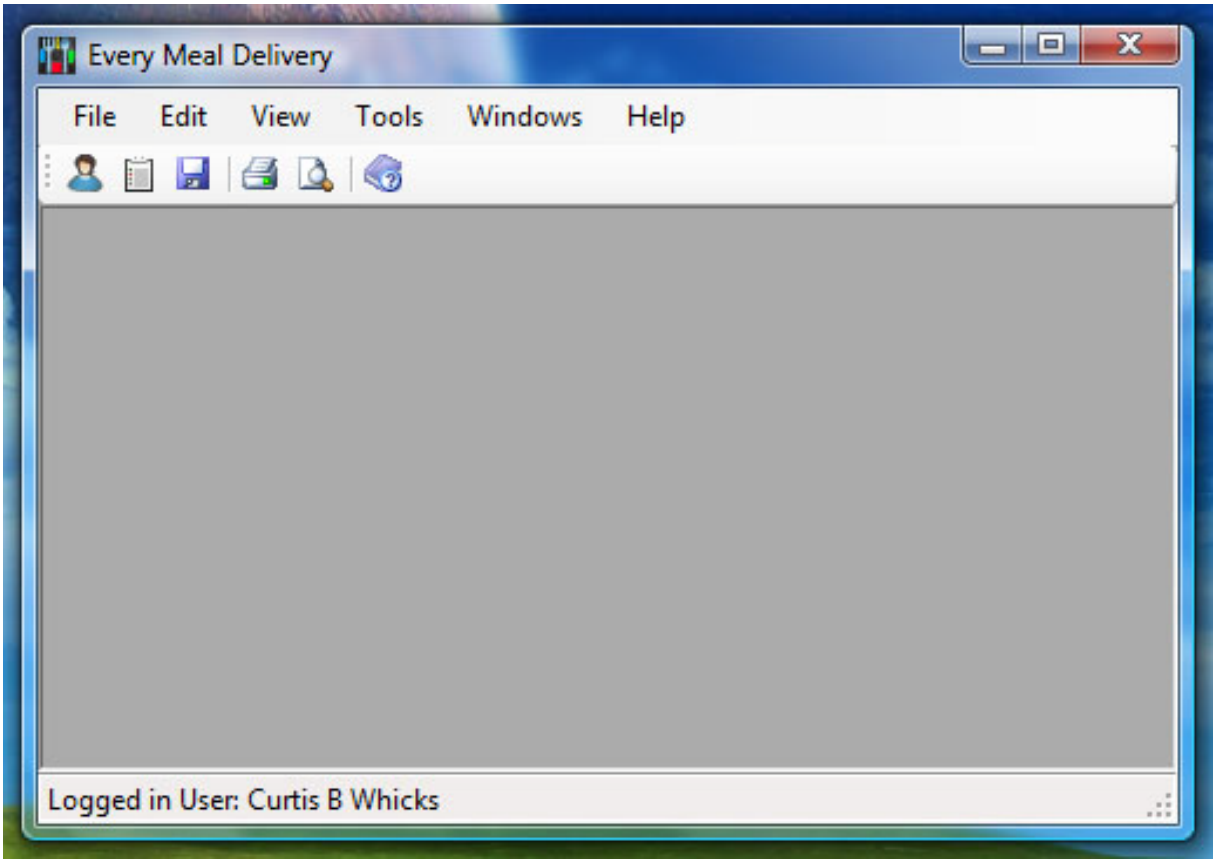
Part 3: Screenshots of the Application

The main goal of a graphical user interface is ease-of-use. A user must be able to perform tasks required by the organization without having to sort through any sort of confusing interface. The easier it is to understand how to use a program the more the user can focus on getting more done. The basic idea is to make a graphical stable user interface which makes it easy to do more, because it's intuitive, while still enforcing business requirements.

This section shows screenshots of the basic dispatcher Every Meal Delivery system graphical user interface front end application with descriptions of how it's intentionally setup to guide the user along the process of properly setting up customers, and orders for customers. Not all screenshots will be set to 100% scale.

3.1 Application Style and Main Menu

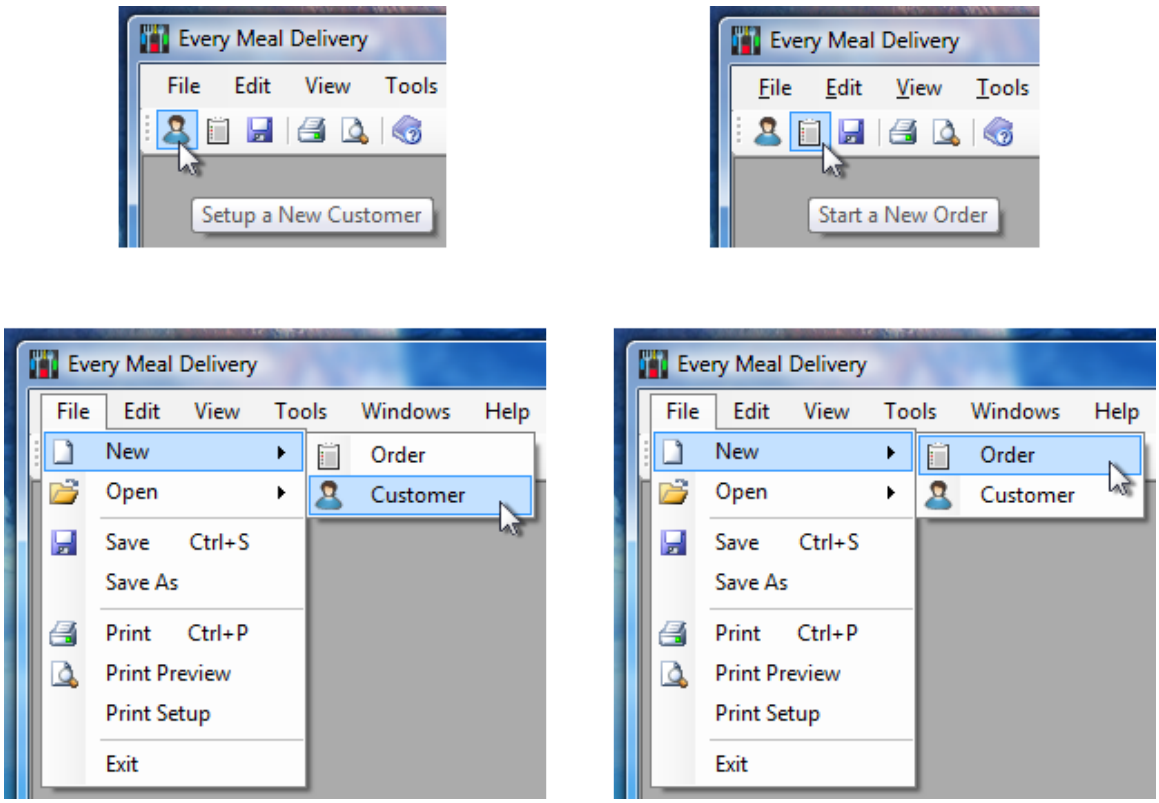
The first thing to note about this application is that it uses a Multiple Document Interface format, meaning it's setup with a dedicated workspace to which forms related to the application are restricted. This is useful because it ensures a theme is setup for the application, making organization of business related tasks easier.



Basic Application Workspace

The application has an extensive use of common menu bar options, listed as File, Edit, View, Tools, Windows, and Help. Not all of these menu options are implemented, however, ideally, such basic menu options are useful things to implement in a production application. The same goes for the icon toolbar strip below the main menu bar.

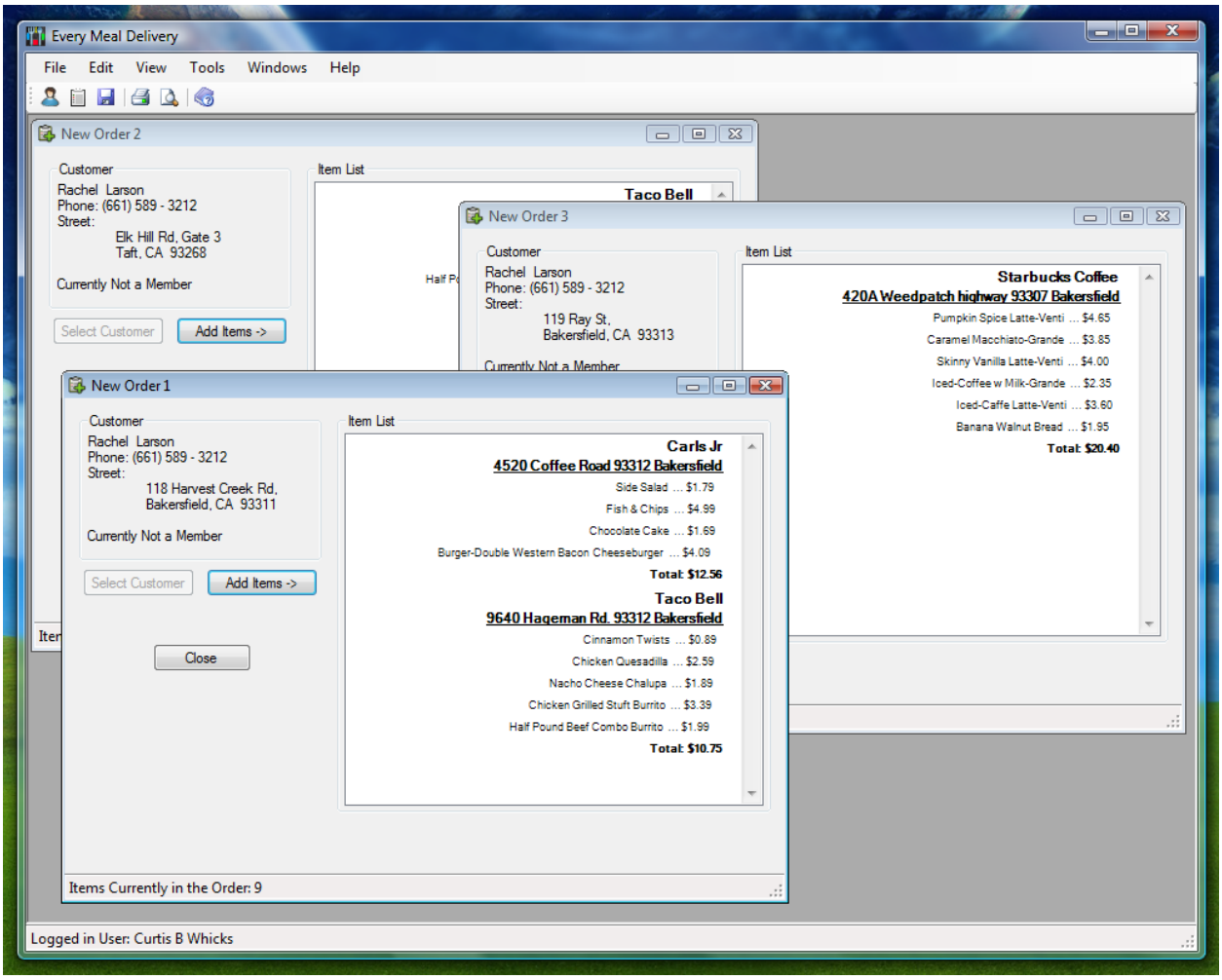
The two things that are implemented in this application are the new customer, and new order options. Both of these can be started either by clicking the new customer, or new order icon in the toolbar (represented by the blue shirt bust, or the clipboard), or selecting those options in the menu bar through File → New → Customer or File → New → Order.



