Conceptual Database Design

Rusty's Pizza Database

Christopher Jestice

10/19/2010

Table of Contents

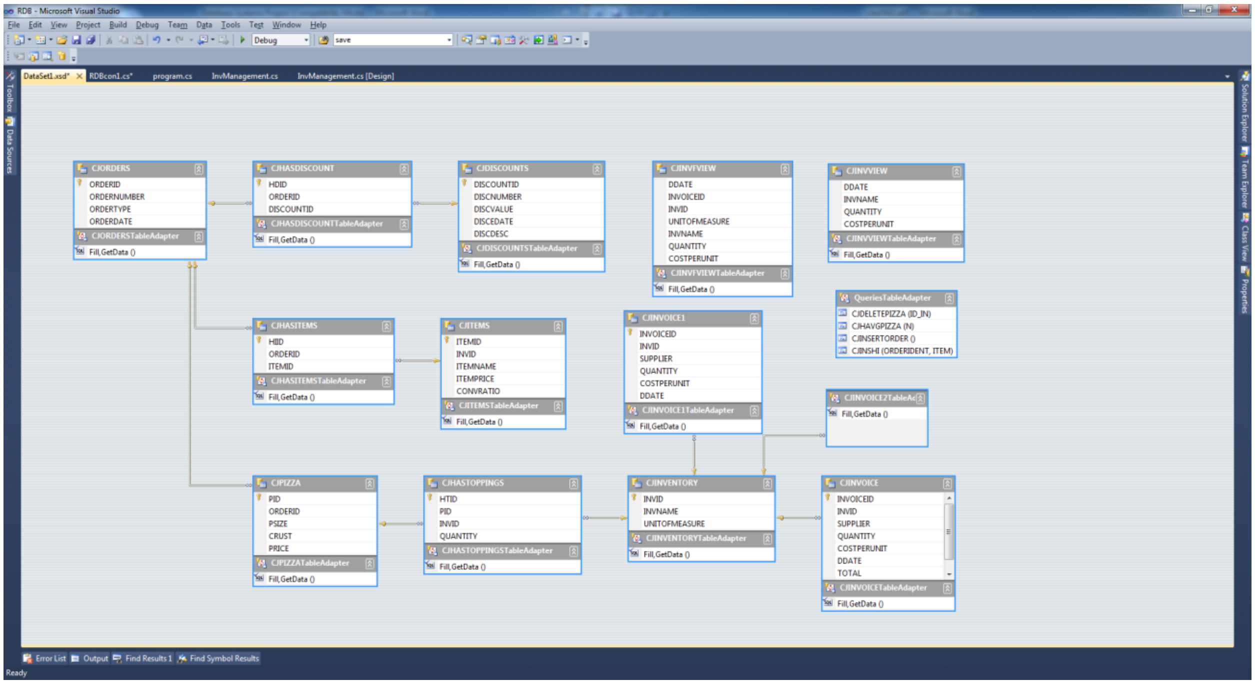
Phase I: Conceptual Database Model

Phase II: Relational Model

Phase III: Oracle Implementation

Phase IV: Stored Procedures

Phase V: GUI Design and Implementation


RDB - Microsoft Visual Studio
DataSet1.xsd*
RDBcon1.cs*
program.cs
InvManagement.cs
InvManagement.cs [Design]
CJORDERS
ORDERID
ORDERNUMBER
ORDERTYPE
ORDERDATE
CJORDERSTableAdapter
Fill,GetData ()
CJHASDISCOUNT
HDID
ORDERID
DISCOUNTID
CJHASDISCOUNTTableAdapter
Fill,GetData ()
CJDISCOUNTS
DISCOUNTID
DISCNUMBER
DISCVALUE
DISCEDATE
DISCDESC
CJDISCOUNTSTableAdapter
Fill,GetData ()
CJINVFVIEW
DDATE
INVOICEID
INVID
UNITOFMEASURE
INVNAME
QUANTITY
COSTPERUNIT
CJINVFVIEWTableAdapter
Fill,GetData ()
CJINVVIEW
DDATE
INVNAME
QUANTITY
COSTPERUNIT
CJINVVIEWTableAdapter
Fill,GetData ()
QueriesTableAdapter
CJDELETEPIZZA (ID_IN)
CJHAVGPIZZA (N)
CJINSERTORDER ()
CJINSHI (ORDERIDENT, ITEM)
CJHASITEMS
HIID
ORDERID
ITEMID
CJHASITEMSTableAdapter
Fill,GetData ()
CJITEMS
ITEMID
INVID
ITEMNAME
ITEMPRICE
CONVRATIO
CJITEMSTableAdapter
Fill,GetData ()
CJINVOICE1
INVOICEID
INVID
SUPPLIER
QUANTITY
COSTPERUNIT
DDATE
CJINVOICE1TableAdapter
Fill,GetData ()
CJINVOICE2TableAdapter
Fill,GetData ()
CJPIZZA
PID
ORDERID
PSIZE
CRUST
PRICE
CJPIZZATableAdapter
Fill,GetData ()
CJHASTOPPINGS
HTID
PID
INVID
QUANTITY
CJHASTOPPINGSTableAdapter
Fill,GetData ()
CJINVENTORY
INVID
INVNAME
UNITOFMEASURE
CJINVENTORYTableAdapter
Fill,GetData ()
CJINVOICE
INVOICEID
INVID
SUPPLIER
QUANTITY
COSTPERUNIT
DDATE
TOTAL
CJINVOICETableAdapter
Fill,GetData ()
Error List
Output
Find Results 1
Find Symbol Results
Ready

Phase I: Conceptual Database Model

Fact-Finding Techniques and Information Gathering

Description Fact-Fining Techniques

Fact finding is a technique adapted to reduce rework as the development process progresses. Through this process the developer becomes well informed regarding the activities and structure of a specific company or organization. The result is a high level of understanding that prevents mistakes that would be made otherwise.

The method used for fact finding in this instance is full submersion. I have been employed by Rusty's Pizza for nearly five years and because of my position as the Assistant Manager, I have become acutely aware of the inner workings of the company and which items are kept track of, and those that are not. This knowledge has been acquired by learning every role within the store, including that of the Manager, who keeps track of the inventory, labor, and sales that the store has. The result of my training and experience is a very high level of understanding with regards to the structure of Rusty's Pizza.

Introduction to Enterprise/Organization

Rusty's Pizza is a company that purchases raw ingredients and sells it in the form of a finished product. The store operates on a first in first out basis and maintains a limited amount of inventory so as to provide the best product possible.

Rusty's Pizza Bakersfield is split into two different companies comprised of the east and the west side businesses. Both are joined by a joint operation in which all of the phone orders are placed from a single location and then routed to the appropriate store.

Each store has orders that are sent from the phone center and those that are from walk-in customers. The orders from the phone center are either pick-up orders, where the customer picks the order up from the store and pays for it at the register, or delivery orders, where payment is handled at the customer's house and does not require the use of a register. Walk-in customers place their orders at the register and can choose from all products that are available.

For every order that is placed, the primary ingredients used (mushrooms, onions, cans of soda, etc.) are kept track of and compared to the on-hand inventory taken at the conclusion of the week. Based on these numbers, the quality of product is able to be maintained and theft can be prevented or stopped.

The part of enterprise you are designing conceptual database for

The most critical operations involved in Rusty's Pizza is the accurate accounting and tracking of sales and the quantities of items used up during normal business operations. The key concentrations are comprised of the ingredients used as toppings for pizzas and non-perishable food or beverage items. Each pizza has a specified weight that is supposed to be distributed per specific ingredient. These weights are used as a tool to measure how accurately each topping is being used. The end result is a product that can be counted on to be consistent. Otherwise, customers would be lost due to varying qualities in the product.

It is for this reason that the focus of this database will be to design and implement a means of accounting for each sale, the items sold on each order, and the quantity of product used during normal business operations. This will enable good inventory control, and subsequently high product quality. To do this, items will need to be added into the inventory, orders must be placed, their items must be accounted for, and all of the ingredients used will need to be computable.

User Groups, Data views and Operations

There will be two types of users: managers and standard employees. The manager is the one responsible for recording items received from the suppliers. He is also the one responsible for entering the current inventory and reviewing the discrepancies between actual and ideal usage. To enable these activities an invoice entering interface and an inventory report must be available. Furthermore, there must also be a derivation of the data entry interface must allow for adjusting stored inventory to actual inventory.

As for the input of orders into the system, a standard employee needs an interface that will allow the selection of items to be sold and provide a subtotal and total. A further requirement will be the design of the interface so that it is very user friendly and has built-in safeguards to prevent improper or unintended actions.

Conceptual Database Design

Entity Set Description

Invoices

Description: Holds the name of the supplier from which a delivery was made.

Candidate Keys: N/A

Primary Key: N/A

Fields to be Indexed: N/A

Name	Supplier
Description	Name of the supplier
domain/type	string
Value-range	1->30
default value	Jordanos
null value allowed	no
Unique	no
single/multiple value	single
simple or composite	simple
Strong/Weak	

Inventory

Description: Stores the numerous names and units of measurement for all items purchased from vendors.

Candidate Keys: InvName

Primary Key: InvName

Fields to be Indexed: InvName, UnitOfMeasure

Name	InvName	UnitOfMeasure
Description	Name of standard inventory items	Unit of measure used for inventory shipments
domain/type	string	string
Value-range	1->30	2
default value	NULL	lb
null value allowed	no	no
Unique	yes	no
single/multiple val	single	single
simple or composite	simple	simple
Strong/Weak	strong	

Items

Description: Contains the name and price, of a single item sold to a customer. The items are not sold at the same rate or in the same form as when purchased from a vendor, so it also contains a conversion ratio so that inventory tracking can be achieved.

Candidate Keys: ItemName

Primary Key: ItemName

Fields to be Indexed: convRatio

Name	ItemName	ItemPrice	convRatio
Description	Name of the item. Used for display purposes.	Pricing for items to be sold	Ratio used to convert from stocked items to sold items.
domain/type	string	currency	number
Value-range	1 -> 45	NULL	1 -> 200
default value	NULL	NULL	NULL
null value allowed	no	yes	no
Unique	yes	no	no
single/multiple value	single	single	single
simple or composite	simple	simple	simple
Strong/Weak			

Order

Description: Contains the order number and order type used for differentiation between orders for a given day. Therefore, each is also give the order date.

