COFFEE SHOP

DATABASE SYSTEMS 342

Drew Dishman
Jacob Owen

Table of Contents

PHASE 5:

1.1 Fact-Finding Techniques:

Employee enters each transaction made into the database. Manager enters employee, inventory, and purchases into the database. The database will generate a monthly income and expense, and inventory reports. In our fact-finding, we contacted various business owners of coffee shops in order to find out their database needs.

1.3 Introduction to Enterprise/Organization:

Coffee shop that sells food/drinks and resupplies inventory from the distributor. We've been in business for a number of years and it is becoming difficult to access necessary data in a timely manner. For these reasons we have decided to develop a database in order to better store our company's information in a more secure and efficient manner.

1.4/ 1.5 Designing a conceptual database to keep records of financial information.

Major Entity Sets:

Supplier: Name, phone number, address.

Inventory: Date/Time, employee who completed the inventory

Inventory information: The ID of every item, total in inventory, amount sold since previous month, amount purchased from previous month, and total amount

unaccounted for.

Item: Name, price per unit, ItemID.

Employee: SSN, name, phone number, address, date hired, pay.

Purchase: TransactionID, Time and Date, Employee's Social, Price and Supplier's name.

Purchase Info: The TransactionID, ItemID and quantity of item bought.

ItemSold: TransactionID, Time and Date, Employee's Social, and price of transaction.

ItemSoldInfo: TransactionID, ItemID and quantity of item bought.

Relationship Sets:

The relationship between the entities Transaction:Supplier is Purchased From. The quantity and price is stored within Purchased From. Transaction:Supplier is a M:M ratio.

The relationship between the entities Transaction:Item is Item Purchased. The quantity and price is stored within Item Purchased. Transaction:Item is a M:M ratio.

The relationship between the entities Sale:Item is Item Sold. The quantity and price is stored within Item Sold. Sale:Item is a M:M ratio.

The relationship between the entities Transaction:Employee is Sold By to distinguish which employee made the sale. Transaction:Employee is a M:M ratio.

The relationship between the entities Employee:Inventory is Completed By to distinguish which employee completed the inventory. Employee:Inventory is a M:M ratio.

1.6 User Groups, Data views and Operations:

The two groups who will use the database are the manager and employee. Manager will have access to every entity and be able to do add, remove, change, and reporting operations. Employees will only have access to the Sale entity and be able to do add operations.

2.1Entity Set Description

1. Transaction

The purpose of the transaction entity is to keep a record of all purchases and sales made by the business. It will hold data for the date and time of the transaction, the employee who made the purchase or sale, the ID number for the transaction and the supplier if the transaction was a purchase. It will also keep track of whether the transaction was a purchase or a sale.

Attribute Name: Transaction ID

Description: Gives a number to each transaction.

Domain/ Type	Value-Range	Default Value	Null Value	Unique	Single or Multiple Value	Simple or Composite
Integer	1 - 10,000	Next Integer	Not Allowed	Yes	Single	Simple

Attribute Name: Date/Time

Description: Gives the date and time of each transaction.

Domain/Type	Value-Range	Default Value	Null Value	Unique	Single or Multiple Value	Simple or Composite

Date/Time	Date/Time	Current Date	Not Allowed	No	Single	Simple

Attribute Name: Purchase/Sale

Description: States whether the transaction was a purchase or a sale, denoted by a 1 for sale and a 2 for purchase

Domain/Type	**Value-Range**	**Default Value**	**Null Value**	**Unique**	**Single or Multiple Value**	**Simple or Composite**
Integer	1 - 2	1	Not Allowed	No	Single	Simple

Candidate Keys	**Primary Key**	**Strong/Weak Entity**	**Fields to be Indexed**
Transaction ID	Transaction ID	Strong	Transaction ID

2. Supplier

The purpose of the supplier entity is to keep a record of the suppliers the store uses

regularly as well as keep track of which supplier was used for each purchase.