Selecting Options either through the Toolbar or the Menu Bar

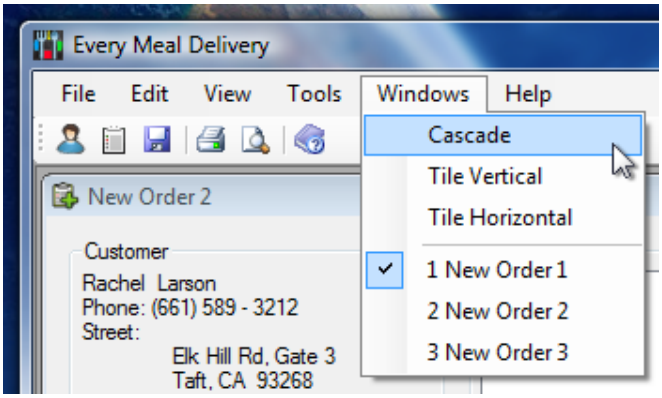
The icons in the toolbar are labeled with their function through use of ToolTip text. This helps to clarify to the user the purpose of the icons. Menu bar options are also given the same icon pictures as the toolbar for the sake of interface consistency.

An advantage of having a multiple document interface application is being able to have the program help arrange the user's workspace. The following shows three order forms opened up for customer Rachel Larson who wants three separate orders to be delivered to three separate locations.



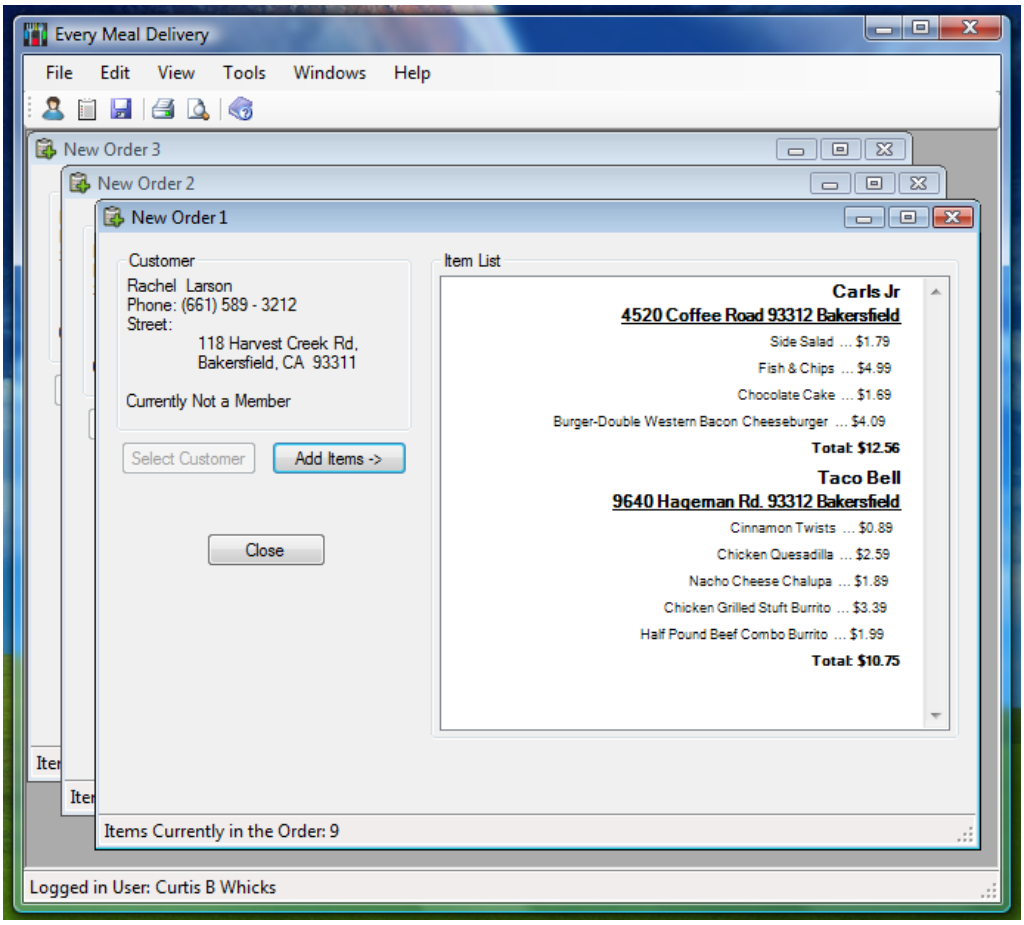
Three Order Documents in the Application Workspace

The application workspace can be corner selected and expanded to fit the order forms so a dispatcher employee can more easily see them, or the menu bar windows organization features can be used to help keep things organized.



Selecting Cascade from the Menu Bar

Selecting Windows → Cascade for example will automatically organize the windows in the workspace so they become easier to manage.



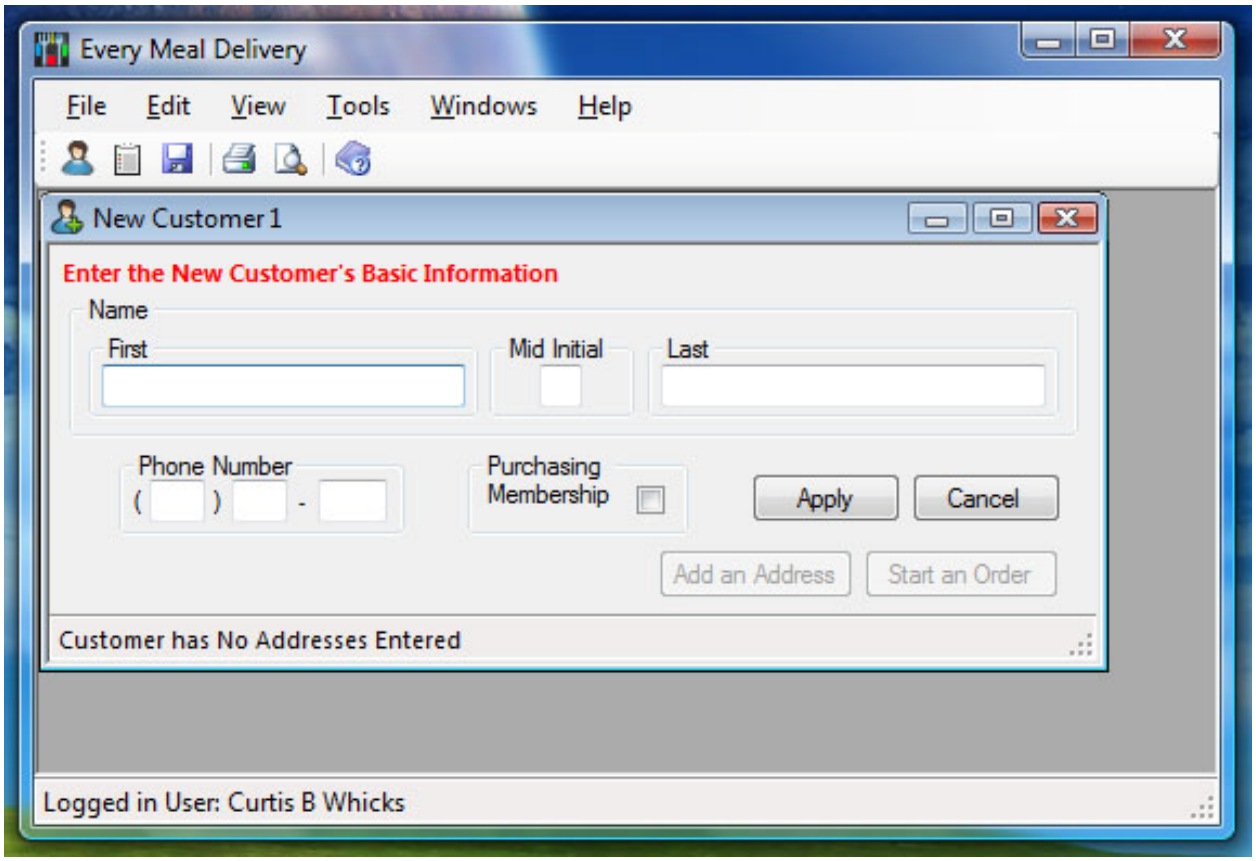
Order Documents Arranged in Cascade Fashion

Selecting Windows from the main menu also allows users the option of bringing whatever document desired to the front of the workspace. The pictures above have the New Order 1 document selected to the front.

The pictures above also show how the application numbers new document windows as they're initiated to help the dispatcher keep track.

3.2 The New Customer Forms

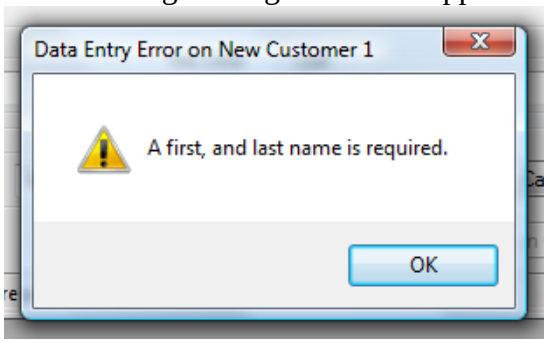
A major part of being a dispatcher employee is setting up new customers in the Every Meal Delivery system as customers call in. To do this a dispatcher can bring up the new customer form by clicking the new customer icon in the toolbar. Doing so brings up the following form:



Blank New Customer Entry Form

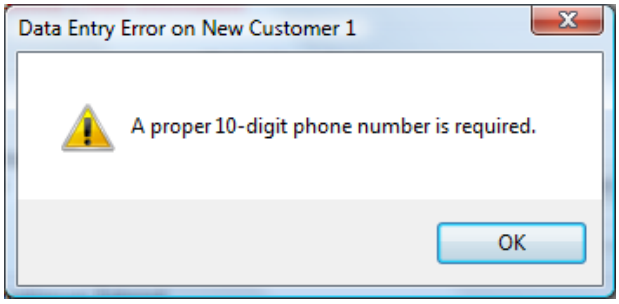
Whenever a dispatcher opens the above form, it becomes clear what is required from a new customer in the system. A dispatcher asks a customer for a first name, middle initial, last name, complete phone number, and asks the customer whether he/she would like to purchase a membership for discount delivery rates. After filling in the required information, the dispatcher may then click the Apply button to lock in the information in order to proceed.

The new customer form is setup to check data entered to make sure it follows the correct entry format. The system ensures no numbers may be entered in any of the name fields, and no letters may be entered in any of the number fields. If a dispatcher forgets to put in a first, or last name the following message box error appears.