Candidate Keys: OrderNumber

Primary Key: OrderNumber + OrderDate

Fields to be Indexed: N/A

Name	OrderNumber	OrderType	OrderDate
Description	Order Number used for identification during a given day.	Order Type, can be in-house, delivery, or pick-up	Date that the order is taken
domain/type	integer	string	date/time
Value-range	0 → 100000	0 ->9	
default value	n/a	In-house	today
null value allowed	no	no	no
Unique	no	no	no
single/multiple valu	single	single	single
simple or composit	simple	simple	simple
Strong/Weak			

Discounts

Description: Contains the list of discount codes, values, and a description of each. The discount number is used as input when an order is placed.

Candidate Keys: N/A

Primary Key: N/A

Fields to be Indexed: N/A

Name	DiscNumber	DiscValue	DiscDesc
Description	The local unit used for identying which discount is given	The amount of discount given	Description used to identify each discount
domain/type	integer	currency	integer
Value-range	0-500		30-Jan
default value	NULL	0	NULL
null value allowed	no	no	no
unique	no	no	no
single/multiple valu	single	single	single
simple or composit	simple	simple	simple
Strong/Weak	weak		

Pizza

Description: Contains the size, crust type, and price of every pizza ordered.

Candidate Keys: N/A

Primary Key: N/A

Fields to be Indexed: N/A

name	Size	Crust	Price
description	Contains the size of the pizza. Can be an individual, small, medium, or large.	Specifies the type of crust used. Can be thin or pan crust.	The price for a specific pizza.
domain/type	string	string	currency
Value-range	2->4	thin, pan	
default value	lrg	thin	NULL
null value allowed	no	no	no
unique	no	no	no
single/multiple value	single	single	single
simple or composite	simple	simple	simple
Strong/Weak	weak		

Relationship Set Description

hasDiscounts

Description: Links each order to the discounts that can be applied. Each order can have many discounts
Entity Sets: Order, Discounts
Cardinality: 1..*
Descriptive Field: DiscEdate
Participation Constraint: overlap, partial

hasItems

Description: Links orders to the Items contained in that order. One order can have many items
Entity Sets: Order, Items
Cardinality: 1..*
Descriptive Field: N/A
Participation Constraint

hasPizzas

Description: Links orders to the pizzas contained in that order. One order can have many pizzas
Entity Sets: Order, Pizza
Cardinality: 1..*
Descriptive Field: N/A
Participation Constraint

HasToppings

Description: Links pizzas to their toppings in the inventory. One pizza has many toppings
Entity Sets: Pizza, Inventory
Cardinality: *..1
Descriptive Field: quantity
Participation Constraint

ItemsFromInventory

Description: Maps items to the quantities used from the inventory. One item in inventory can have many item quantities
Entity Sets: Items, Inventory
Cardinality: *..1

Descriptive Field: N/A
Participation Constraint

IntoInventory

Description: Links incoming shipments to inventory. Hold details for each item purchased. Many invoices to one inventory
Entity Sets: Invoices, Inventory
Cardinality: *..1
Descriptive Field: dDate, costPerUnit, quantity.
Participation Constraint

Related Entity Set

Specialization

Subs, Pizza, Chicken, Wings, Items

Contains:

HasPizzas(Order, Pizza)

Partial participation / Overlap

HasSubs(Order, Subs)

Partial participation / Overlap

HasChicken(Order, Chicken)

Partial participation / Overlap

HasWings(Order, Wings)

Partial participation / Overlap

HasItems(Order, Items)

Partial participation / Overlap

Generalization

Inventory

Contains:

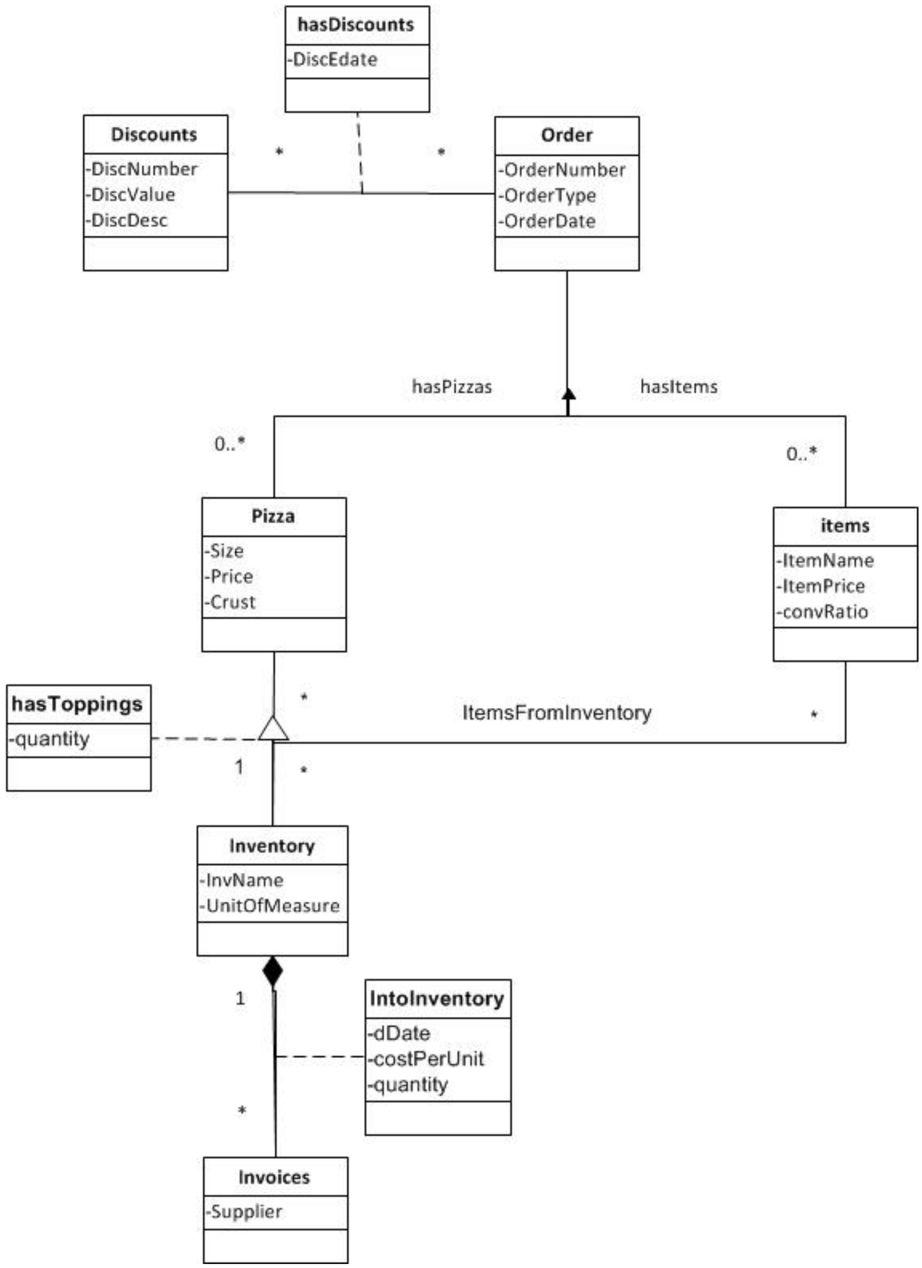
HasToppings(Pizza, Inventory)

Full participation / disjoint

ItemsFromInventory(Items, Inventory)

Full participation / disjoint

E-R Diagram


hasDiscounts
-DiscEdate
Discounts
-DiscNumber
-DiscValue
-DiscDesc
*
*
Order
-OrderNumber
-OrderType
-OrderDate
hasPizzas
hasItems
0..*
0..*
Pizza
-Size
-Price
-Crust
items
-ItemName
-ItemPrice
-convRatio
hasToppings
-quantity
*
ItemsFromInventory
*
1
*
Inventory
-InvName
-UnitOfMeasure
1
IntoInventory
-dDate
-costPerUnit
-quantity
*
Invoices
-Supplier

Phase II: Relational Model

E-R model and relational model

Description

The entity-relationship model discussed in the previous section is a very good method for designing the conceptual structure of a database. Unfortunately, the DBMS that are available have a design based on a different model. Therefore, it is important to convert an E-R model into a relational model so that the database can be implemented. The Relational model, conceptualized by Ted Codd, is one that has a strong foundation of mathematics. Because of this, it has been widely adopted, and is the basis for all mainstream database management systems.

Comparison

The entity-relationship model is one that focuses on the conceptual design rather than facts and intricate details. This model overlooks these details and instead has implied, relations, cardinality, and attributes. While this method is great for coming up with a conceptual design, it lacks the ability to specify the physical implementation of a database. For this, the relational model is well suited. The relationship between two entities can be converted into a relationship between relations, in which each record can be uniquely identified and data is easily retrievable. Through this method, the designer is able to understand the more intricate needs of the database by the implementation of constraints and tuple entries.

Conversion: E-R Model to Relational Model

Mapping of Regular Entity Types:

- For each strong entity type, create a relation that is comprised of all simple attributes.
- Choose one of the key attributes as a primary key.
- If the chosen attribute is composite, then the set of simple attributes form the key.

Mapping of Weak Entity Types:

- Create a relation comprised of all simple attributes.
- Include the primary key of owner entity as a foreign key.
- Combine the foreign key and a partial key to create the primary key.

One-to-One Mapping: 3 approaches

- Foreign Key: Add primary key of superclass entity to subclass entity.
- Merged Relation: Merge both entities into one relation having all simple attributes.
- Cross-Reference/Relationship Relation: Create a separate relation to hold foreign keys of both relations involved.