Attribute Name: Name

Description: The name of the company the store is buying their products from.

Domain/Type	Value-Range	Default Value	Null Value	Unique	Single or Multiple Value	Simple or Composite
A character string	A - Z, 0 - 9 (Length of 20)	' '	Not Allowed	Yes	Single	Simple

Attribute Name: Number

Description: The phone number of the company the store is buying products from

Domain/Type	Value-Range	Default Value	Null Value	Unique	Single or Multiple Value	Simple or Composite
Character string	0 - 9 (Length of 10)	NULL	Allowed	Yes	Single	Simple

Attribute Name: Address

Description: The address of the company the store is buying products from.

Domain/Type	Value-Range	Default Value	Null Value	Unique	Single or Multiple Value	Simple or Composite
Character string	0 - 9, A - Z (Length 30)	NULL	Allowed	Yes	Single	Simple

Candidate Keys	Primary Key	Strong/Weak Entity	Fields to be Indexed
Name, Number, Address	Name	Strong	Name

3.Item

The item entity will keep track of all of the items the coffee shop will sell as well as the ingredients used to make those items they have in their inventory. It has the Attributes Item name, price per unit and total amount of items in the inventory.

Attribute Name: Name

Description: The name of the item

Domain/Type	Value-Range	Default Value	Null Value	Unique	Single or Multiple Value	Simple or Composite
Character string	0 - 9, A - Z (Length 30)	' '	Not Allowed	Yes	Single	Simple

Attribute Name: Price Per Unit

Description: The total price per unit of the item

Domain/Type	Value-Range	Default Value	Null Value	Unique	Single or Multiple Value	Simple or Composite
Decimal (Monetary Value)	$0.00 - $1,000.00	$0.00	Not Allowed	No	Single	Simple

Attribute Name: Total Units

Description: The total amount of the item in inventory updated every inventory

Domain/Type	Value-Range	Default Value	Null Value	Unique	Single or Multiple Value	Simple or Composite

Double	1 - 1,000	0	Not Allowed	No	Single	Simple

Candidate Keys	Primary Key	Strong/Weak Entity	Fields to be Indexed
Name	Name	Weak	Name

4. Employee

The employee entity is for keeping a record of all employees in the database and their information such as their name, SSN, phone number, address, date hired and pay so that the employer can easily have access to any of this information.

Attribute Name: SSN

Description: The employee's social security number

Domain/Type	Value-Range	Default Value	Null Value	Unique	Single or Multiple Value	Simple or Composite
Character String	0 - 9 (Length of 9)	NULL	Allowed	Yes	Single	Simple

Attribute Name: Fname

Description: The first name of the employee

Domain/Type	Value-Range	Default Value	Null Value	Unique	Single or Multiple Value	Simple or Composite
Character String	A - Z (Length of 15)	' '	Not Allowed	Yes	Single	Simple

Attribute Name: Lname

Description: The last name of the employee

Domain/Type	Value-Range	Default Value	Null Value	Unique	Single or Multiple Value	Simple or Composite
Character String	A - Z (Length of 15)	' '	Not Allowed	Yes	Single	Simple

Attribute Name: Phone Number

Description: The phone number of the employee

Domain/Type	Value-Range	Default Value	Null Value	Unique	Single or Multiple Value	Simple or Composite
Character String	0 - 9 (Length of 10)	NULL	Allowed	Yes	Single	Simple

Attribute Name: Address

Description: The address of the employee

Domain/Type	Value-Range	Default Value	Null Value	Unique	Single or Multiple Value	Simple or Composite
Character String	0 - 9, A - Z (Length of 30)	NULL	Allowed	Yes	Single	Simple

Attribute Name: Date Hired

Description: The date the employee was hired

Domain/Type	Value-Range	Default Value	Null Value	Unique	Single or Multiple Value	Simple or Composite
Date	Time and Date	Current Date	Not Allowed	No	Single	Simple