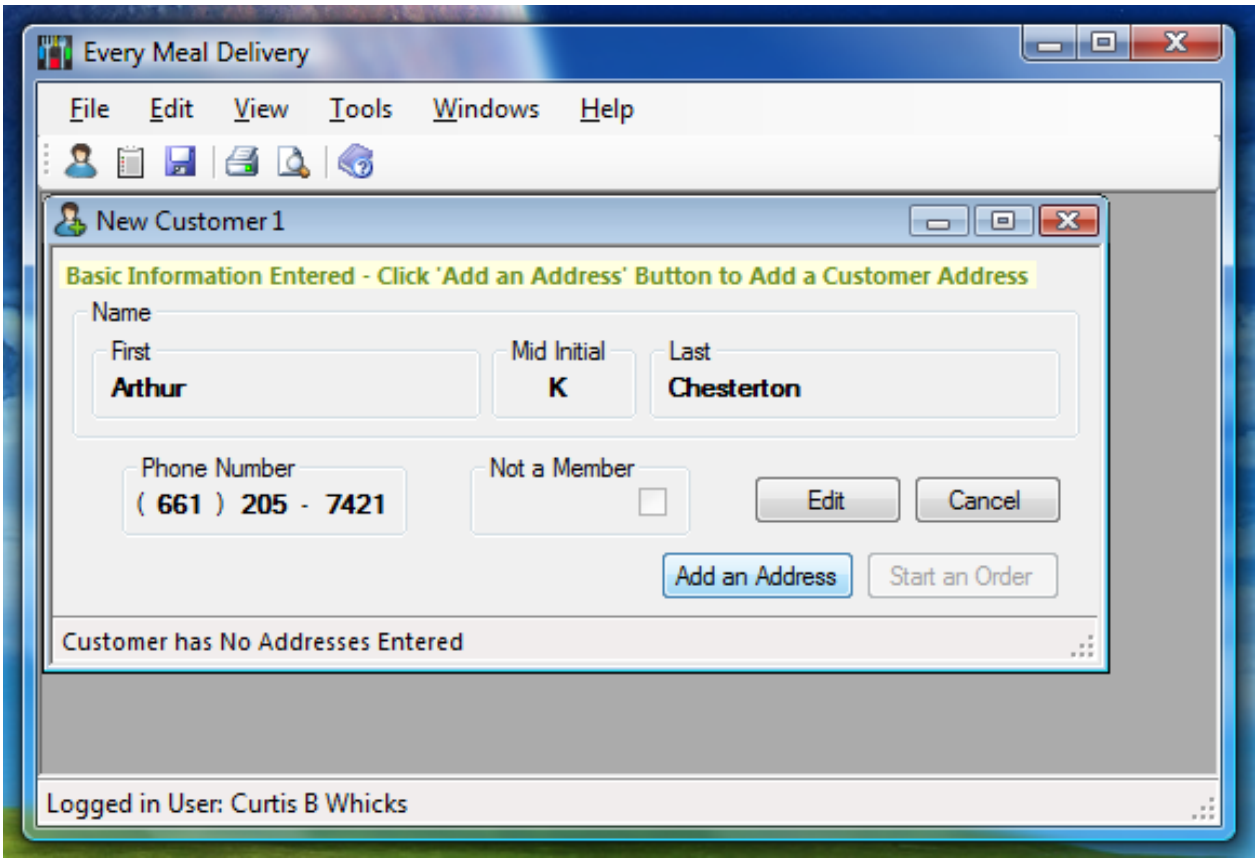
Name Entry Error

If the dispatcher enters correctly enters the name areas, but incorrectly enters the phone number, then another message box error appears.



Phone Entry Error

Correctly entering a new customer's basic information, and clicking the Apply button locks in the information, and tells the dispatcher to click the Add an Address button to enter in a new address for the customer.

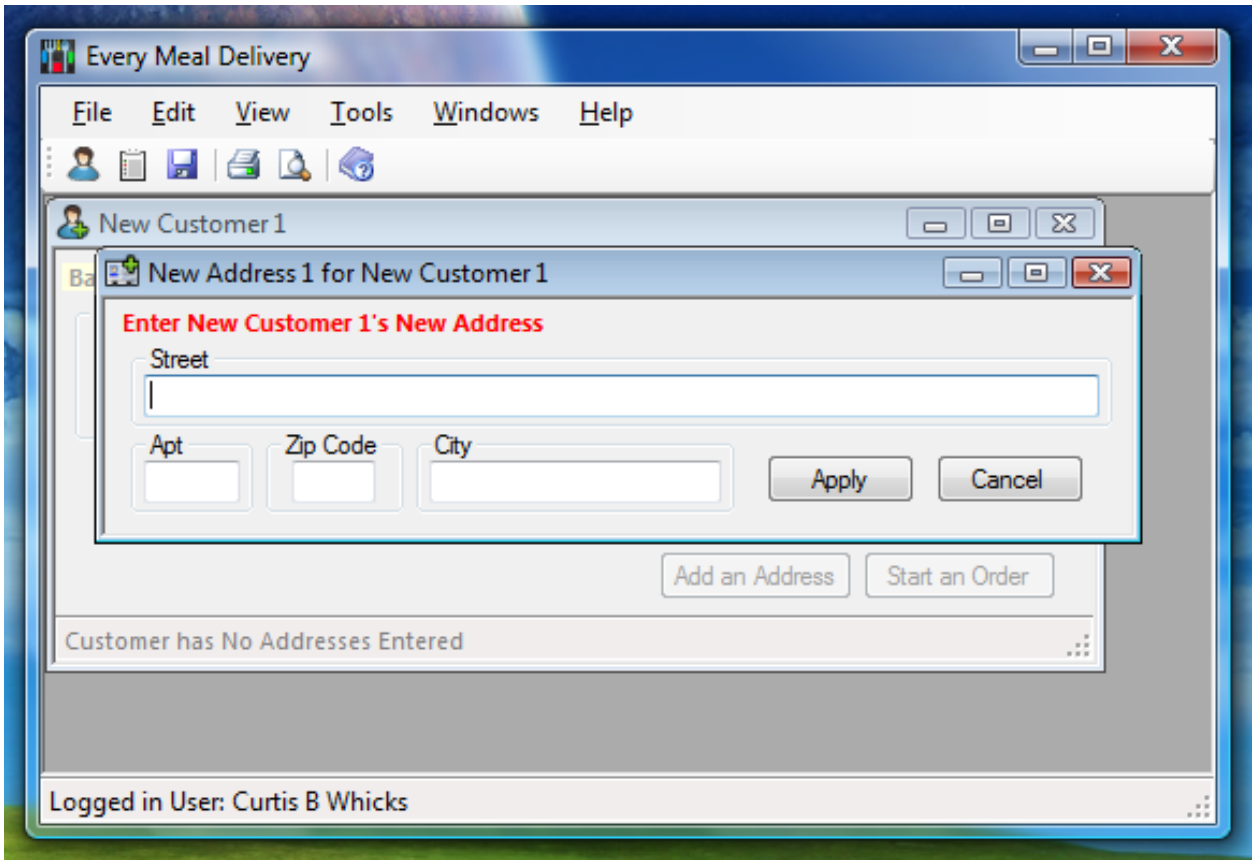


New Customer Basic Information Locked and Ready

The Apply button also switches to an Edit button the dispatcher can still click to edit the customer's basic information. However, clicking the Edit button also disables the Add an Address button, thus ensuring the dispatcher follows the business orders of operation for

customer information entry. The Add an Address button is only enabled after proper basic customer information has been locked in.

The next step involves adding a new delivery address for the customer. After the basic customer information has been locked in, the dispatcher clicks the Add an Address button, and the new address form comes up.

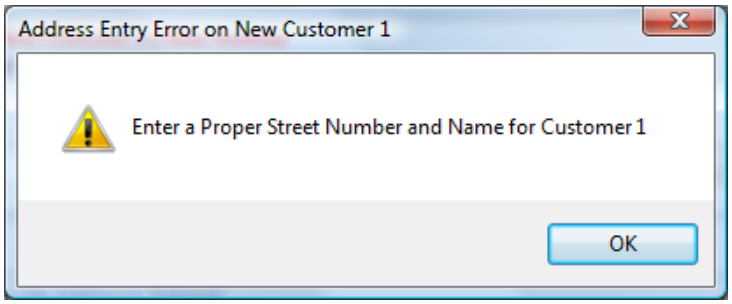


Blank New Address Form

Something of special note to take into account that will be coming up a few times in this program is that when a sub-form comes up (such as the new address form coming up here from the new customer form), the parent form becomes disabled (turned to light gray) so it cannot be selected, or brought into focus above the child form. This way, the user knows to complete everything that needs to be done on the child form before going back to the parent form. In this case, the new address form must be filled in or canceled before the user can go back to dealing with the new customer's basic form.

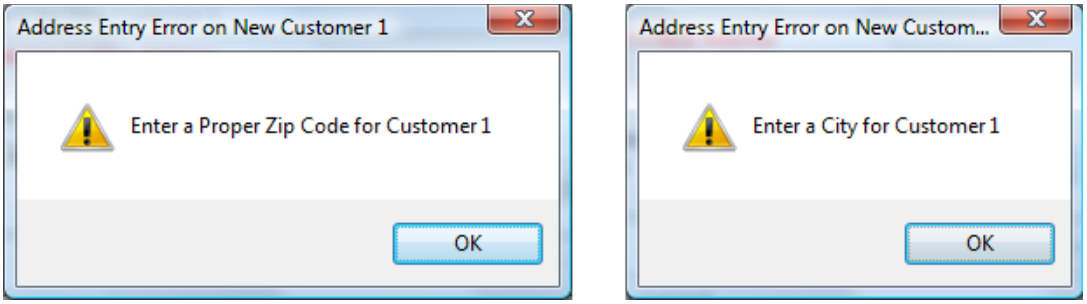
Just as the basic new customer information form did type entry checking, so does the address entry form. With the address entry form, a legitimate street number of a somewhat correct format must be entered. Entering a number alone, or letters alone in the Street field brings up a data entry error Message Box. The correct format for address entry is a few numbers followed by a word, and any other alphanumeric characters after

that. No special characters are permitted in the Street text field. Failure to enter in correctly formatted street information results in the following Message Box error.



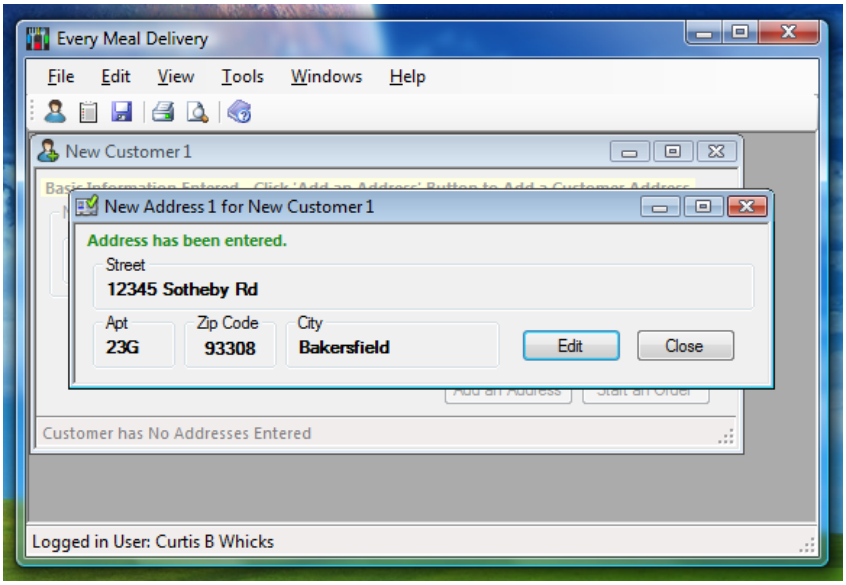
Street Entry Error

To help enforce data entry business rules the same error prompts are brought up should the dispatcher enter in a correct Street, but not a correct zip code, or enter in a correct street, and zip code, but not a correct city.