Many-to-Many Mapping:

- Cross-Reference

Mapping of Multivalued Attributes:

- Create a separate relation for the attribute

Mapping N-ary Relationship Types:

- Cross-Reference

Constraints

An entity constraint is one that requires a primary key not to be null. This helps ensure that a record can be uniquely identified. Otherwise we might not be able to distinguish them when referenced from other relations. Another constraint is that of the primary key or uniqueness constraint. This requires that any values within the attribute be unique. A referential constraint is one that maintains a link between two relations when one is pointed to by the foreign key field of the other. Removal of the record pointed at by the foreign key is not permitted. Another type of constraint is a check constraint. It consists of checking that a value entered satisfies the requirements of that attribute. A DBMS can differ from one to another as to how they enforce constraints but have fundamental similarities. In the case of ensuring referential integrity, a DBMS can choose between cascade deleting records that are pointed to by a record being deleted or choose to display a warning and not take any action.

Converted E-R model to Relational Model

Order

Attributes:

OrderId
Domain: unsigned integer: 1 to 2^32 – 1. Cannot be NULL
OrderNumber:
Domain: unsigned integer: 1 to 2^32 – 1. Cannot be NULL
OrderType
Domain: string. Cannot exceed 30 characters. Cannot be NULL
OrderDate
Domain: valid DateTime. Cannot be NULL

Constraints:

Primary Key: OrderId. It must be unique and not NULL.
Business Rule: OrderType must not exceed 30 characters and cannot be NULL.

Candidate Keys:

OrderId

Discounts

Attributes:

DiscountId
Domain: unsigned integer: 1 to 2^32 – 1. Cannot be NULL
DiscNumber
Domain: unsigned integer: 1 to 2^32 – 1. Cannot be NULL
DiscValue
Domain: Currency. Cannot be NULL
DiscEdate
Domain: : valid DateTime. Cannot be NULL
DiscDesc
Domain: string. Cannot exceed 30 characters. Cannot be NULL

Constraints:

Primary Key: DiscountId. It must be unique and not NULL.

Candidate Keys:

DiscountId

Pizza

Attributes:

PID
Domain: unsigned integer: 1 to 2^32 – 1. Cannot be NULL
OrderId

Domain: unsigned integer: 1 to 2^32 – 1. Cannot be NULL
pSize
Domain: String. Must be 2 or 3 characters. Cannot be NULL
Crust
Domain: string. Cannot be NULL
Price
Domain: Currency. Cannot be NULL

Constraints:
Primary Key: PID. It must be unique and not NULL.
Foreign Key: OrderId. It must exist in the Order Table.
Business Constraint: Size kept to 2 or 3 character for screen printing.
Business Constraint: Crust can be only one of two values: thin or pan.

Candidate Keys:
PID

Items

Attributes:
ItemId
Domain: unsigned integer: 1 to 2^32 – 1. Cannot be NULL
ItemName
Domain: string. Cannot exceed 45 characters. Cannot be NULL
ItemPrice
Domain: Currency.
InvId
Domain: unsigned integer: 1 to 2^32 – 1. Cannot be NULL
convRatio
Domain: unsigned integer: 1 to 2^32 – 1. Cannot be NULL

Constraints:
Primary Key: ItemId. It must be unique and not NULL.
Foreign Key: InvId. It must exist in the Inventory Table.
Business Constraint: ItemName cannot exceed 30 characters to allow display and must be unique

Candidate Keys:
ItemId

Inventory

Attributes:

InvId
Domain: unsigned integer: 1 to 2^32 – 1. Cannot be NULL
InvName
Domain: string. Cannot exceed 30 characters. Cannot be NULL
UnitOfMeasure
Domain: string. Between 2 and 4 characters. Cannot be NULL

Constraints:

Primary Key: InvId. It must be unique and not NULL.
Business Constraint: InvName cannot exceed 30 characters to allow display and must be unique
Business Constraint: UnitOfMeasure restricted to simple representations not to exceed 4 characters and no less than 2.

Candidate Keys:

InvId, InvName

Invoices

Attributes:

InvoiceId
Domain: unsigned integer: 1 to 2^32 – 1. Cannot be NULL
Supplier
Domain: string. Cannot exceed 30 characters. Cannot be NULL
Quantity
Domain: unsigned integer: 1 to 2^32 – 1. Cannot be NULL
costPerUnit
Domain: Currency. Cannot be NULL
InvId
Domain: unsigned integer: 1 to 2^32 – 1. Cannot be NULL
dDate
Domain: valid DateTime. Cannot be NULL

Constraints:

Primary Key: InvoiceId. It must be unique and not NULL.
Foreign Key: InvId. It must exist in the Inventory Table.

Candidate Keys:

InvoiceId, InvId

hasDiscount

Attributes:

HDID
Domain: unsigned integer: 1 to 2^32 – 1. Cannot be NULL
OrderId
Domain: unsigned integer: 1 to 2^32 – 1. Cannot be NULL
DiscountId
Domain: unsigned integer: 1 to 2^32 – 1. Cannot be NULL

Constraints:

Primary Key: HDID. It must be unique and not NULL.
Foreign Key: OrderId. It must exist in Order Table.
Foreign Key: DiscountId. It must exist in the Discount Table.

Candidate Keys:

HDID, OrderId, DiscountId

hasItems

Attributes:

HIID
Domain: unsigned integer: 1 to 2^32 – 1. Cannot be NULL
OrderId
Domain: unsigned integer: 1 to 2^32 – 1. Cannot be NULL
ItemId
Domain: unsigned integer: 1 to 2^32 – 1. Cannot be NULL

Constraints:

Primary Key: HIID. It must be unique and not NULL.
Foreign Key: OrderId. It must exist in the Order Table.
Foreign Key: ItemId. It must exist in the Item Table.

Candidate Keys:

HIID, OrderId, ItemId

hasToppings

Attributes:

HTID
Domain: unsigned integer: 1 to 2^32 – 1. Cannot be NULL
PID
Domain: unsigned integer: 1 to 2^32 – 1. Cannot be NULL
InvId
Domain: unsigned integer: 1 to 2^32 – 1. Cannot be NULL
Quantity

Domain: Double. Cannot be NULL

Constraints:

Primary Key: HTID. It must be unique and not NULL.
Foreign Key: PID. It must exist in the Pizza Table.
Foreign Key: InvId. It must exist in the Inventory Table.

Candidate Keys:

HTID, PID, InvId

Relation Instances

Order(**<u>OrderId</u>,** OrderNumber, OrderType, OrderDate**)**

Order			
OrderId	OrderNumber	OrderType	OrderDate
1	1	Eat in	10/17/2010
2	2	Take out	10/17/2010
3	3	Take out	10/17/2010
4	4	Eat in	10/17/2010
5	5	Take out	10/17/2010
6	6	Eat in	10/17/2010
7	7	Eat in	10/18/2010
8	8	Take out	10/18/2010
9	9	Eat in	10/18/2010
10	10	Take out	10/18/2010

Discounts(**DiscountId**, DiscNumber, DiscValue, DiscEdate, DiscDesc)

Discounts				
DiscountId	DiscNumber	DiscValue	DiscEdate	DiscDesc
1	1	$2.98	11/17/2011	#1 $3 of lrg pizza
2	2	$3.32	1/20/2011	#2 Lrg 1item 17.99
3	3	$5.00	2/15/2010	#3 free ind chz/pep
4	4	$4.42	3/13/2011	#4 lunch special 4
5	5	$3.28	9/5/2012	#5 lunch special 5
6	1	$9.02	3/13/2008	#1 free sm 1item
7	99	$9.55	8/20/2012	#99 free sm 2item phbk
8	20	$1.15	9/19/2019	#20 free reg wedge
9	13	$2.20	11/12/2013	#13 free spr wedge
10	14	$3.61	1/2/2011	#14 free 6pk

hasDiscount(<u>**HDID**</u>, **OrderId, DiscountId**)

hasDiscount		
HDID	OrderId	DiscountId
1	1	1
2	2	2
3	3	3
4	4	5
5	6	4
6	5	10
7	7	8
8	9	5
9	9	4
10	10	10

hasItems(**HIID, OrderId, ItemId**)

hasItems		
HIID	OrderId	ItemId
1	1	0
2	1	3
3	2	4
4	5	7
5	6	8
6	3	9
7	4	3
8	5	1
9	6	2
10	9	6

Items(**<u>ItemId</u>**, ItemName, ItemPrice, **InvId**, convRatio)

Items				
ItemId	**ItemName**	**ItemPrice**	**InvId**	**convRatio**
0	8 Piece Chicken	$16.99	7	0.50
1	25 Hot Wings Medium	$13.99	8	0.25
2	12 Hot Wings Medium	$8.99	8	0.15
3	Regular Drink	$1.76	9	1.00
4	Large Drink	$2.20	10	1.00
6	25 Hot Wings Hot	$13.99	8	0.25
7	25 Hot Wings Killer	$13.99	8	0.25
8	12 Hot Wings Hot	$8.99	8	0.15
9	12 Hot Wings Killer	$8.99	8	0.15

Pizza(**<u>PID</u>**, **OrderId**, Size, Crust, Price)

Pizza				
PID	OrderId	Size	Crust	Price
1	1	Lrg	Thin	$17.00
2	2	Med	Thin	$13.98
3	3	Lrg	Thin	$21.35
4	4	Ind	Thin	$5.42
5	5	Ind	Pan	$5.42
6	6	Lrg	Thin	$17.00
7	7	Sm	Thin	$9.08
8	8	Lrg	Thin	$21.35
9	9	Lrg	Pan	$16.00
10	9	Med	Thin	$14.50

hasToppings(<u>**HTID**</u>, **PID, InvId**, quantity)

hasToppings			
HTID	PID	InvId	quantity
1	1	14	0.25
2	3	14	0.25
3	4	14	0.1
4	6	15	0.15
5	6	14	0.35
6	6	11	0.4
7	6	13	0.12
8	6	12	0.68
9	6	16	0.55
10	6	17	0.12
11	2	14	0.25
12	5	15	0.3
13	5	15	0.2
14	7	13	0.1
15	7	13	0.1

Inventory(**<u>InvId</u>**, InvName, UnitOfMeasure)