Attribute Name: Pay

Description: The employee's pay

Domain/Type	Value-Range	Default Value	Null Value	Unique	Single or Multiple Value	Simple or Composite
Decimal (Monetary Value)	$8.00 - $20.00	$8.00	Not Allowed	No	Single	Simple

Candidate Keys	Primary Key	Strong/Weak Entity	Fields to be Indexed
Fname, Lname, SSN	SSN	Strong	SSN

5. Inventory

The inventory entity's purpose is to provide the company with a way to make an inventory every month/week etc. They can then use the data to compare with previous inventories to find out how much loss they experienced in that time period. It will accomplish this by having a field for the date as well as a field for all of the items.

Attribute Name: Date

Description: The date the inventory was completed

Domain/Type	Value-Range	Default Value	Null Value	Unique	Single or Multiple Value	Simple or Composite
Date	Time and Date	Current Date	Not Allowed	Yes	Single	Simple

Attribute Name: Items

Description: The total units and names for all items in inventory

Domain/Type	Value-Range	Default Value	Null Value	Unique	Single or Multiple Value	Simple or Composite
Character String, Integer	A - Z, 0 - 10,000	' ' 0	Not Allowed	No	Multiple	Composite

Candidate Keys	Primary Key	Strong/Weak Entity	Fields to be Indexed
Date	Date	Strong	Date

3.Item

The item entity will keep track of all of the items the coffee shop will sell as well as the ingredients used to make those items they have in their inventory. It has the Attributes Item name, price per unit and itemID.

Attribute Name: Name

Description: The name of the item

Domain/Type	Value-Range	Default Value	Null Value	Unique	Single or Multiple Value	Simple or Composite

Character string	0 - 9, A - Z (Length 30)	' '	Not Allowed	Yes	Single	Simple

Attribute Name: Price Per Unit

Description: The total price per unit of the item

Domain/Type	**Value-Range**	**Default Value**	**Null Value**	**Unique**	**Single or Multiple Value**	**Simple or Composite**
Decimal (Monetary Value)	$0.00 - $1,000.00	$0.00	Not Allowed	No	Single	Simple

Attribute Name: ItemID

Description: The ID number of every item we have inventoried.

Domain/Type	Value-Range	Default Value	Null Value	Unique	Single or Multiple Value	Simple or Composite
Number(9)	A nine digit number	0	Not Allowed	Yes	Single	Simple

Candidate Keys	Primary Key	Strong/Weak Entity	Fields to be Indexed
Name	Name	Weak	Name

4. Employee

The employee entity is for keeping a record of all employees in the database and their information such as their name, SSN, phone number, address, date hired and pay so that the employer can easily have access to any of this information.

Attribute Name: SSN

Description: The employee's social security number

Domain/Type	Value-Range	Default Value	Null Value	Unique	Single or Multiple Value	Simple or Composite
Number(9)	0 - 9 (Length of 9)	000000000	Not Allowed	Yes	Single	Simple

Attribute Name: Fname

Description: The first name of the employee

Domain/Type	Value-Range	Default Value	Null Value	Unique	Single or Multiple Value	Simple or Composite
Character String	A - Z (Length of 15)	' '	Not Allowed	Yes	Single	Simple

Attribute Name: Lname

Description: The last name of the employee

Domain/Type	Value-Range	Default Value	Null Value	Unique	Single or Multiple Value	Simple or Composite
Character String	A - Z (Length of 15)	' '	Not Allowed	Yes	Single	Simple

Attribute Name: Phone Number

Description: The phone number of the employee

Domain/Type	Value-Range	Default Value	Null Value	Unique	Single or Multiple Value	Simple or Composite
Character String	0 - 9 (Length of 10)	NULL	Allowed	Yes	Single	Simple