Zip and City Entry Error Prompts

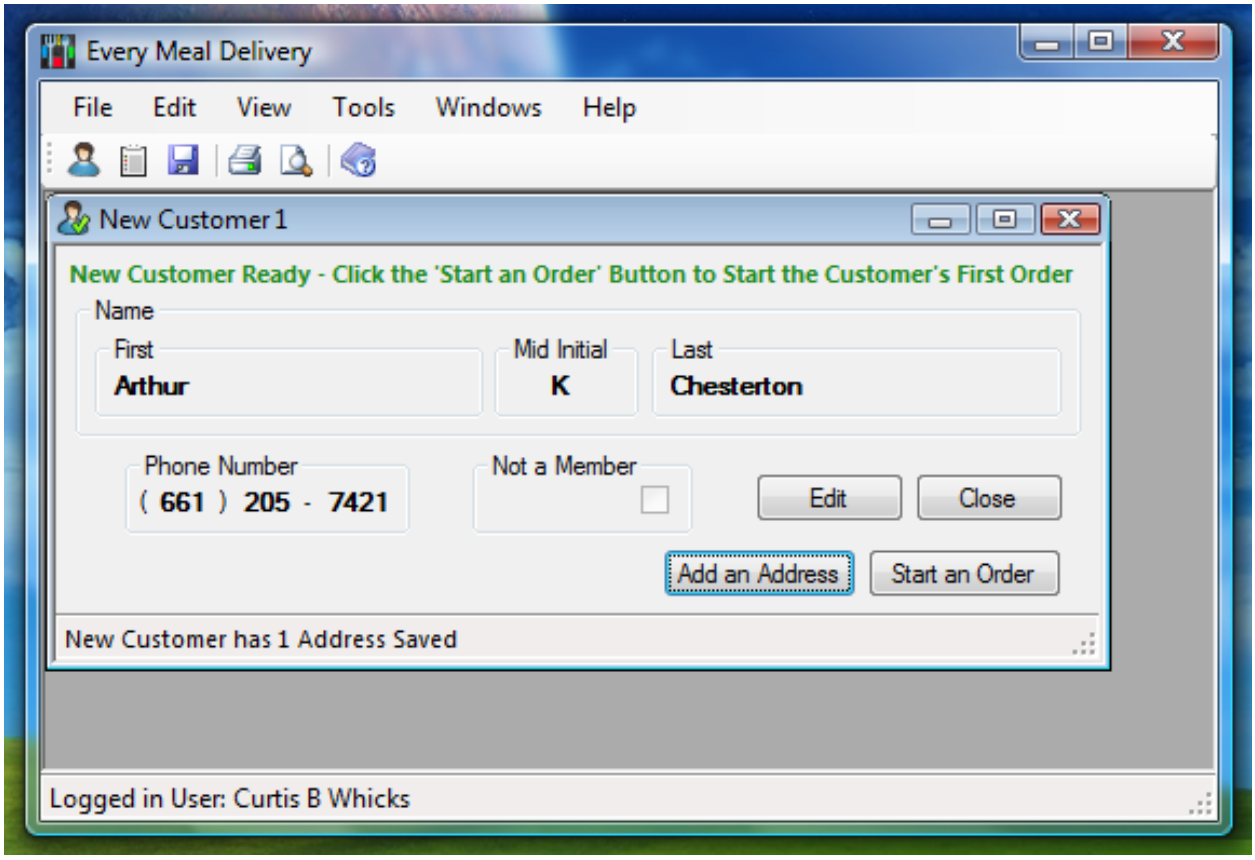
After keying in a full correct address entry (aside from the optional Apt text field), and clicking the Apply button, the address entry form locks in, and notes a successful address in the form.



Address Entry Success Form

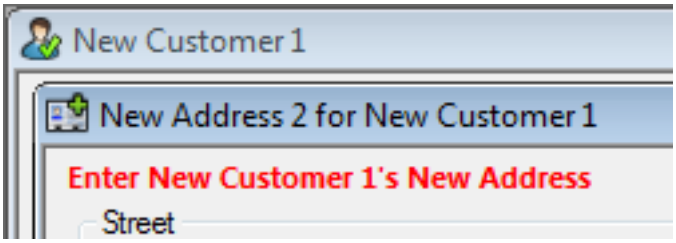
Like in the main new customer entry form, the Apply button in the address entry form turns into an Edit button, letting the dispatcher know it's still possible to edit the current address.

Upon the dispatcher deciding the new address information is correct, and clicking the Close button, the new address is queued in the new user's address list for uploading to the database. The main new user window notes the addition of a new address, and lets the dispatcher know a new order for the new user can be started.



New Customer Setup Ready

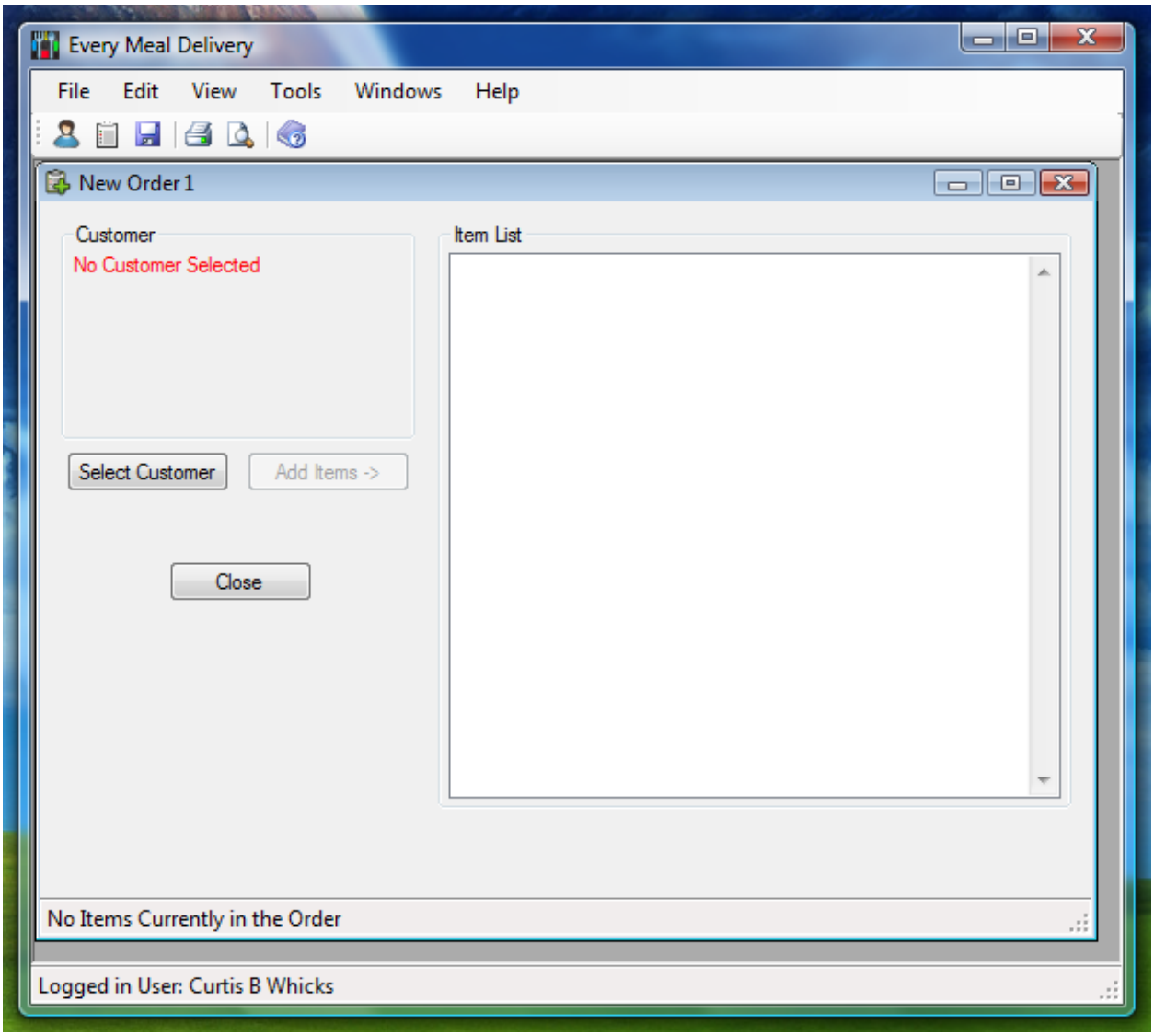
At this point the dispatcher can modify the user's basic information, add another address to the user's account, or start the order for the new customer. Either way, closing from here on saves the new information to the database. Should the customer have another address to add, the program will number in the next address for the new customer.



Second Address Form Title

3.3 The Order Setup Forms

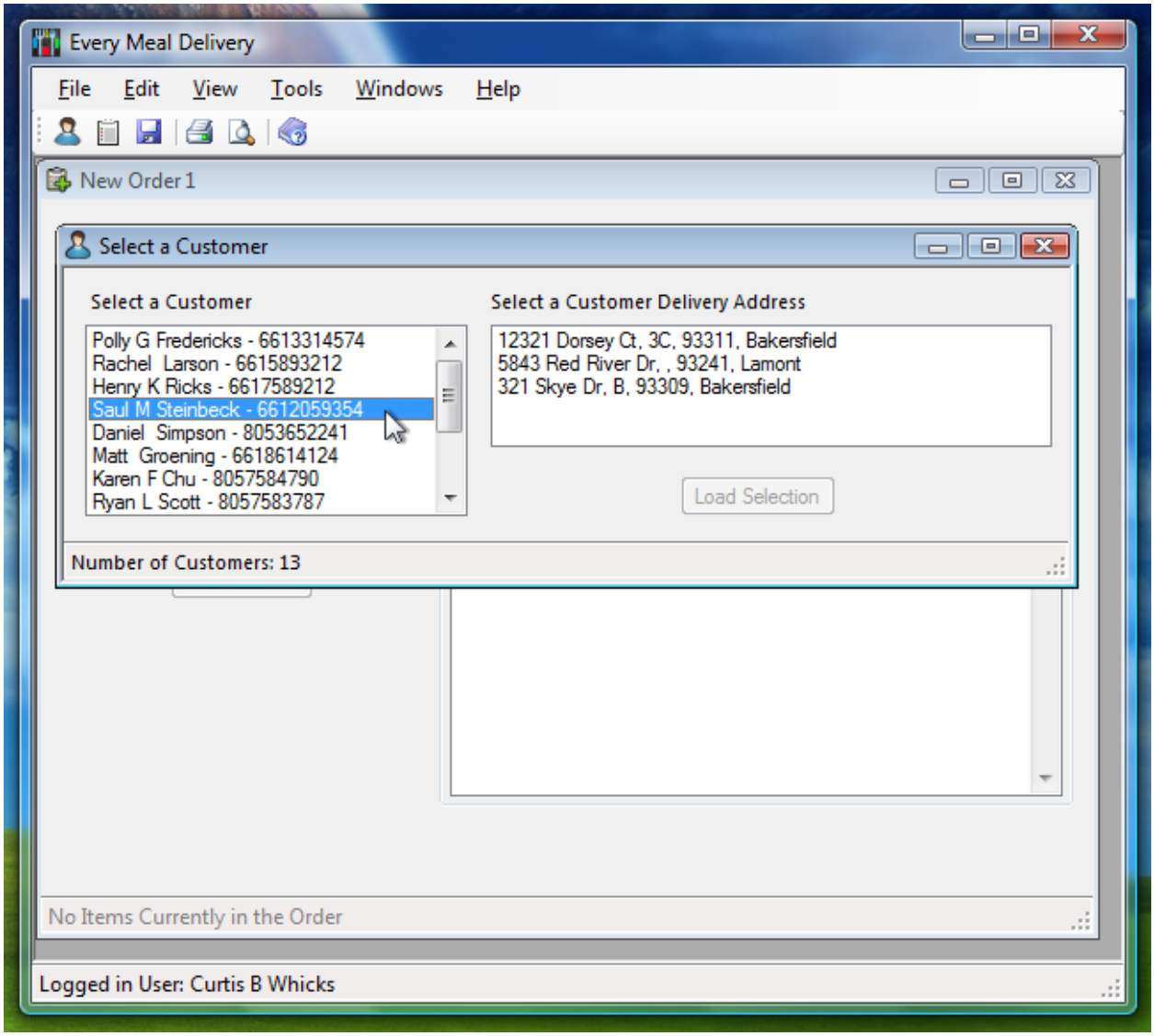
The other major component of the dispatcher role is setting up customer orders. To start setting up an order, a dispatcher can bring up the new order form by clicking the new order icon in the toolbar. Doing so brings up the following form:



Blank Order Form

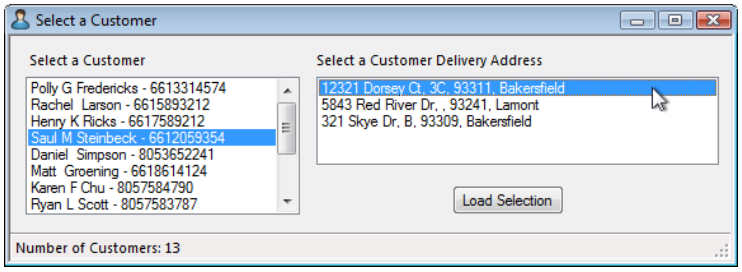
The first thing one can see in this form is not only that it's empty, but that the Add Items -> button is disabled by default. This is done so that the dispatcher is made to select a customer for the order first. Clicking the Select Customer button brings up a child form from the order form, making the overall order form disabled. The child form is a customer select form with two list boxes, one for the customer names, and phone numbers, and the other on the right to list every address of the customer selected on the right. For example, the next screen shows that selecting customer Saul M Steinbeck with

phone number 6612059354 displays three addresses belonging to the customer in the right list box.