Inventory		
InvId	**InvName**	**UnitOfMeasure**
7	Whole Chicken	lb
8	Wings	lb
9	16 oz. cup	cs
10	32 oz. cup	cs
11	Mus	lb
12	Oni	lb
13	Olv	lb
14	Pep	lb
15	Sal	lb
16	Bel	lb
17	Sau	lb

Invoices(**<u>InvoiceId</u>**, Supplier, quantity, costPerUnit, **InvId**, dDate)

Invoices					
InvoiceId	**Supplier**	**quantity**	**costPerUnit**	**InvId**	**dDate**
12	Cross Distributing	20	$3.62	7	11/19/2001
13	Cross Distributing	15	$2.72	8	11/19/2001
14	Jordanos	1	$16.98	9	11/19/2001
15	Jordanos	1	$20.68	10	11/19/2001
16	Jordanos	10	$2.98	11	11/19/2001
17	Jordanos	20	$2.13	12	11/19/2001
18	Jordanos	30	$3.13	13	11/19/2001
19	Jordanos	60	$6.72	14	11/19/2001
20	Jordanos	15	$3.38	15	11/19/2001
21	Jordanos	20	$4.25	16	11/19/2001
22	Jordanos	40	$6.10	17	11/19/2001

Queries

1. List all pizzas without extra toppings.
2. List all pizzas ordered.
3. List all discounts given.
4. List all take-out orders.
5. List suppliers that are not Jordanos.
6. List orders that do not have pizzas.
7. List inventory items bought that cost at least $20 per unit.
8. List orders with 2 or more pizzas.
9. List largest discount available.
10. List Inventory items received before October 1, 2009.

Query Forms

1. List all pizzas without extra toppings.

Relational Algebra

$\Pi_{p2.OrderId \wedge p2.Size \wedge p2.Crust \wedge p2.price}(\partial_{p2.PID = p2.PID}(p2 \leftarrow \text{Pizza X } (\pi_{PID}(p1 \leftarrow \text{Pizza}) - \pi_{PID}(\text{hasToppings}))))$

Tuple Relational Calculus

$\{P|Pizza(P) \wedge \neg(\exists h)(hasToppings(h) \wedge h.PID = P.PID)\}$

Domain Relational Calculus

$\{<a,b,c,d,e>|Pizza(a,b,c,d, e) \wedge \neg(hasToppings(\_,=a, \_, \_))\}$

2. List all pizzas ordered.

Relational Algebra

$\pi_{PID, \wedge OrderId \wedge Size \wedge Price \wedge Crust}(Pizza)$

Tuple Relational Calculus

$\{P|Pizza(P)\}$

Domain Relational Calculus

{<a,b,c,d, e>|Pizza(a,b,c,d,e)}

3. List all discounts given.

Relational Algebra

$\pi_{DiscNumber \wedge DiscValue \wedge DiscDesc}(\partial_{h.DiscountId = D.DiscountId}(D \leftarrow Discounts) \text{ X } (h \leftarrow hasDiscount))$

Tuple Relational Calculus

$\{D | Discounts(D) \wedge (\exists h)(hasDiscounts(h) \wedge h.DiscountId = D.DiscountId)\}$

Domain Relational Calculus

{<a,b,c,d,e>|Discounts(a,b,c,d,e) ∧ hasDiscount(_, _, =a)}

4. List all take-out orders.

Relational Algebra

$\pi_{OrderNumber}(\partial_{OrderType = \text{“take-out”}}(Order)$

Tuple Relational Calculus

$\{O | Orders(O) \wedge O.OrderType = \text{“Take-Out”}\}$

Domain Relational Calculus

{<a,b,c,d>|Order(a,b,=”Take-Out”,d)}

5. List suppliers that are not Jordanos.

Relational Algebra

$\pi_{Supplier}(Invoices) - \partial_{Supplier = \text{“Jordanos”}}(\pi_{Supplier}(Invoices))$

Tuple Relational Calculus

$\{I.Supplier | Invoices(I) \wedge (\exists i2)(Invoices(I2) \wedge I2.Supplier = \text{“Jordanos”} \wedge I2.InvoiceId = I.InvoiceId)\}$

Domain Relational Calculus

{<s>|Invoices(_, ≠ “Jordanos”,_,_,_,_)}

6. List orders that do not have pizzas.

Relational Algebra

$\pi_{O2.OrderId \wedge O2.OrderNumber \wedge O2.OrderType \wedge O2.OrderDate}(\partial_{O2.OrderId = O.OrderId}(O2 \leftarrow Order\ X\ (\pi_{O.OrderId}(O \leftarrow Order) - \pi_{OrderId}(Pizzas)))$

Tuple Relational Calculus

$\{O | Orders(O) \wedge \neg(\exists P)(Pizza(P) \wedge P.OrderId = O.OrderId)\}$

Domain Relational Calculus

{<a,b,c,d>|Order(a,b,c,d) ∧ ¬Pizza(_, =a, _, _, _)}

7. List inventory items bought that cost at least $20 per unit.

Relational Algebra

$\pi_{i2.InvName}(\partial_{I2.InvId = I.InvId}((I2 \leftarrow Inventory)\ X\ \pi_{i1.InvId}(\partial_{CostPerUnit \geq \$20}(I1 \leftarrow Invoices))))$

Tuple Relational Calculus

$\{I | Inventory(I) \wedge (\exists I2)(Invoice(I2) \wedge I2.CostPerUnit \geq \$20 \wedge I2.InvId = I.InvId)\}$

Domain Relational Calculus

{<a>|(∃i)Inventory(I, a, _) ∧ Invoices(_, _, _, ≥ $20, =I, _)}

8. List orders with 2 or more pizzas.

Relational Algebra

$\pi_{O.OrderNumber}(\partial_{O.OrderId = p1.OrderId}((O \leftarrow Order)\ X\ \pi_{p1.OrderId}(\partial_{p1.OrderId = p2.OrderId \wedge p1.PID \neq p2.PID}((p1 \leftarrow Pizza)\ X\ (p2 \leftarrow Pizza)))))$

Tuple Relational Calculus

$\{O | Orders(O) \wedge (\exists P)(Pizza(P) \wedge P.OrderId = O.OrderId \wedge (\exists P2)(Pizza(P2) \wedge P2.OrderId = O.OrderId \wedge P2.PID \neq P.PID))\}$

Domain Relational Calculus

{<a,b,c,d>|Order(a,b,c,d) ∧ (∃p)(Pizza(p, a, _, _, _) ∧ Pizza(≠ p, a, _, _, _))}

9. List largest discount available.

Relational Algebra

$D * (\Im_{max(DiscValue)}Discount)$

Tuple Relational Calculus

$\{D | Discounts(D) \wedge \neg(\exists D1)(Discount(D1) \wedge D1.DiscValue > D.DiscValue)\}$

Domain Relational Calculus

$\{<a,b,c,d,e> | Discounts(a,b,c,d,e) \wedge \neg Discounts(\_, \_, >c, \_, \_)\}$

10. List Inventory items received before October 1, 2009.

Relational Algebra

$\pi_{InvId \wedge InvName \wedge UnitOfMeasure}(\partial_{I2.InvId = I1.InvId}((I2 \leftarrow Inventory)\ X\ \pi_{I1.InvId}(\partial_{I1.dDate < \text{“10/01/2009”}}(Invoice))))$

Tuple Relational Calculus

$\{I | Inventory(I) \wedge (\exists I2)(Invoice(I2) \wedge I2.dDate < \text{“10/01/2009”} \wedge I2.InvId = I.InvId)\}$

Domain Relational Calculus

$\{<a> | (\exists i) Inventory(i, a, \_) \wedge Invoices(\_, \_, \_, \_, i, < \text{“10/01/2009”})\}$

Phase III: Oracle Implementation

SQL*PLUS

SQL*Plus is a command-line interface that connects to an Oracle database. It primary purposes include allowing a user to create, edit, and view the results of queries. The Developers are able to inspect the structure of the database along with the ability to enter and execute PL/SQL code. The interface also incorporates a data definition language (DDL) which allows the creation of database tables, views, indexes, and other objects. Furthermore, database administrators have added functionality so that commands like SHUTDOWN and STARTUP can be given making SQL*PLUS a valuable tool no matter what level of user a person may be.

Oracle Schema Objects

Tables

Tables are the objects that hold all user-accessible data within an Oracle database. Each table is comprised of a series of columns and rows. The columns represent the different types of data that the table holds, whereas the rows represent each individual instance of a set of data of the given types.

Views

Views represent a specific set of data that spans one or more tables. Views do not store any actual data they just retrieve the data from the tables that they reside in by calling upon a stored query. Views and tables share the ability to be queried, updated, inserted into, and deleted from, assuming that the constraints of the underlying tables are met. An additional advantage to a view is the ability to restrict access to specific rows and columns of a table along with implementing information hiding techniques.

Sequences

Sequences are objects within the database that enable multiple users to generate unique integers. Frequently used as a method to generate primary key values.

Synonyms

A synonym is an alias for a table, view, sequence, function, procedure, or other object type including other synonyms. It only requires a definition in the data dictionary without any other storage allocation.

Dimensions

A dimension is a structure that allows users to answer business questions by putting data into categories.

Indexes

Indexes are used in an effort to aid in increasing the performance of data retrieval. Indexes point to the location of a specific attribute or set of attributes. Oracle automatically uses indexes for the primary key of tables and the designer is able to create other indexes for fields that may be accessed frequently. After instantiation, Oracle automatically maintains all indexes.

Procedures/Functions

Stored procedures are a set of SQL statements that have been assigned a name. It is stored in the database in a compiled form. The primary difference between a procedure and a function is that a procedure can return many values while a function returns only one.

Database Links

Database links are pointers that define a one-way communication path between an Oracle database server and another database server. It is implemented as a data dictionary entry and in order to access the link, the local database must be the one that contains that entry.

Clusters

Tables that share common attributes are stored in the same location. Because of this, related records are physically stored together and disk access time is improved. While clusters serve an important role, end-users and application designers are not necessarily aware of their existence since access and implementation is unaffected.