Attribute Name: Address

Description: The address of the employee

Domain/Type	Value-Range	Default Value	Null Value	Unique	Single or Multiple Value	Simple or Composite
Character String	0 - 9, A - Z (Length of 30)	NULL	Allowed	Yes	Single	Simple

Attribute Name: Date Hired

Description: The date the employee was hired

Domain/Type	Value-Range	Default Value	Null Value	Unique	Single or Multiple Value	Simple or Composite
Date	Time and Date	Current Date	Not Allowed	No	Single	Simple

Attribute Name: Pay

Description: The employee's pay

Domain/Type	Value-Range	Default Value	Null Value	Unique	Single or Multiple Value	Simple or Composite
Decimal (Monetary Value)	$8.00 - $20.00	$8.00	Not Allowed	No	Single	Simple

Candidate Keys	Primary Key	Strong/Weak Entity	Fields to be Indexed
Fname, Lname, SSN	SSN	Strong	SSN

5. Inventory

The inventory entity's purpose is to provide the company with a way to make an inventory every month. They can then use the data to compare with previous inventories to find out how much loss they experienced in that time period. It will accomplish this by having a field for the date as well as a field for all of the items.

Attribute Name: Date

Description: The date the inventory was completed

Domain/Type	Value-Range	Default Value	Null Value	Unique	Single or Multiple Value	Simple or Composite
Date	Time and Date	Current Date	Not Allowed	Yes	Single	Simple

Attribute Name: ItemIDs

Description: The total units and ID numbers for all items in inventory

Domain/Type	Value-Range	Default Value	Null Value	Unique	Single or Multiple Value	Simple or Composite
Character String, Integer	A - Z, 0 - 10,000	' ' 0	Not Allowed	No	Multiple	Composite

Attribute Name: Employee SSN

Domain/Type	Value-Range	Default Value	Null Value	Unique	Single or Multiple Value	Simple or Composite
Number(9)	9 Digit Number	Null	Allowed	Yes	Single	Simple

Attribute Name: Total Number in inventory

Domain/Type	Value-Range	Default Value	Null Value	Unique	Single or Multiple Value	Simple or Composite
Number	0 - 10000	0	Not Allowed	No	Multiple	Simple

Attribute Name: Purchased Since Last

Domain/Type	Value-Range	Default Value	Null Value	Unique	Single or Multiple Value	Simple or Composite
Number	0 - 10000	0	Not Allowed	No	Multiple	Simple

Attribute Name: Sold Since Last

Domain/Type	Value-Range	Default Value	Null Value	Unique	Single or Multiple Value	Simple or Composite
Number	0 - 10000	0	Not Allowed	No	Multiple	Simple

Attribute Name: Amount unaccounted for

Domain/Type	Value-Range	Default Value	Null Value	Unique	Single or Multiple Value	Simple or Composite
Number	0 - 10000	0	Not Allowed	No	Multiple	Simple

Candidate Keys	Primary Key	Strong/Weak Entity	Fields to be Indexed
Date	Date	Strong	Date

2.2/2.3 Relationship Set Description

RESUPPLY:

- Relationship between the entities Transaction & Supplier which stores the quantity and price of the supplies we purchased from.

Entity Sets Involved	Mapping Cardinality	Descriptive Field	Participation Constraint
Transaction & Supplier	M:M	Quantity & Price	Total

ITEM BOUGHT:

- Relationship between the entities Transaction & Item which stores the quantity and price of the items we purchased.

Entity Sets Involved	Mapping Cardinality	Descriptive Field	Participation Constraint
Transaction & Item	M:M	Quantity & Price	Total

ITEM SOLD:

- Relationship between the entities Transaction & Item which stores the quantity and price of the items we sold.

Entity Sets Involved	Mapping Cardinality	Descriptive Field	Participation Constraint
Transaction & Item	M:M	Quantity & Price	Total

SELLS:

- Relationship between the entities Transaction & Employee to keep track of which employee made the sale.