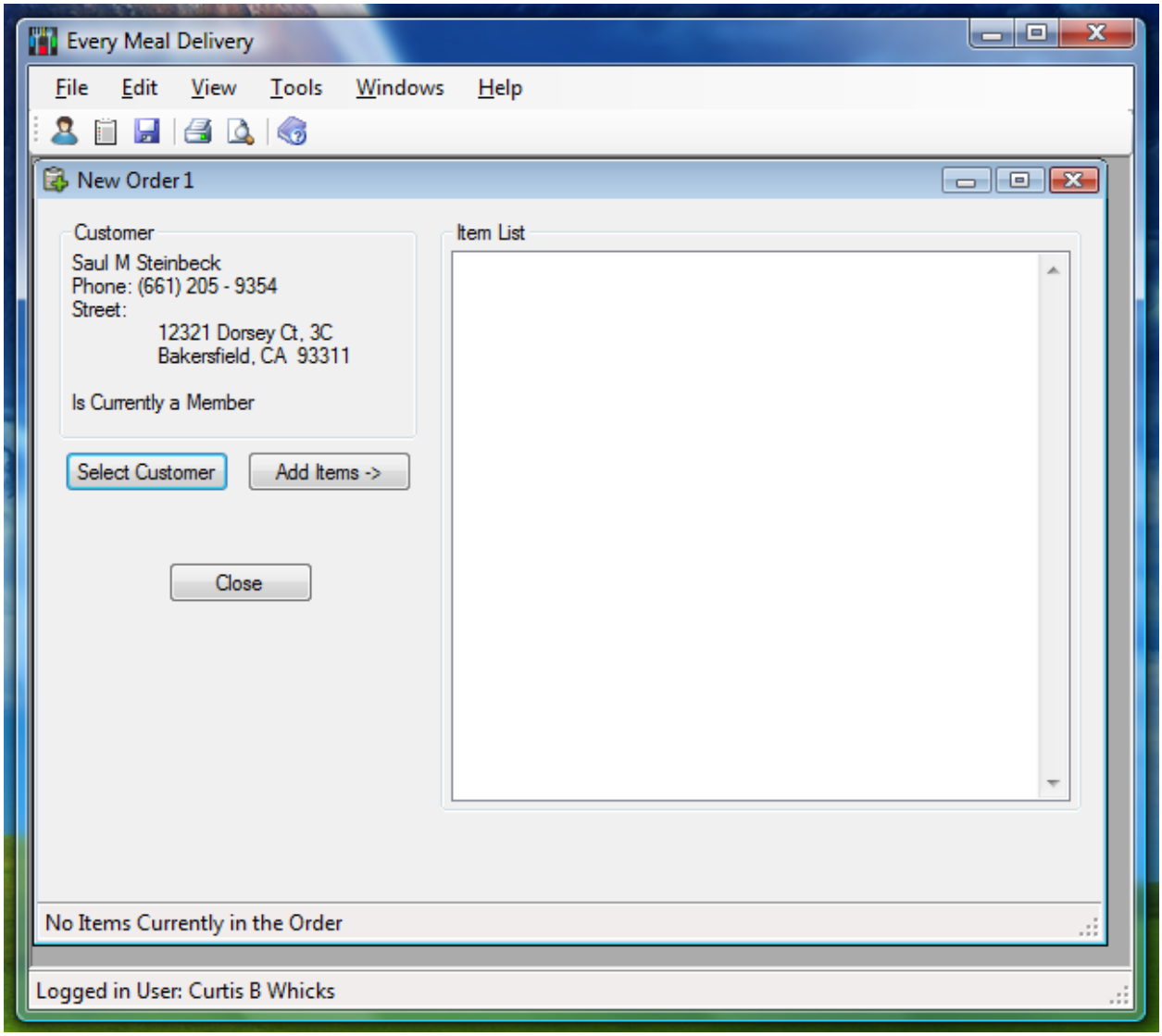
Selecting a Customer for the Order

The application's order of operations is again enforced here in that the dispatcher isn't allowed to load the customer into the order until a customer address is selected from the list box to the right. The Load Selection button becomes available the moment the dispatcher selects an address.



Load Selection Button Enabled after Address Selection

After selecting, and loading a customer, the Select a Customer form closes, thus enabling the main order form with the Customer area filled with the customer delivery information.

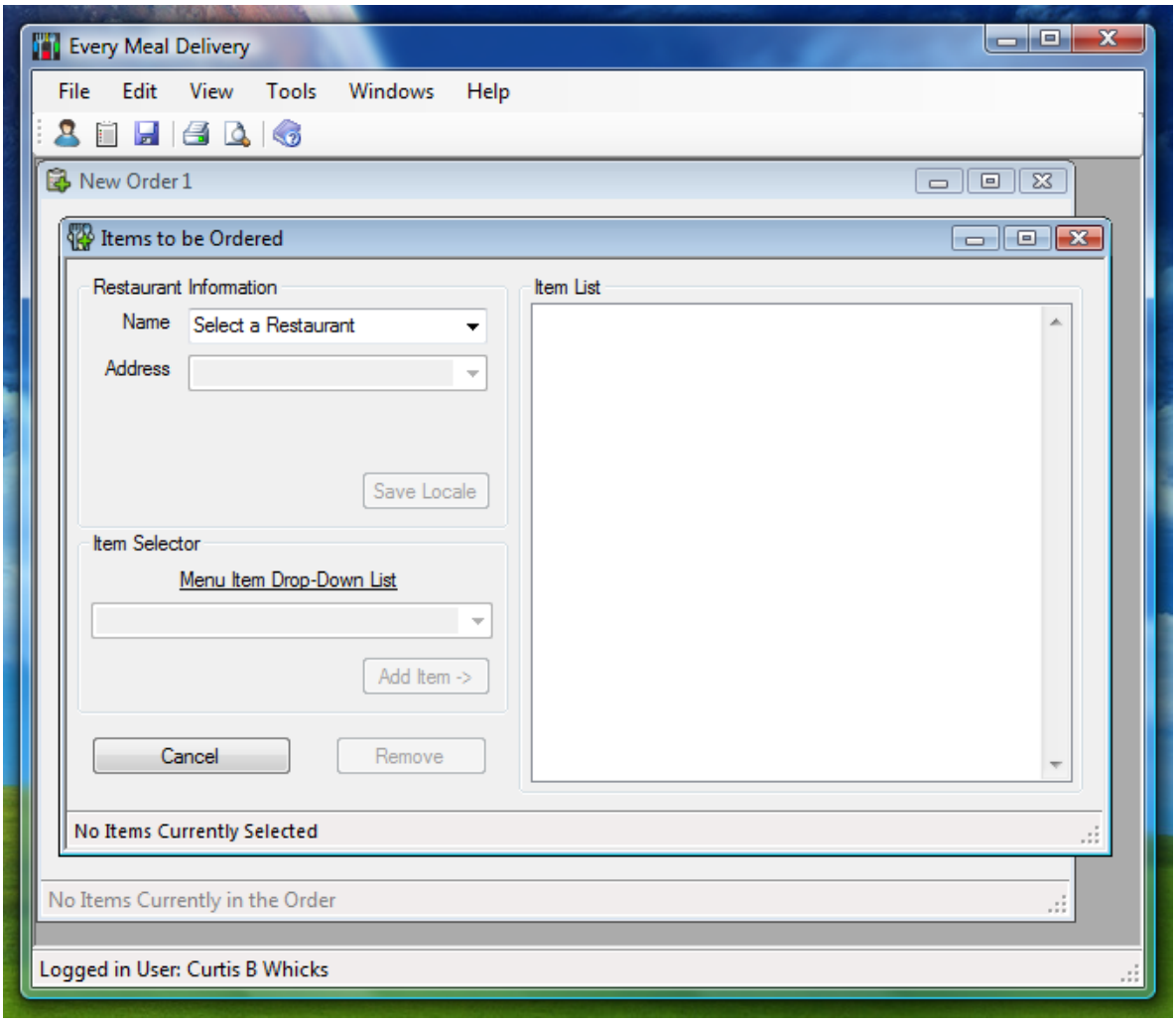


Order Form with Customer Information

As soon as customer information has been filled in, the business order of operations then allows the dispatcher to add items to the order by enabling the Add Items -> button.

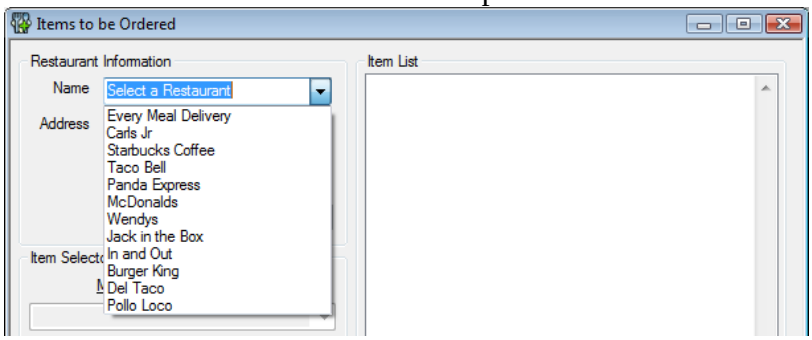
Clicking the Add Items -> button disables the new order form, and opens the Items to be Ordered child form. Through this form the dispatcher selects a single restaurant, and specific franchise location of the restaurant. All the menu items for the specific franchise are available for choosing from within this form, and any number of such items can be added to the order. There is a business restriction, however, that holds that only one franchise of one restaurant can be included in each order. For example, a dispatcher cannot add items from two different Taco Bell locations to the order. If the order is to

include Taco Bell items, all such Taco Bell items will be ordered from the same franchise.