Schema Objects

cjOrders

```
CS342 SQL> desc cjOrders;
Name                                                      Null?    Type
 ----------------------------------------------------- -------- -------------
------
 ORDERID                                                   NOT NULL NUMBER(30)
 ORDERNUMBER                                                        NUMBER(4)
 ORDERTYPE                                                          VARCHAR2(30)
 ORDERDATE                                                          DATE

CS342 SQL> select * from cjOrders;

   ORDERID ORDERNUMBER ORDERTYPE                      ORDERDATE
---------- ----------- ------------------------------ ---------
         1           1 Eat-in                         17-OCT-10
         2           2 Take-out                       17-OCT-10
         3           3 Take-out                       17-OCT-10
         4           4 Eat-in                         17-OCT-10
         5           5 Take-out                       17-OCT-10
         6           6 Eat-in                         17-OCT-10
         7           7 Eat-in                         18-OCT-10
         8           8 Take-out                       18-OCT-10
         9           9 Eat-in                         18-OCT-10
        10          10 Take-out                       18-OCT-10

10 rows selected.
```

cjDiscounts

```
CS342 SQL> desc cjDiscounts;
 Name                                                      Null?    Type
 ----------------------------------------------------- -------- -------------
-----------------------
 DISCOUNTID                                                NOT NULL NUMBER(30)
 DISCNUMBER                                                         NUMBER(4)
 DISCVALUE                                                          NUMBER(10,2)
 DISCEDATE                                                          DATE
 DISCDESC                                                           VARCHAR2(30)

CS342 SQL> select * from cjDiscounts;

DISCOUNTID DISCNUMBER  DISCVALUE DISCEDATE DISCDESC
---------- ---------- ---------- --------- ------------------------------
         1          1       2.98 15-NOV-11 #1 $3 of lrg pizza
         2          2       3.32 15-JAN-11 #2 Lrg 1item 17.99
         3          3          5 15-FEB-10 #3 free ind chz/pep
         4          4       4.42 15-MAR-11 #4 lunch special 4
         5          5       3.28 15-SEP-12 #5 lunch special 5
         6          1       9.02 15-MAR-08 #1 free sm 1item
         7         99       9.55 15-AUG-12 #99 free sm 2item phbk
         8         20       1.15 15-SEP-19 #20 free reg wedge
         9         13        2.2 15-NOV-13 #13 free spr wedge
        10         14       3.61 15-JAN-11 #14 free 6pk

10 rows selected.
```

cjPizza

```
CS342 SQL> desc cjPizza;
 Name                                                      Null?    Type
 --------------------------------------------------------- -------- -------------
-----------------------
 PID                                                       NOT NULL NUMBER(30)
 ORDERID                                                            NUMBER(30)
 PSIZE                                                              VARCHAR2(4)
 CRUST                                                              VARCHAR2(4)
 PRICE                                                              NUMBER(10,2)

CS342 SQL> select * from cjPizza;

       PID    ORDERID PSIZ CRUS       PRICE
---------- ---------- ---- ---- -----------
         1          1 Lrg  Thin      $17.00
         2          2 Med  Thin      $13.98
         3          3 Lrg  Thin      $21.35
         4          4 Ind  Thin       $5.42
         5          5 Ind  Pan        $5.42
         6          6 Lrg  Thin      $17.00
         7          7 Sm   Thin       $9.08
         8          8 Lrg  Thin      $21.35
         9          9 Lrg  Pan       $16.00
        10          9 Med  Thin      $14.50

10 rows selected.
```

cjInventory

```
CS342 SQL> desc cjInventory;
 Name                                                  Null?    Type
 ----------------------------------------------------- -------- -------------
-----------------------
 INVID                                                 NOT NULL NUMBER(30)
 INVNAME                                                        VARCHAR2(30)
 UNITOFMEASURE                                                  VARCHAR2(4)

CS342 SQL> select * from cjInventory;

     INVID INVNAME         UNIT
---------- --------------- ----
         7 Whole Chicken   lb
         8 Wings           lb
         9 16 oz. cup      cs
        10 32 oz. cup      cs
        11 Mus             lb
        12 Oni             lb
        13 Olv             lb
        14 Pep             lb
        15 Sal             lb
        16 Bel             lb
        17 Sau             lb

11 rows selected.
```

cjItems

```
CS342 SQL> desc cjItems;
 Name                                                      Null?    Type
 --------------------------------------------------------- -------- -------------
-----------------------
 ITEMID                                                    NOT NULL NUMBER(30)
 INVID                                                              NUMBER(4)
 ITEMNAME                                                           VARCHAR2(45)
 ITEMPRICE                                                          NUMBER(10,2)
 CONVRATIO                                                          NUMBER(5,3)

CS342 SQL> select * from cjItems;

    ITEMID      INVID ITEMNAME                         ITEMPRICE  CONVRATIO
---------- ---------- ------------------------------ ---------- ----------
         0          7 8 Piece Chicken                     16.99         .5
         1          8 25 Hot Wings Medium                 13.99        .25
         2          8 12 Hot Wings Medium                  8.99        .15
         3          9 Regular Drink                        1.76          1
         4         10 Large Drink                           2.2          1
         5          8 12 Hot Wings Spicy Barbeque          8.99        .15
         6          8 25 Hot Wings Hot                    13.99        .25
         7          8 25 Hot Wings Killer                 13.99        .25
         8          8 12 Hot Wings Hot                     8.99        .15
         9          8 12 Hot Wings Killer                  8.99        .15

10 rows selected.
```

cjInvoice

```
CS342 SQL> desc cjInvoice;
 Name                                                      Null?    Type
 ----------------------------------------------------- -------- -------------
------
 INVOICEID                                                 NOT NULL NUMBER(30)
 INVID                                                     NOT NULL NUMBER(30)
 SUPPLIER                                                  NOT NULL VARCHAR2(30)
 QUANTITY                                                  NOT NULL NUMBER(4)
 COSTPERUNIT                                               NOT NULL NUMBER(10,2)
 DDATE                                                              DATE

CS342 SQL> select * from cjInvoice;

 INVOICEID      INVID       SUPPLIER          QUANTITY COSTPERUNIT DDATE
---------- ---------- -------------------- ---------- ----------- ---------
        12          7 Cross Distributing           20        3.62 15-OCT-10
        13          8 Cross Distributing           15        2.72 15-OCT-10
        14          9 Jordanos                      1       16.98 15-OCT-10
        15         10 Jordanos                      1       20.68 15-OCT-10
        16         11 Jordanos                     10        2.98 15-OCT-10
        17         12 Jordanos                     20        2.13 15-OCT-10
        18         13 Jordanos                     30        3.13 15-OCT-10
        19         14 Jordanos                     60        6.72 15-OCT-10
        20         15 Jordanos                     15        3.38 15-OCT-10
        21         16 Jordanos                     20        4.25 15-OCT-10
        22         17 Jordanos                     40         6.1 15-OCT-10
        23         17 Jordanos                     10         6.2 12-OCT-99

12 rows selected.

CS342 SQL> spool off
```

cjhasItems

```
CS342 SQL> desc cjhasItems;
 Name                                                  Null?    Type
 ----------------------------------------------------- -------- -------------
-----------------------
 HIID                                                  NOT NULL NUMBER(30)
 ORDERID                                                        NUMBER(30)
 ITEMID                                                         NUMBER(30)

CS342 SQL> select * from cjhasItems;

      HIID    ORDERID     ITEMID
---------- ---------- ----------
         1          1          0
         2          1          3
         3          2          4
         4          5          7
         5          6          8
         6          3          9
         7          4          3
         8          5          1
         9          6          2
        10          9          6

10 rows selected.
```

cjhasToppings

```
CS342 SQL> desc cjhasToppings;
 Name                                                      Null?    Type
 ----------------------------------------------------- -------- -------------
-----------------------
 HTID                                                      NOT NULL NUMBER(30)
 PID                                                                NUMBER(30)
 INVID                                                              NUMBER(30)
 QUANTITY                                                           NUMBER(6,4)

CS342 SQL> select * from cjhasToppings;

      HTID        PID      INVID   QUANTITY
---------- ---------- ---------- ----------
         1          1         14        .25
         2          3         14        .25
         3          4         14         .1
         4          6         15        .15
         5          6         14        .35
         6          6         11         .4
         7          6         13        .12
         8          6         12        .68
         9          6         16        .55
        10          6         17        .12
        11          2         14        .25
        12          5         15         .3
        13          5         15         .2
        14          7         13         .1
        15          7         13         .1

15 rows selected.
```

cjhasDiscount

```
CS342 SQL> desc cjhasDiscount;
 Name                                                  Null?    Type
 ----------------------------------------------------- -------- -------------
-----------------------
 HDID                                                  NOT NULL NUMBER(30)
 ORDERID                                                        NUMBER(30)
 DISCOUNTID                                                     NUMBER(30)

CS342 SQL> select * from cjhasDiscount;

      HDID    ORDERID DISCOUNTID
---------- ---------- ----------
         1          1          1
         2          2          2
         3          3          3
         4          4          5
         5          6          4
         6          5         10
         7          7          8
         8          9          5
         9          9          4
        10         10         10

10 rows selected.
```

SQL Queries

1. **List all pizzas without extra toppings**.

```
select *
from  cjpizza
        where not exists(select * from cjhasToppings where
        cjhasToppings.PID = cjPizza.PID);
```

```
       PID    ORDERID PSIZ CRUS      PRICE
---------- ---------- ---- ---- ----------
         9          9 Lrg  Pan          16
        10          9 Med  Thin       14.5
         8          8 Lrg  Thin      21.35
```

2. **List all pizzas ordered**.

```
select p.*
from cjPizza p
where exists
   (select o.OrderId from cjOrders o
        where o.OrderId = p.OrderId);
```

```
       PID    ORDERID PSIZ CRUS      PRICE
---------- ---------- ---- ---- ----------
         1          1 Lrg  Thin         17
         2          2 Med  Thin      13.98
         3          3 Lrg  Thin      21.35
         4          4 Ind  Thin       5.42
         5          5 Ind  Pan        5.42
         6          6 Lrg  Thin         17
         7          7 Sm   Thin       9.08
         8          8 Lrg  Thin      21.35
         9          9 Lrg  Pan          16
        10          9 Med  Thin       14.5
```