Entity Sets Involved	Mapping Cardinality	Descriptive Field	Participation Constraint
Transaction & Employee	M:M	Employee	Partial

COMPLETES:

- Relationship between the entities Employee & Inventory to keep track of which employee completed inventory checks.

Entity Sets Involved	Mapping Cardinality	Descriptive Field	Participation Constraint
Employee & Inventory	M:M	?	Partial

PHASE 2:

(3.1) E-R Model and Relational Model

Conceptual modeling is a very important phase in designing a successful database application. Generally, the term database application refers to a particular database and the associated programs that implement the database queries and updates. These programs provide user-friendly graphical user interfaces (GUIs) utilizing forms and menus for the end users of the application. A major part of the database application will require the design, implementation, and testing of these application programs.

The Entity-Relationship (ER) model is a popular high-level conceptual data model. This model is frequently used for the conceptual design of database applications. Also used to create a ER-Diagram of your conceptual database using various entity types, sets, keys, and attributes.

The relational data model was first introduced by Ted Codd of IBM research in 1970, and it attracted immediate attention due to its simplicity and mathematical foundation. The model uses the concept of a mathematical relation -- which looks somewhat like a table of values -- as its basic building block, and has its theoretical basis in set theory and first-order predicate logic. SQL query language is the standard for commercial relational DBMSs. Relational Algebra and relational calculus are two formal languages associated with the relational model. The relational calculus is considered to be the basis for the SQL language, and the relational algebra is used in the internals of many database implementations for query processing and optimization.

ER MODEL	RELATIONAL MODEL
Entity type	Entity Relation
1:1 or 1:N relationship type	Foreign key (or relationship relation)
M:N relationship type	Relationship relation and two foreign keys
n-ary relationship type	Relationship relation and n foreign keys
Simple Attribute	Attribute
Composite Attribute	Set of simple component attributes
Multivalued Attribute	Relation and foreign key

Value Set	Domain
Key Attribute	Primary (or secondary) key

In order to convert an Entity-Relationship model into a Relational model you must follow this 7 step algorithm:

Step 1: Mapping of Regular Entity Types
Step 2: Mapping of Weak Entity Types
Step 3: Mapping of Binary 1:1 Relationship Types
Step 4: Mapping of Binary 1:N Relationship Types
Step 5: Mapping of Binary M:N Relationship Types
Step 6: Mapping of Multivalued Attributes
Step 7: Mapping of N-ary Relationship Types

Conversion Issues:

For each strong entity type E in the ER schema, create a relation R that includes all the simple attributes of E. Include only the simple component attributes of a composite attribute. Choose one of the key attributes of E as the primary key for R. if the chosen key of E is a composite, then the set of simple attributes that form it together will form the primary key of R.

For each weak entity type W in the ER schema with owner entity type E, create a relation R and include all simple attributes of W as attributes of R. In addition, include as foreign key attributes of R, the primary key attribute(s) of the relation(s) that correspond to the owner entity type(s); this takes care of mapping the indentifying relationship type of W. The primary key of R is the combination of the primary key(s) of the owner(s) and the partial key of the weak entity type W, if any.

Constraints:

A referential constraint is usually used in the context of relationships and asserts that exactly one value exists in a given context. If R is a (M:1 or 1:1) relationship from E to F, we require that the entity in F related by R to an entity in E must exist.

A entity constraint is a constraint placed on an entity in how it relates to another entity.

A primary key constraint is a rule that says that the primary key fields cannot be null and cannot contain duplicate data.

A foreign key constraint specifies that the data in a foreign key must match the data in the primary key of the linked table.