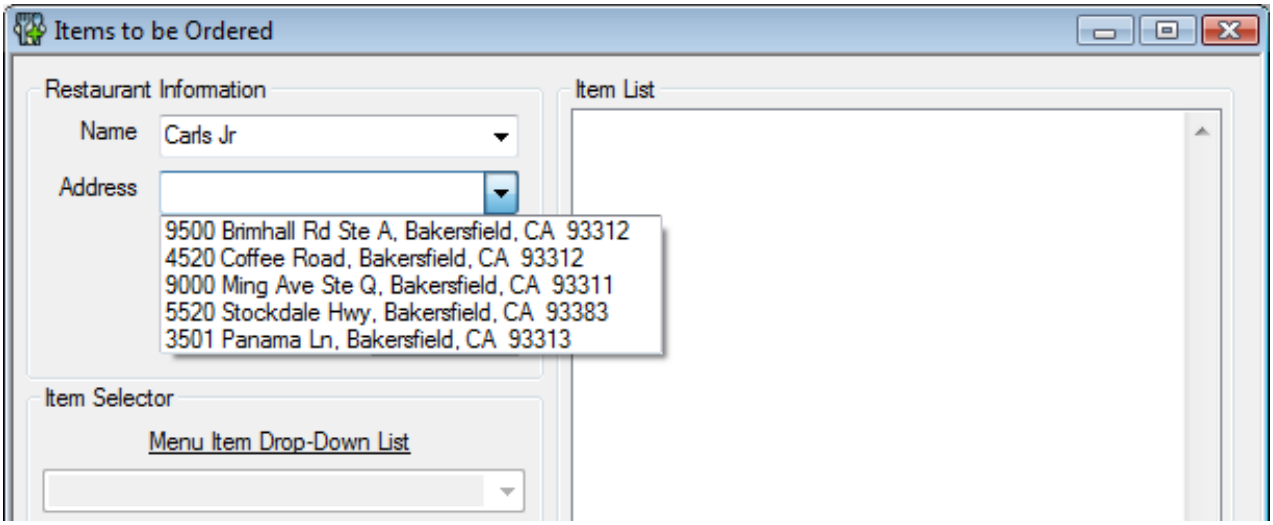
Items to be Ordered Form

Having the Items to be Ordered form up, the dispatcher first selects a restaurant name from the drop down combo box. The combo boxes in this form are set to enable search suggestions. Should the dispatcher decide to type in a selection, the combo box will attempt to append to the typed entry the rest of what is being selected. The dispatcher may only choose from what is listed in the drop down box.



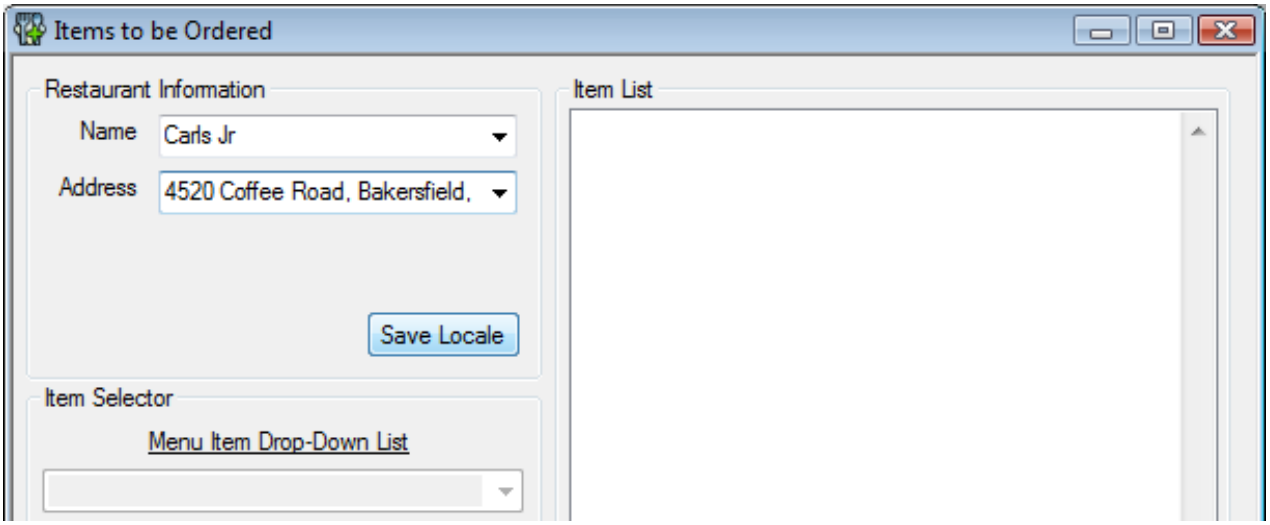
Restaurant Name Drop Down Combo Box

Upon selecting the restaurant name, the Address combo box then becomes enabled, and filled with every franchise address available for the chosen restaurant. By this time the dispatcher knows the customer's delivery location, so he/she knows which franchise address to choose closest to the customer delivery location. The application screen shows the available franchise addresses for the Carls Jr restaurant name.



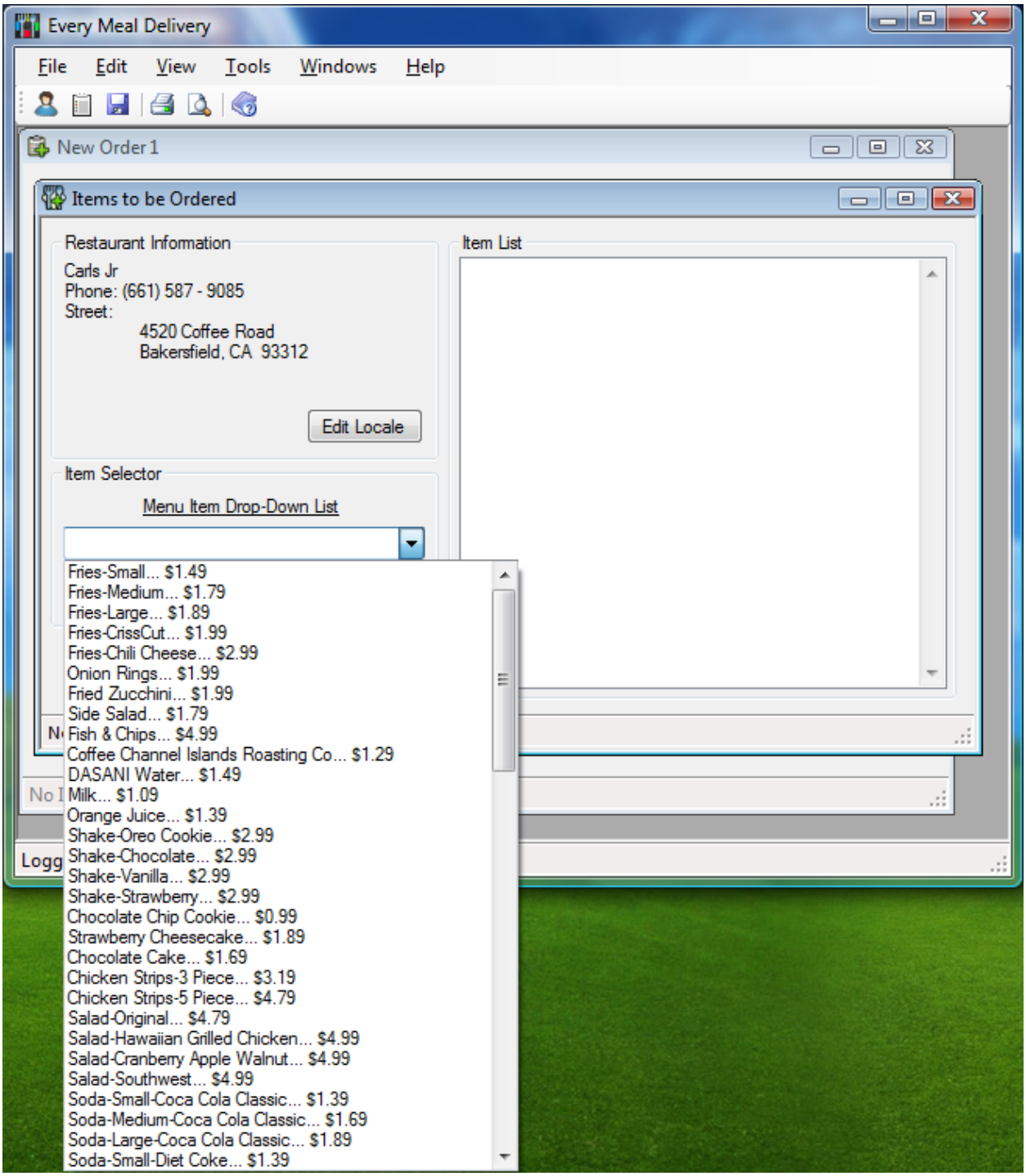
Franchise Address Drop Down Combo Box

After the dispatcher selects the franchise address the form then enables the Save Locale button.



The Save Locale Button is Enabled

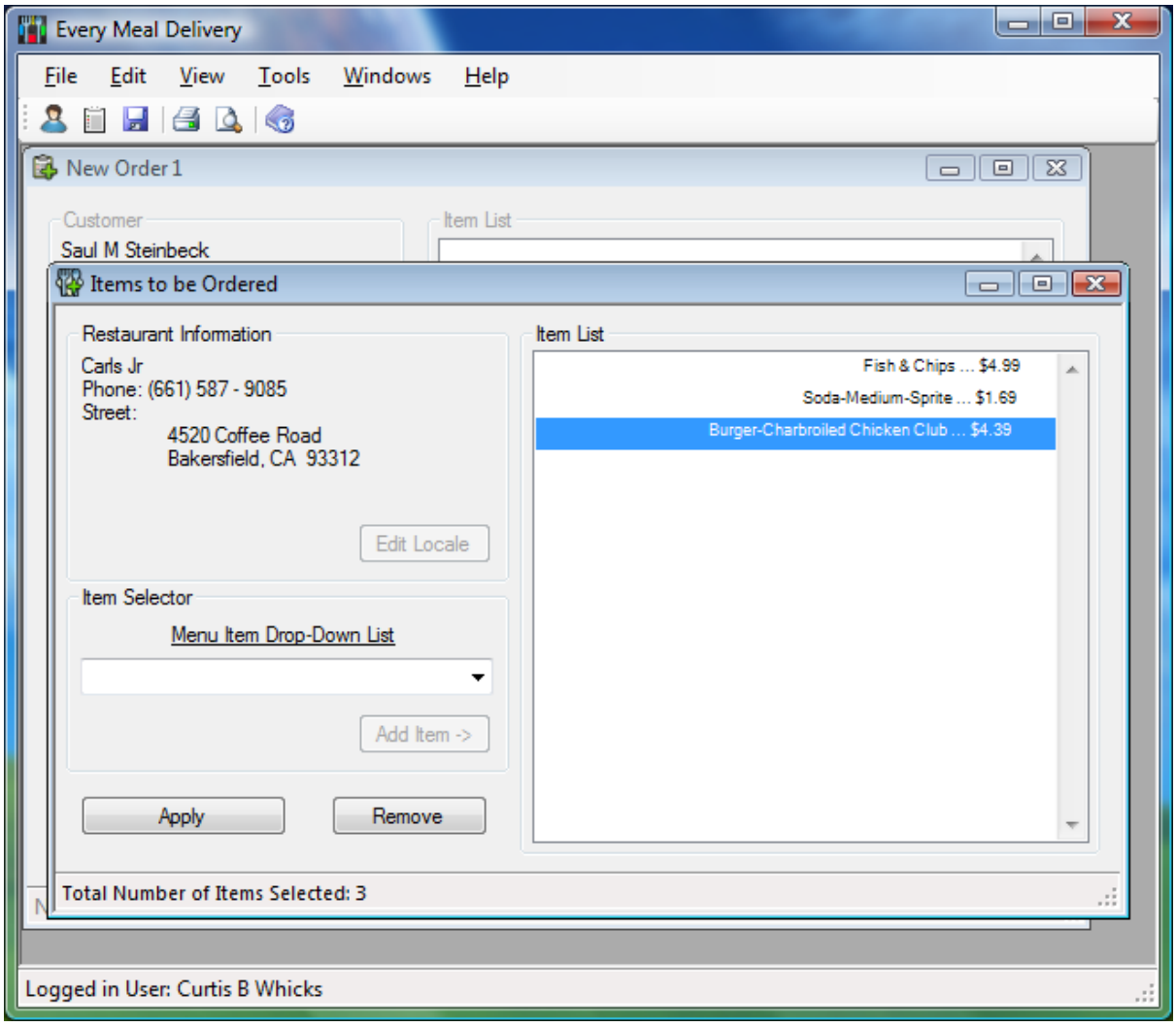
To lock in the restaurant, and franchise, and bring up the franchise's item menu the dispatcher clicks the Save Locale button. After which, the restaurant information display gets formatted to an easy to view layout, the Save Locale button switches to an Edit Locale button, and the franchise's menu items become selectable in a drop down combo box in the Item Selector group box located below the Restaurant Information group box.