3. **List all discounts given**.

```
Select d.*
from cjdiscounts d
where exists
   (select h.DiscountId from cjhasDiscount h
         where h.DiscountId = d.DiscountId);
```

```
DISCOUNTID DISCNUMBER  DISCVALUE DISCEDATE DISCDESC
---------- ---------- ---------- --------- -----------------------------
-
         1          1       2.98 15-NOV-11 #1 $3 of lrg pizza
         2          2       3.32 15-JAN-11 #2 Lrg 1item 17.99
         3          3          5 15-FEB-10 #3 free ind chz/pep
         5          5       3.28 15-SEP-12 #5 lunch special 5
         4          4       4.42 15-MAR-11 #4 lunch special 4
        10         14       3.61 15-JAN-11 #14 free 6pk
         8         20       1.15 15-SEP-19 #20 free reg wedge
```

4. **List all take-out orders**.

```
select *
from cjOrders
   where cjOrders.OrderType = 'Take-out';
```

```
   ORDERID ORDERNUMBER ORDERTYPE                      ORDERDATE
---------- ----------- ------------------------------ ---------
         2           2 Take-out                       17-OCT-10
         3           3 Take-out                       17-OCT-10
         5           5 Take-out                       17-OCT-10
         8           8 Take-out                       18-OCT-10
        10          10 Take-out                       18-OCT-10
```

5. **List suppliers that are not Jordanos**.

```
select distinct inv.Supplier
from cjInvoice inv
   where inv.Supplier != 'Jordanos';
```

```
SUPPLIER
------------------------------
Cross Distributing
```

6. **List orders that do not have pizzas.**

```
select o.*
from cjOrders o
where not exists
   (select p.* from cjPizza p
         where p.OrderId = o.OrderId);
```

```
   ORDERID ORDERNUMBER ORDERTYPE                      ORDERDATE
---------- ----------- ------------------------------ ---------
        10          10 Take-out                       18-OCT-10
```

7. **List inventory items bought that cost at least $20 per unit.**

```
select i.InvName, inv.CostPerUnit
from cjInventory i inner join cjInvoice inv on i.InvId = inv.InvId
where inv.CostPerUnit >= 20;
```

```
INVNAME                        COSTPERUNIT
------------------------------ -----------
32 oz. cup                           20.68
```

8. **List orders with 2 or more pizzas.**

```
select distinct o.*
from cjOrders o, cjPizza p
where o.OrderId = p.OrderId and exists
   (select * from cjPizza where cjPizza.PID != p.PID and o.OrderID =

         cjPizza.OrderId);
```

```
   ORDERID ORDERNUMBER ORDERTYPE                      ORDERDATE
---------- ----------- ------------------------------ ---------
         9           9 Eat-in                         18-OCT-10
```

9. **List largest discount available**.

```
select d.*
from cjDiscounts d
where not exists
   (select * from cjDiscounts
         where cjDiscounts.DiscValue > d.DiscValue);
```

```
DISCOUNTID DISCNUMBER  DISCVALUE DISCEDATE DISCDESC
---------- ---------- ---------- --------- ------------------------------
-
         7         99       9.55 15-AUG-12 #99 free sm 2item phbk
```

10. **List Inventory items received before October 1, 2009**.

```
select i.InvName
from cjInventory i
where exists
   (select * from cjInvoice
         where cjInvoice.InvId = i.InvId
                     and cjInvoice.dDate < (to_date('10/01/2009',
         'mm/dd/yyyy')));
```

```
INVNAME
------------------------------
Sau
```

11. **Create a table that lists all order ID'S, dates, Pizza ID's, pizza sizes, and cost.**

```
create table cjNewOrders
As (Select cjOrders.OrderId, PID, OrderDate, pSize, Price from
cjOrders left outer join cjPizza on cjOrders.OrderId =
cjPizza.OrderId);
```

```
Table created.

CS342 SQL> desc cjNewOrders
 Name                                                      Null?    Type
 ----------------------------------------------------------- -------- ----------
--------------------------
 ORDERID                                                            NUMBER(30)
 PID                                                                NUMBER(30)
 ORDERDATE                                                          DATE
 PSIZE                                                              VARCHAR2(4)
 PRICE                                                              NUMBER(10,2)

CS342 SQL> select * from cjNewOrders
  2  ;

   ORDERID        PID ORDERDATE PSIZ      PRICE
---------- ---------- --------- ---- ----------
         1          1 17-OCT-10 Lrg          17
         2          2 17-OCT-10 Med       13.98
         3          3 17-OCT-10 Lrg       21.35
         4          4 17-OCT-10 Ind        5.42
         5          5 17-OCT-10 Ind        5.42
         6          6 17-OCT-10 Lrg          17
         7          7 18-OCT-10 Sm         9.08
         8          8 18-OCT-10 Lrg       21.35
         9          9 18-OCT-10 Lrg          16
         9         10 18-OCT-10 Med        14.5
        10            18-OCT-10

11 rows selected.
```

12. **Count the number of orders that have pizzas.**

```
Select cjOrders.OrderDate, count(cjOrders.OrderId) "Orders Per Day"
from cjOrders left outer join cjPizza on cjOrders.OrderId =
cjPizza.OrderId
where PID is not null
group by OrderDate;


ORDERDATE Orders Per Day
--------- --------------
17-OCT-10              6
18-OCT-10              4
```

Data Loading

Methods

Insert

```
INSERT INTO "table_name" ("column1", "column2", ...)
VALUES ("value1", "value2", ...)

 INSERT INTO "table1" ("column1", "column2", ...)
SELECT "column3", "column4", ...
FROM "table2"
```

Update

```
UPDATE "table_name"
SET "column_1" = [new value]
WHERE {condition}
```

Delete

```
DELETE FROM "table_name"
WHERE {condition}
```

DBMS Data Loading Utilities:

Oracle Data Pump

Oracle Data Pump has been released with the second release of 11g. It enables fast bulk data movement between Oracle databases. It includes integrated export and import utilities.

SQL Loader

SQL Loader is a high-speed utility that loads data from external files. Accepts data in a variety of formats, can filter data, and load it into multiple tables during one execution.

Java DataLoader

The java DataLoader is a java program that takes preformatted text files and inserts the data into database tables. The program integrates user password authentication and allows the user to choose the data separating character. The usefulness of this program is demonstrated in its ability to quickly reload the data from a table upon a loss or mass insertion.

Modifications

To increase the user friendliness of this application, we have been instructed to make modifications the the Java DataLoader. My first thought was regarding the issue regarding database connections. The connection must be hard coded and makes it impossible to update different servers without modifying the code. That is why I made these changes:

```
      --String url added to allow passing of local variable url.

    public DataLoader(String user, String passwd , String url)
    DataLoader ldr = new DataLoader(user, passwd, url);

      --Code added to allow passing in a new connection at runtime.
      --Default connection was also set to helios.
    String tmp = null;
    char UserSel;

    String url = "jdbc:oracle:thin:@helios.cs.csubak.edu:1521:orcl";
    tmp = ScreenIO.promptForString("New Connection String Desired? ");
    tmp.toUpperCase();
    UserSel = tmp.charAt(0);
    if ( UserSel == 'Y' || UserSel == 'y')
    {
       url = ScreenIO.promptForString("Enter the new connection string");
}
```

Phase IV: Stored Procedures

Common PL/SQL and MS Trans-SQL Features

Components:

While the exact implementation of PL/SQL and MS Trans-SQL have significant differences, the fact that both share the Structured Query Language as their base language, lends to the separate creation of these corresponding database abilities.

Ability to Create:

- Tables
- Constraints
- Functions
- Procedures
- Cursors
- Triggers
- Packages

Purpose of Stored Subprograms

A stored program is designed to limit the need for compilation for queries that are executed often. The subprograms are stored in a compiled state so that system resource demands are minimized and transaction times are decreased due to not requiring compilation at runtime and reduced network traffic.

Benefits of Subprogram Calls

Benefits include:

- Modular: it is easier to troubleshoot a subprogram than it is to troubleshoot large section of code within the graphical user interface.
- The stored procedures can be modified without any change to deployed front-end software.
- Easier to code front-end applications due to a separation of client and server side functions.
- Server memory usage is minimized by lessening the number of times a query must be compiled
- No data transfer required during query processing, request is sent, then results are received.
- Enhanced security controls.

Oracle PL/SQL

Program structure

Declaration section: Space where variables, cursors, types and local subprograms are stored.

Executions section: Contained within a BEGIN and END statement, this section contains all code that the main execution of the program consists of. This is the only section required.

Exception section: Space where exception handling procedures are coded.

Basic Format:

```
Create [or Replace] <Procedure, Trigger, or Function> program_name
        [Declaration statements]
BEGIN
        <Execution statements>
EXCEPTION
         [Exception statements]
End;
/
```

Control statements

Control statements are features of programming languages that perform computations or actions depending on the conditions present.

IF – THEN:

```
IF condition THEN
  sequence_of_statements
END IF;
```

IF-THEN-ELSE

```
IF condition THEN
  sequence_of_statements1
ELSE
  sequence_of_statements2
END IF;
```

IF-THEN-ELSIF

```
IF condition1 THEN
  sequence_of_statements1
ELSIF condition2 THEN
  sequence_of_statements2
ELSE
  sequence_of_statements3
END IF;
```

CASE STATEMENT

```
[<<label_name>>]
CASE selector
  WHEN expression1 THEN sequence_of_statements1;
  WHEN expression2 THEN sequence_of_statements2;
  WHEN expressionN THEN sequence_of_statementsN;
 [ELSE sequence_of_statementsN+1;]
END CASE [label_name];
```

Cursors

A cursor is a control structure that allows for the iterative traversal and processing of records from a result set.

Syntax

```
Declare
        Cursor cursor_name[parameters]
        IS <select statement>
Usage:
        FOR x in cursor_name LOOP
                Statements
        END LOOP:
```

Stored Procedure

A stored procedure is a subroutine that is available to applications accessing a database. These procedures are stored in the database's data dictionary. For improving database performance, stored procedures are compiled once, then stored in executable form.