A Franchise Expanded Item List

After selecting an item from the franchise's item list the dispatcher can then click the Add Item -> button to add the item to the list of items to be bought from the current restaurant franchise for the order. Adding an item not only adds the selected item to the list, it also enables the Remove button allowing the dispatcher to remove items from the order list, and the Cancel button becomes an Apply button which when clicked adds every item chosen to the general order list. The Add Item -> button also becomes disabled

when a non-legitimate franchise menu item is typed into the menu item drop-down list. The Add Item -> button only becomes enabled when an actual item from the franchise menu is selected, or typed. As soon as an item is added to the list, the Edit Locale button becomes disabled. It only becomes enabled at this stage when the selected item list is empty. The status bar also updates to display the number of items entered for the franchise order. The following screen shows the Items to be Ordered form after three items have been added to the franchise order.

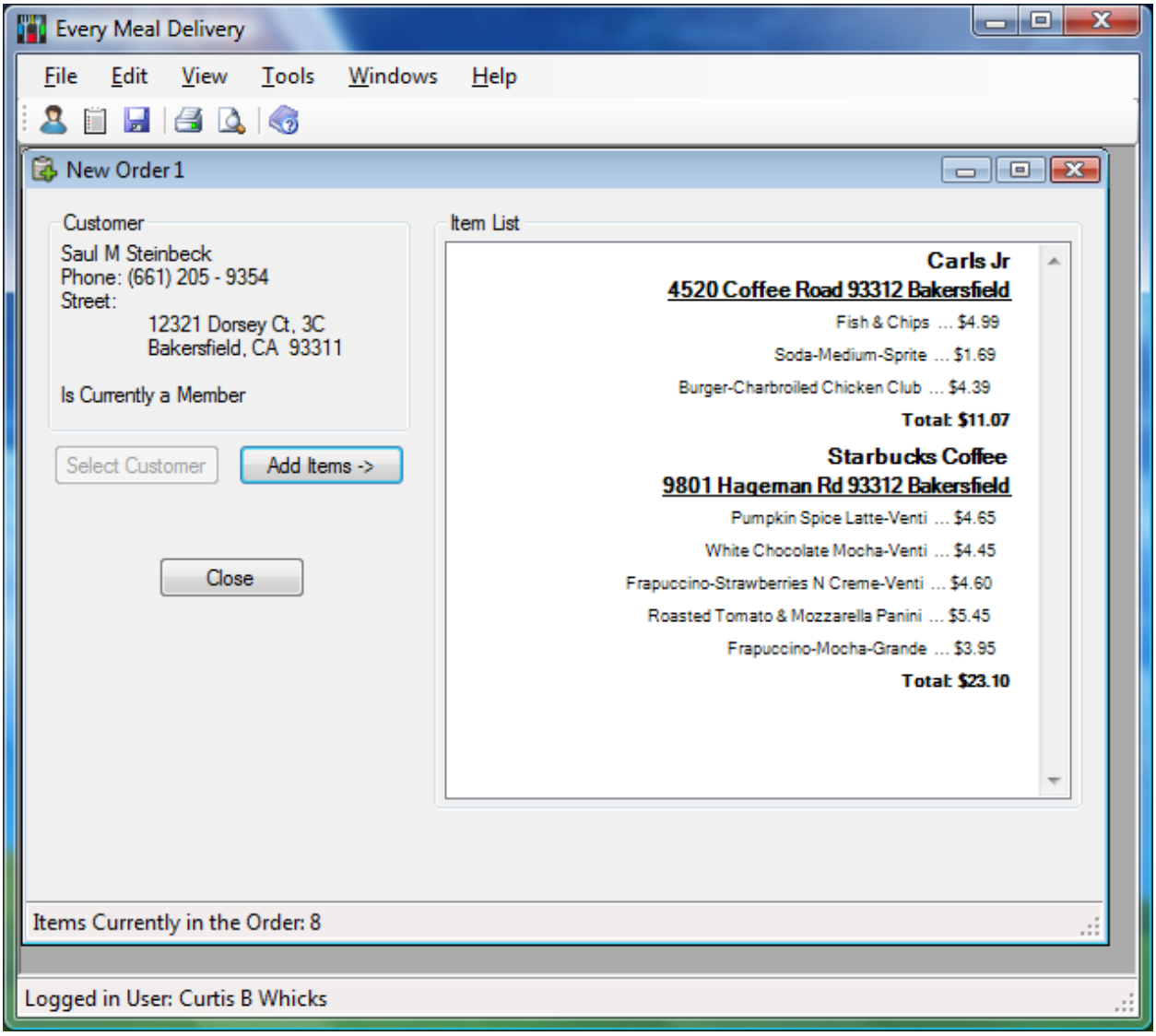


Three Items Entered in the List

It's possible to delete the blue selected item by pressing the Remove button.

At this point let's say the dispatcher clicks Apply, because the customer has decided the selected items from the specified franchise are all he wants. The customer then decides to move on to order items from a different franchise. When the dispatcher clicks Apply the Items to be Ordered form closes while re-enabling the parent New Order 1 form. Every item in the franchise list gets carried over, and the item costs get tallied in the main order form. The main form also groups the items under the name, and address location of the restaurant franchise from which the items are to be purchased. The dispatcher can

also go back, and add items from a different franchise of the customer's choosing to add to the order. The process for adding another restaurant's items to the order is the same as the process of adding the first restaurant's items to the order. The following screen picture shows the result on the main order form of the dispatcher adding various items from different restaurants.



Total Set of Order Items from Two Different Restaurants

Part 4: Describing the Code

Though the front end of a database program may look simple in its layout, getting everything put together to function in a way that enforces program data entry rules, and order of program operation requires quite a bit of programming. The software design, and analysis process requires one to do all the investigating necessary not only to understand how a program works, but how it can be further improved. For now, however, the main concern of this section is understanding the code design structures necessary for this project.

4.1 Major Step in Designing a User Interface

Initially, I had in mind to use a single form interface through which all forms in the application would swap. Instead of having a multiple document interface workspace, I figured a simple form would be enough. The application would start with a basic form with a simple set of menu options, all the same ones included in the current design. If a user were to click the new customer tool bar icon the entire application form would be replaced with the new customer entry form, and everything described in the previous section would carry on similarly. My main problem with this design, however, was that it didn't allow a flexible environment through which a user could do multiple things at once. If, for example, a customer were to want to make three separate orders as demonstrated in screenshots of the previous section, the dispatcher would have to complete one order, then move on to the next, then the next. Going back to recover the three separate orders seemed to me to be a bit too cumbersome. Either the user interface would require a super summary form allowing a dispatcher to merge together orders already entered into the system into one report, or the dispatcher would have to look up each order made individually to be verified back to the customer. For such a case I would have had to make up a pause status on each order so it wouldn't be entered into the system until the customer, and dispatcher agreed on the item totals. In the end, I figured the multiple document interface best, because multiple orders could be started at once, and easily reviewed to the customer for confirmation at once.

Initially, I'd had the idea for a single form layout, because the prior application I made for record loading was of this same design. Single form layout worked fine for such a simple data loading application, but for more complex operations such as the ones required in this project, it just wasn't robust enough.

4.2 Major Class Descriptions

The majority of this program so far involves use of classes used to hold up the forms used in the GUI part of the application. Though these are interesting in their own, and they will be summarily described, their focus doesn't touch so much on the main premise of this project, which is databases. For that there is one main class dealing with managing data through the Oracle 10g DMBS used in the project. Though most of the classes will

have their brief expository summations in this section, I think it best to expound a bit further on the Oracle related database class this project contains. Something important to note is that anytime a database manager object is brought up in any of the following major form classes, the object being referred to is an instantiation of the same main Oracle related database class already mentioned.

4.2.1 Major Form Classes

Class: fMain

This is the main form class for the multiple document interface. It contains an array of form objects it uses to keep track of child forms it spawns within the workspace. Since it is the principal form of the program, it contains all the event methods necessary to react to toolbar icon clicks, and main menu selections. It also contains functions used to restrict text box data entry to specified formats.

Class: cfCustomer

cfCustomer is a form class containing the new customer form called from the main form. Most of this form class involves event based methods tailored to change the form's structure should data entry events happen as described in the previous part of this phase. This class also contains an array of address list forms it uses to keep track of the number of new addresses pertaining to new customers. A database manager object is also included in this class for when new customer information is pushed to the database.

Class: cfNewAddress

Form class cfNewAddress is the class for the new address form used to setup a new customer's new address information. Much like the new customer form which instantiates new objects of this form class, cfNewAddress also contains many event methods used primarily for the sake of enacting form manipulation processes described in the previous part of this phase. When told to do so by the parent new customer form, objects of this class use their local database manager objects to push new address information to the database with the help of customer information passed along from the parent new customer form class.

Class: cfOrder

This class is the base order form class. Not only does this form class contain many event methods used to enforce the required order of application data entry, it also contains custom application container objects used to hold requested order data to be pushed to the database later in the application process. Of course, methods included in this class instantiate form objects which add customer, and item information to this form. All the standard order information found on the

order form is information held in the container objects until the order is confirmed ready to be sent to the database.

Class: cfExstingCstmr

cfExstingCstmr is the class form from which an object is instantiated when the customer information needs to be added to the main order form. This is the class for the select a customer form. This class has necessary form list box event methods used in conjunction with a database manager object to pull information from the database which is displayed in this form class objects' list boxes. Event methods already described for this form are also part of this class.

Class: cfAddItems

cfAddItems is the form class used to instantiate restaurant item listing forms when requested from the main order form. Aside from the personally customized set of form event methods dealing with enabling/disabling buttons, combo boxes, locking in restaurant information, etc., this class also uses a database manager object to pull specified information from the database. Information pulled from the database is stored in custom class containers for the purposes both of display in combo boxes, as well as sending back to the order form which instantiates objects of this class.

4.2.2 Important Basic Application Data Utility Classes

Class: emsForm

emsForm is a form management utility class used to tell the main form of the program the type of new form it's to make. An object of this type is instantiated prior to the initiation of any of the form types described above. This class is used mainly for coding convenience to make the base parent's form making function more manageable. This class contains objects of every form type mentioned above, of which only one type is instantiated every time an object of this type is instantiated.

Classes: containerCustomer, containerAddress, containerArrayAddress, containerRestaurant, containerFranchise, containerItems

This set of classes is essentially a group of advanced structure-like types, each used to retain Database information either after it's pulled, or before it's pushed. Data members in any of these classes are simple types matching the information required for storage, except for the containerArrayAddress class which holds an array of simple data-types for the sake of holding record sets of information to be

sent to the database all at once. Objects of these kinds of classes are used throughout the application by the various forms that need them.

4.2.3 The Primary Database Management Class

Class: OManager

In order to understand how data is transmitted/received between the front end application, and the DBMS it is necessary to expound in detail on the contents of some of class OManager's members. The purpose of this class is to setup transmission lines between the front end application, and the DBMS. Some detail on the basics of how this is done is explained in the following few sections.

Setting Up a Data Connection

As has already been made clear, the DBMS to which this application connects is Oracle 10g, and as such there are certain Oracle related syntax structures required to have a .NET C# application connect properly for the purposes of data transmission.

Prior to do doing anything else, an object of the OManager class runs a few basic necessary routines during instantiation through its constructor. These are listed in the following C# code with appropriate comments for explanation.

```
//The database IP address as well as Login, and Password is specified in a
//    full independent connection string structure.
   oradb = "Data Source=(DESCRIPTION=" + "(ADDRESS_LIST=(ADDRESS=(PROTOCOL=TCP)"+
           "(HOST=helios.cs.csubak.edu)(PORT=1521)))(CONNECT_DATA=(SERVER=DEDICATED)"+
           "(SERVICE_NAME=ORCL)));" + "User Id=cs342;Password=c3m4p2s;";

//An Oracle Connection Object is Constructed, and Assigned the Database Login Info
            conn = new OracleConnection();
            conn.ConnectionString = oradb;

//A new Oracle Command Object is Made, and Assigned the Oracle Database Connection
//    also recently just Made
            cmd = new OracleCommand();
            cmd.Connection = conn;
```

Opening and Closing the Data Connection for Command Execution

After making an Oracle connection object, anytime a call to the database is to be made, the connection object must be opened before the Oracle command can be executed. After the command is executed, the connection is then closed demonstrated in the following simple C# code:

```
conn.Open();
<Oracle execute command goes here>
conn.Close();
```

Using an Oracle Stored Procedure to get a Record Set to C# .NET

The following Oracle stored procedure named HV_SP_GETCUSTOMERS is used in the OManager class CustomerList method member.

```
-- Inside Oracle
-- This Simple Procedure Returns a Record Set through a SYS_REFCURSOR

PROCEDURE "HV_SP_GETCUSTOMERS" ("RCLIST" OUT SYS_REFCURSOR)
AUTHID CURRENT_USER IS

BEGIN -- executable part starts here

      OPEN RCLIST FOR SELECT * FROM HV_CUSTOMER;

END;
```

/*Inside Visual Studio*/

```
//The Following Method Stores the Return Record Set into a containerCustomer object
//                                                                          Array
//The custom containerCustomer object class has the following definition:
//
//   public class containerCustomer
//  {
//      public int    _ID;
//      public string _fName;
//      public string _mInitial;
//      public string _lName;
//      public long   _Phone;
//      public int    _status;
//      public containerCustomer () {}
//  }
//
//*********************************************************
// Beginning of OManager Class Method Member CustomerList()
//
public containerCustomer[] CustomerList() {
 //Holds the size of the Customer Data Container Array
   int size;

 //Instantiates a Zero Length Customer Data Container Array
   containerCustomer [] cHolder = new containerCustomer [0];

 //Tell the Oracle Command Object the Name of the Stored Procedure
   cmd.CommandText = "HV_SP_GETCUSTOMERS";

 //Tell the Oracle Command Object it's an Oracle Stored Procedure type of Command
   cmd.CommandType = CommandType.StoredProcedure;

 //Add the SYS_REFCURSOR OUT Parameter to the Oracle Command
   cmd.Parameters.Add("outValue", OracleDbType.RefCursor,
                                  ParameterDirection.Output);
 //Open the Oracle Connection
   conn.Open();

 //Make an Oracle Data Reader Object
    OracleDataReader reader = cmd.ExecuteReader();
```

```
 //While the reader still has return information, return the reader's return values
 //(Return values are stored as an array in the reader with the number of array
 //                  offsets equal to the number of attributes in each Oracle record)
 //The Customer container is resized +1 everytime it needs the space to store
 //                                                          another returned record.
   for (int i = 0; reader.Read(); i++ )
   {
       size = cHolder.Length;
       Array.Resize<containerCustomer>(ref cHolder, size + 1);
       cHolder[i] = new containerCustomer();
       cHolder[i]._ID       = Convert.ToInt32(reader.GetValue(0).ToString());
       cHolder[i]._fName    = reader.GetString(1);
       cHolder[i]._mInitial = reader.GetString(2);
       cHolder[i]._lName    = reader.GetString(3);
       cHolder[i]._Phone    = reader.GetInt64(4);
       cHolder[i]._status   = reader.GetInt32(5);
   }

//Close, and Dispose of the reader, clear the Oracle command's parameters, close the
//Oracle connection, and return the newly filled customer container array.
   reader.Close();
   reader.Dispose();
   cmd.Parameters.Clear();
   conn.Close();
   return cHolder;
}//End of OManager Class Method Member CustomerList()
```

Send a C# .NET array of records all at once to an Oracle Stored Procedure

I was able to implement the next function at the last minute so I know it works, even though it hasn't been used in the application. It's important to include it for the sake of having an efficient way to send a record set, eliminating the need to make repeated individual slow connection calls to the DBMS.

The following Oracle stored procedure named HV_SP_TBTEST is used in the OManager class apTester method member.

```
-- Inside Oracle
-- This Simple Procedure Insert a single Record with 3 Attributes

PROCEDURE "HV_SP_TBTEST" ("STRINGPASS" IN VARCHAR2,"INTPASS" IN NUMBER)
AUTHID CURRENT_USER IS

BEGIN -- executable part starts here
     insert into HV_TABLETEST (PKCOLUMN, STRINGER, NUMBSKY)
                          values(HV_SQ_TEST.NextVal, STRINGPASS, INTPASS);
END;
```

/*Inside Visual Studio*/

```
//The Following Method Sends a Simple Record Set (Represented by Two Arrays) to the
//                                                 Oracle HV_SP_TBTEST stored procedure
//The Method sends a record set once to a stored procedure where Oracle repeats the
//                    stored procedure multiple times to insert the entire record set.
//
//*********************************************************
// Beginning of OManager Class Method Member CustomerList()
//
public void apTester() {
   //Tell the Oracle Command the Name of the Oracle Stored Procedure
     cmd.CommandText = "HV_SP_TBTEST";

   //Tell the Oracle Command Object it's an Oracle Stored Procedure type of Command
     cmd.CommandType = CommandType.StoredProcedure;

   //Tell the Oracle Command the number of rows it will receive.
     cmd.ArrayBindCount = 3;

   //Two Arrays to Function as Simple Sample Record Sets
String [] Street = new String[3]{"234 Pal", "234 Jim Bean", "345 Plane Dr" };
    int[] Zip    = new int   [3]{ 93280,     93312,          90210 };

   //Tell the Oracle command the type of Parameters the Stored Procedure
   //                                                              will receive
cmd.Parameters.Add("Street_Column", OracleDbType.Varchar2, Street,
                                                  ParameterDirection.Input);
cmd.Parameters.Add("Zip_Column",    OracleDbType.Int32,    Zip,
                                                  ParameterDirection.Input);
   //Open the Connection, Execute the Procedure, Close the Connection, and
   //                                  clear the Parameters from the command.
     conn.Open()
     cmd.ExecuteNonQuery();
     conn.Close();
     cmd.Parameters.Clear();
}//End of OManager Class Method Member apTester()
```

The OManager class contains essential methods like the ones just listed so the other form classes can send or receive database information in an efficient enough manner.

4.3 Major Features of the GUI

The strong suite of the graphical user interface for the Every Meal Delivery system lies in the fact that it uses icons whenever possible with clear labels on what everything does, through a user-friendly multiple document interface that enforces business application rules of operation whenever possible. This combined suite of features offers the user a reliable easy to understand, easy to use application with a low tolerance for errors. The low tolerance for errors may not be something the typical user would care about, but the managers in charge certainly would.

Easy to understand symbols of operation in the application are very important to how the typical user goes about doing his/her job within the application's workspace environment. From the moment the user brings up the new customer entry form, the current task at hand is made extremely clear. The form itself tells the user what to do at every moment: "Enter the New Customer's Basic Information", "Click 'Add an Address' Button to Add a Customer Address", "New Customer Ready", etc. Even the icons in the corner of each form inform the user to the category of the form itself. When it comes to training new people into how to use a Business Management System, all such symbolically supported educational queues are very important.

4.4 Learning New Tools

When I first came into development in this project my experience with developing project forms in C#, and Oracle was little to none. My background experience came mostly in C++, with some web programming here, and there. What helped me to develop this C# application was reading straight to what I needed to accomplish. I knew C# wasn't much different from C++ so I felt quite assured to do okay in that respect, and learning the Oracle portions of the program came to me much the same way learning new things in C# did. I consistently researched to learn what I needed using every useful resource I had at my disposal. If I didn't learn it in class, I learned it through a Visual C# 2008 book I bought by Deitel, and Deitel, and the internet. Mostly it was the internet. If ever there was something I needed to do in my project, my main recourse consistently came back to researching the topic at hand through a Google search leading me to the Microsoft Developer Network, The Code Project Development resource, Oracle forums, Oracle help files, etc. If the information I required even the web didn't have, sometimes I'd have no choice but to trial, and error the functionality of objects provided in the Visual Studio class libraries. What helped throughout the entire ordeal was staying on point researching, trying whatever I could, no matter how long it took doing whatever I could to get things done. Primarily, what laid at the root of my learning in this project was making a constant effort to sustain a tenacity to focus on what I needed to meet the

goals I set to accomplish. It all starts and ends with wanting to do something, and for me it's been about doing whatever I can to learn from any useful resource I can find to do what I need to see this project through.

What also helped is the Visual Studio development environment. At this point the IntelliSense features the Visual Studio IDE offers are extremely helpful when trying ensure syntax in a program is structured correctly. Its suggestive nature consistently teaches, and reminds one of how to build a working program.

Part 5: Application Design and Implementation

My main intention with the database application I was designing was to have a well-known layout style with sufficiently good use of graphic design setup in way easy to understand by anyone who uses the application. I took ideas from user interfaces provided by common popular applications such as the Firefox web browser. However, I quickly came to realize the application interfaces for such products are beyond what is easily offered in the Visual Studio environment. One thing easy to learn in application programming is that getting dynamic good looking graphics in a graphical user interface takes a lot of planning, and time to implement.

American computer scientist professor Ben Shneiderman who specializes in the field of human-computer interaction recommends the following guidelines for a good form/report design:

- Meaningful title – the title of a form should unambiguously identify its purpose
- Comprehensible instructions – terminology should be used to instruct the user with a standard grammatical style
- Logical grouping and sequencing of fields – fields that are related in a form should be placed near one-another.
- Visually appealing layout – fields, and groups should be evenly positioned through out the form.
- Familiar field labels – fields should be easy to distinguish
- Consistent terminology and abbreviations – as the description implies, familiar terms, and abbreviations should be used consistently in the form
- Consistent use of color – color should be used to highlight important areas of the form

- Visible space, and boundaries for data-entry fields – it should be easy for a user to know how much space is available in the form

- Convenient cursor movement – Tab key, arrows, or mouse pointers should be used to easily spot required operations

- Error correction for individual characters and entire fields – common items such as Backspace key, or insert should be established as available input mechanisms for righting data entry mistakes in a form

- Error messages for unacceptable values – entering incorrect data into a form should inform the user of incorrect entries being made

- Optional fields marked clearly – fields the user doesn't have to use should be made clear

- Explanatory messages for fields – tooltip or windows status information should inform the user of the purpose of a field

- Completion signal – The application should make evident when the field entry process is complete

The good thing about the application I've set up so far is that most of the Dr. Shneiderman's recommendations are taken into account. Any further form setups made on the project regarding other components are definitely to continue toward more the same type of design ethos. Currently, certain recommendation options such as the clear marking of optional fields are something that can perhaps be added to forms in areas such as the entry of a middle initial, apartment information, or the optional member status checkbox in the new customer form.

Conclusion

The idea for this project came to me a while ago when working out near Taft doing payroll for a contracting company. At the time I thought it'd be a great if there were a fast food delivery business delivering people's favorite fast food items to remotely located businesses, or even just to busy people who could benefit from having such an affordable convenience. Recently, with the research I made to complete this project I came across people from big cities throughout the United States online who practice a business model similar to that proposed by this application. This has led me to ponder that perhaps any great idea can be improved on, if not at the very least modeled with some form of useful technology, in a way that could make life easier, and just plain more productive. For anyone truly wanting to build a more efficient system of organization, even in a field as rare as general fast food delivery, a database can definitely help things go far in ways greatly organized, and perhaps even more resilient.