Syntax

```
Create [or Replace] Procedure procedure_name [(variable_name IN|OUT, …)]
IS or AS
        [Declaration statements]
BEGIN
        <Execution statements>
EXCEPTION
         [Exception statements]
End;
/
```

Stored Function

Stored functions are very similar to stored procedures in the ways that they are created an ran. The difference is that a function must have a return variable.

Syntax

```
Create [or Replace]  Function function_name [(variable_name IN|OUT, …)]
Return datatype;
IS or AS
        [Declaration statements]
BEGIN
        <Execution statements>
EXCEPTION
         [Exception statements]
End;
/
```

Package

An Oracle package is a schema object that groups logically related PL/SQL subprograms, types and items together. It is generally comprised of two parts: the specification and the body. The specification declares the types, variables, exceptions, cursors, and subprograms available for use. The body is where those items are implemented.

Syntax

```
Create [or Replace] PACKAGE package_name
IS or AS
        Procedure names;
        Function names;
End [package_name];

Create [or Replace] PACKAGE BODY package_name
IS or AS
```

```
        Procedure names;
        Function names;
        …
[BEGIN]
        Statements
END [package_name];
```

Trigger

Triggers are a form of procedure that is run implicitly upon the occurrence of a predefined event. These events include: insert, delete, and update events.

Syntax

```
Create [or Replace]  TRIGGER trigger_name [(variable_name IN|OUT, …)]
IS or AS
        [Declaration statements]
BEGIN
        <Execution statements>
EXCEPTION
         [Exception statements]
End;
/
```

PL/SQL Stored Procedures and Functions

cjInsHI(OrderId, ItemId)

This procedure takes two integer values in as input and stores those values in the cjhasItems table with the first integer referencing the OrderId and the second referencing the ItemId.

```
CREATE OR REPLACE PROCEDURE cjInsHI(OrderIdent IN number, item IN number) is
BEGIN
INSERT INTO cjhasItems(OrderId, ItemId) values(OrderIdent,item);
EXCEPTION
    when others then
       raise_application_error( -1269, 'An error occurred in ' || SQLCODE ||
                        '-ERROR-' || SQLERRM );
END cjInsHi ;
/
```

cjDeletePizza(PID)

This procedure takes a valid PID number and deletes the corresponding record in cjPizza where the PID is located.

```
CREATE OR REPLACE PROCEDURE cjDeletePizza(id_in IN number) is
BEGIN
DELETE FROM cjPizza
where PID = id_in;
EXCEPTION
    when others then
       raise_application_error( -1269, 'An error occurred in ' || SQLCODE ||
                        '-ERROR-' || SQLERRM );
END cjDeletePizza;
/
```

cjHavgPizza(integer)

This function takes an integer value as input and returns the average of the top ‘n’ pizza prices from the cjPizza table.

```
CREATE OR REPLACE FUNCTION cjHavgPizza(n IN number)
RETURN number
IS
      cnum number(9,2) := 0.0;
      snum number(9,2) := 0.0;
      cursor c1 is
            select Price
            from cjPizza
            Order by Price Desc;
BEGIN
      OPEN c1;
      FOR i IN 1 .. n LOOP
            fetch c1 into cnum;
            snum := snum + cnum;
      END LOOP;
      CLOSE c1;
      RETURN snum/n;

EXCEPTION
    when others then
       raise_application_error( -1269, 'An error occurred in ' || SQLCODE ||
                        '-ERROR-' || SQLERRM );
END cjHavgPizza;
/
```

cjOrder_update_tr

This trigger fires after a record has been updated and stores the current date, a string containing the old OrderId and OrderNumber, and a string containing the new OrderId, OrderNumber in the cjLogTable.

```
CREATE OR REPLACE TRIGGER cjOrder_update_tr
after update
ON cjOrders
FOR EACH ROW
DECLARE
      oldV varchar2(40);
      newV varchar2(40);
BEGIN
      select concat(:old.OrderId, :old.OrderNumber)
    into oldV
      from cjOrders;
      select concat(:new.OrderId, :new.OrderNumber)
    into newV
      from cjOrders;
      INSERT INTO cjLogTable
      VALUES (sysdate,oldV, newV);
END;
/
```

Phase V: GUI Design and Implementation

User Group Activities

There is only one group that this program has been developed to facilitate: Managers. Managers are responsible for maintaining an accurate count of the store's inventory and as such, must be able to add, edit, and view items that have been received from suppliers. These invoices are associated by vendor, date, and invoice numbers and can be differentiated by those fields accordingly.

The original goal for this project was to design applications that would cover the aspects of inventory management as well as the placement of orders with their associated component allocated to their corresponding ingredients in the inventory. Unfortunately, due to complications with developing using a newer version of oracle and incorporating into the development techniques that had yet to be discovered, the program has been cut short. Therefore, the program only incorporates the inventory management aspect of the originally conceived project.

Relations Views and Subprograms

The following tables have been used during the implementation of this program. They share a 1:N relationship with one item in cjInventory having many items in cjInvoice associated to it.

cjInvoice : has foreign key field InvId that relates cjInvoice to cjInventory.

cjInventory : root table.

cjInvView : view was created but never implemented.

For implementation purposes, a trigger has been created for the cjInvoice table so that writing the front-end code is easier.

```
CREATE OR REPLACE TRIGGER cjInvoices_bir
BEFORE INSERT or UPDATE
ON cjInvoice
REFERENCING NEW AS NEW
FOR EACH ROW
WHEN(new.InvoiceId IS NULL)
BEGIN
	SELECT cjInvoices_seq.NEXTVAL
	INTO   :new.InvoiceId
	FROM   dual;
END;
/
```

The following sequence has also been created for use by the trigger.

```
CREATE SEQUENCE  cjInvoices_seq;
```

Application Screenshots

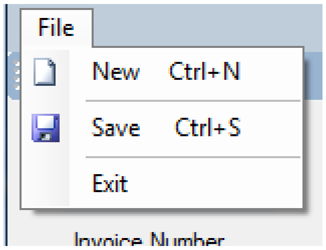
Menu

The menu contains the new, save, and exit options.

New: allows puts the user in add mode and accordingly updates the data in the table.

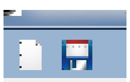
Save: option performs the update, delete, and insert operations that interact with the database.

Exit: prompts the user to save changes and then exits the program.



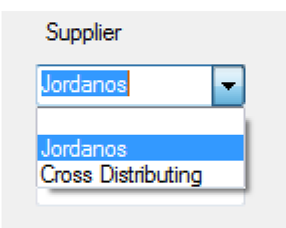
Tool Strip

The tool strip has all of the menu actions except the exit option. Each button has tooltips to inform the user of its purpose.



customCombo

Templated child-class of combobox. It has added functionality for easer integration with oracle data types. The added function sets the valuemember, displaymember, and datasource. Its implementation involves the display of all distinct supplier names.



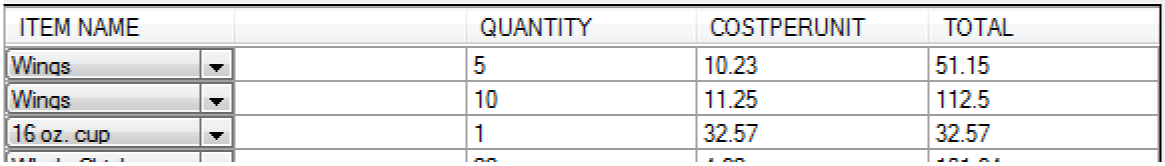
Invoices

This table displays the item name, quantity received, cost per unit, and cost of that item.

Item name: bound to the Inventory table, its value is the InvId and the display member is the item name associated with that ID number.

Quantity/Cost per unit: belong to the Invoice table.

Total: calculated total (Quantity * CostPerUnit).



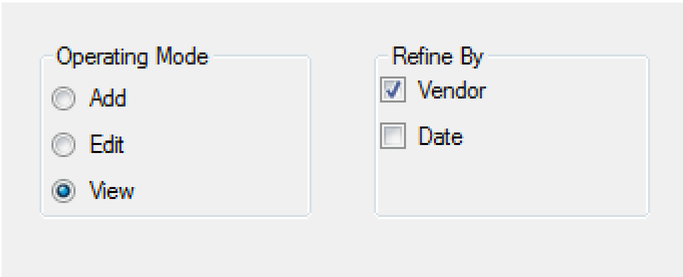
ITEM NAME		QUANTITY	COSTPERUNIT	TOTAL
Wings		5	10.23	51.15
Wings		10	11.25	112.5
16 oz. cup		1	32.57	32.57

Invoices table has three modes:

Add: The table is displayed without any rows in it.

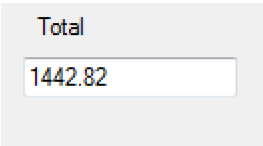
Edit: The table shows entries grouped by supplier and date.

View: By default shows all invoice entries. It can be refined by checking the Vendor or Date check box.

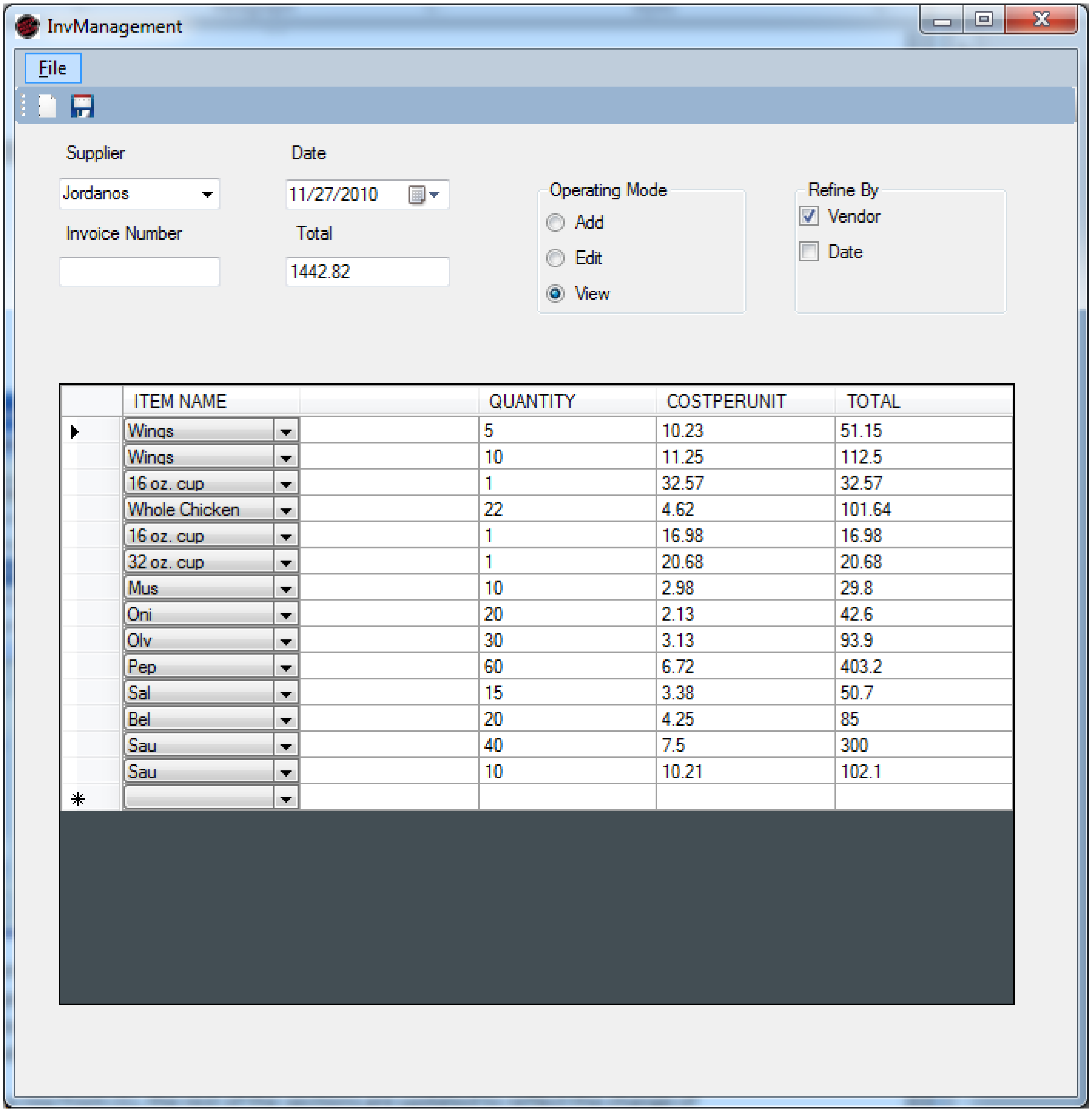


Total

Total: Displays the calculated total of all invoice items currently displayed in the grid view.



Main Form



When the form loads, the table is empty, the date is set to the current date, and all radio buttons, check boxes, combo boxes, and textboxes are empty. From there the user can choose to add, edit, or view invoices.

If the choice is given to add, the add radio button is selected and the table is enabled. From there the vendor, date, invoice number, and items can all be selected. When done, the user saves and the items are inserted into the database.

If the user chooses to edit, the invoice items from the current supplier and date are displayed. Once changed, the corresponding records are displayed and can be edited and saved.

When the user chooses view, the group box Refine By appears with two new check boxes. The data in the table is a list of all invoice entries. The selection can then be refined by selection one or both check boxes and then selecting the appropriate supplier or date. Upon leaving the view state, the check boxes are no longer visible.

Code Description

This project incorporates the use of the Oracle.DataAccess.Client library and as such implements an OracleConnection, OracleCommand, OracleTransaction, and OracleDataAdapter.

OracleConnection

An oracle connection is a class that takes a string formatted string as input and creates a connection with the specified database. Methods used involve open: create an open connection, and close: close the open connection.

OracleCommand

An oracle command represents a stored procedure or statement to execute against a database. The parameters property was used to depict the variable name, data type, size, and location taken from.

OracleTransaction

An oracle transaction represents a transaction to be made in the database. More specifically, it is the ability to perform Rollback, Commit, and Finalize.

OracleDataAdapter

An oracle data adapter is a set of commands coupled with a connection to a database that is used to fill a dataset and update the database.

DataTable

A DataTable was also used in the project. It belongs to the System.Data namespace. It represents a single table of in memory data. More specifically, It holds part of the information that the OracleDataAdapter points to.

DataSet

A DataSet is also in the System.Data namespace and represents an in-memory cache of data.

Classes

RDBcon1

The name stands for Rustys DataBase connection. It is the class that performs database connectivity, loading, updating, inserting, and deletion of data.

It is comprised of three sections:

Database connections: open and closing of connections.

Modification: performs insert, update, and delete procedures.

Get data: returns adapters, tables, and connections.

InvManagement

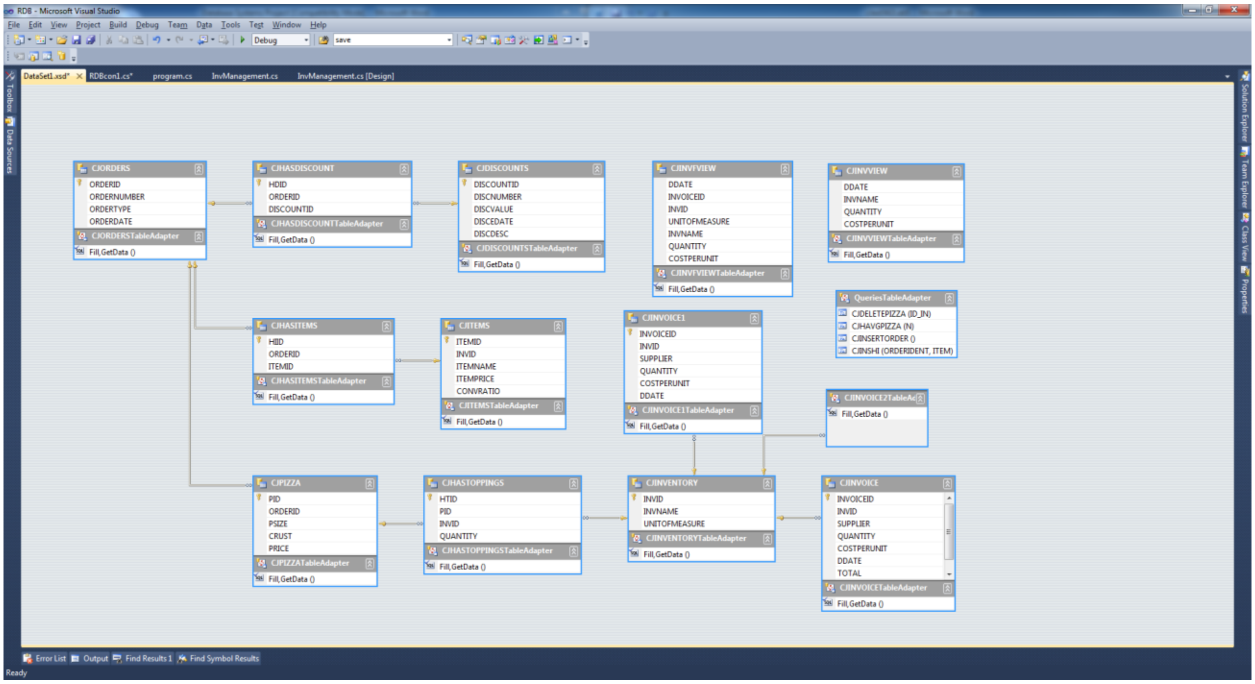
It is the only form of the project. It instantiates all form objects and defines the behavior of those objects. The main functions involved in this class are the filtering of data represented in the grid view, modifying the data in that table, and calling functions defined in the RDBcon1 class.

customCombo

As stated earlier, it is a simple overloaded combobox class with a simplified function to set the datasource, valuemember, and displaymember.

DataSet1

This dataset is a compilation of all tables in the database. The original intent was to incorporate all tables into the project but time consideration limited the selection to only the cjInvoice table and cjInventory table.



Major Features

The major features of this project are the ability to view data that is dependent upon multiple tables concurrently. That data is then able to be inserted, updated and deleted. Then everything is represented in a way that is useful when large amounts of data are present with unnecessary information withheld from the user. The information being displayed is accurate with textual references to numerical fields for ease of use and understanding.

Personal Reflection

The design process has been one full of trial and error. While writing code and getting the desired result naturally come easy to me, trying to get a group of unfamiliar tools to work together is a much more laborious process. Beyond that, once a person has learned which objects are available to be used and how to use them, developing in C# is very easy. The code is very similar to C++ and if you don't remember the proper syntax, Visual Studio tends to give hints. Oracle isn't that hard to manage if the install process is done the right way the first time.

If not, diagnosing the issue can be very time consuming.

Design and Implementation

The first step in designing this application was determining which programming language to use. The primary reasons for developing in C# are the similarity to the C++ language and the ease of designing a GUI using Visual Studio.

The second step was to create a connection between a rudimentary C# application and the oracle database. This step was simple after installing the Oracle client.

After being shown how to integrate database control into the Visual Studio graphical interface, it was time to install oracle database and get things going. This is where the issues started. The next three days were dedicated to uninstalling, installing, doing Google searches, and waiting long periods of time in between each step. In the end, a greater appreciation for the uninstall processes undertaken by most software programs (except oracle database) was achieved. The uninstall is very sloppy in the fact that it does not remove created database folders, files, and registry entries. After many attempts, I was able to install the x64 oracle 11g database with the x86 ODAC client drivers and have full use of the integration with Visual Studio once I copied the tnsnames.ora file to the client folder.

In the final days of my project I began to implement C# code to produce a meaningful front-end application. My process involves:

1) Design a visual representation of the imagined project.
2) Write code until an unfamiliar object or method is needed.
3) Search Google for possible answers.
4) Examine other examples of code that are available.
5) If that doesn't work, try another approach, if it does, repeat.

The end result: a program that works but, as with all programs, more refinement can still be done.