3.3

EMPLOYEE					
SSN	**NAME**	**PHONE #**	**ADDRESS**	**DATE HIRED**	**PAY**
100000000	Billy Bob	234-9548	212 Maple Avenue	05/21/87	$17.00 an hour
110000000	John Jay	982-2312	7901 Revelstoke Way	04/02/91	$12.00 an hour
111000000	Dish Man	723-0932	400 Woodrow	08/27/11	$8.50 an hour
111100000	Kyle Hamster	120-2345	895 Cliffspring Drive	03/14/09	$8.50 an hour
111110000	Luke Paper	687-9123	3242 Evergreen Terrace	07/06/98	$12.00 an hour
111111000	Trashcan James	432-6507	2301 Alley St	02/17/04	$10.00 an hour
111111100	Clifford Red	586-1946	3485 Mountain Vista Rd	12/25/06	$8.50 an hour
111111110	Scuba Steve	834-1968	109 Seapines	11/22/01	$10.00 an hour
111111111	Teapot Sally	518-5205	921 Mothergoose Ave	01/30/10	$8.50 an hour
200000000	Negative Nancy	407-5248	1239 Hate Drive	06/06/06	$8.50 an hour

TRANSACTION			
ID	**DATE/TIME**	**PURCHASE/SALE**	**ESSN**
1000	10/23/11 at 8:01 A.M.	SALE	100000000
1001	10/23/11 at 8:15 A.M.	SALE	110000000
1002	10/23/11 at 8:31 A.M.	SALE	111000000
1003	10/23/11 at 8:35 A.M.	SALE	111100000
1004	10/23/11 at 8:43 A.M.	SALE	111110000
1005	10/23/11 at 9:05 A.M.	SALE	111111000
1006	10/23/11 at 9:10 A.M.	SALE	111111100
1007	10/23/11 at 9:13 A.M.	SALE	111111110
1008	10/23/11 at 9:27 A.M.	SALE	1111111111
1009	10/23/11 at 9:38 A.M.	SALE	200000000

SUPPLIER		
NAME	**PHONE #**	**ADDRESS**
Grounds Distribution	102-2394	1234 17th St.
The Black Seeds	583-4785	0981 Trident Drive
Johnson's & Co	348-9450	231 Enchilada Way
Java Express	709-3240	7129 Coca Cola St.
Cool Beans Delivery	234-8071	182 Clydesdale
Hot Transportation	625-2349	094 Beach St.
Make Believe Utilities	041-4389	1283 Singapore
The Grid	216-3987	3274 Awesome St.
Interior Crocodile Alligator	853-1092	915 Africa
Chevrolet Movie Theater	123-0987	1800 Edwards

ITEM

NAME	ITEM ID	PRICE PER UNIT
Vanilla Frappuccino	100001	$3.50
Pastry	100002	$2.00
Latte	100003	$3.00
Cappuccino	100004	$3.00
Decaf	100005	$2.50
Iced Coffee	100006	$3.00
Cafe Mocha	100007	$3.25
Blueberry Muffin	100008	$3.00
Macchiato	100009	$4.00
Irish Coffee	100010	$4.00

ITEM SOLD			
TRANSACTION ID	**QUANTITY**	**PRICE**	**ITEM ID**
1000	1	$3.50	100001
1001	1	$2.00	100002
1002	1	$4.00	100009
1003	2	$4.00	100002
1004	3	$9.00	100008
1005	1	$3.00	100003
1006	2	$5.00	100005
1007	1	$4.00	100010
1008	1	$3.00	100004
1009	2	$6.50	100007

INVENTORY

ItemId	Total	PurchasedSinceLast	SoldSinceLast	Date/Period	Amount Unnaccounted For	ESSN
100000	100	30	70	4/01/2011 - 4/30/2011	0	100000000
100002	15	10	4		1	
100003	234	100	96		32	
100004	350	200	100		50	
100005	12	5	6		1	
100006	13	3	7		2	
100007	56	14	17		12	
100008	24	20	24		3	
100009	32	30	31		0	

100010	495	495	495		0	
100000	101	31	71	5/01/2011 - 5/31/2011	1	100000001
100002	17	99	5		2	
100003	233	100	93		12	
100004	345	255	102		45	
100005	13	4	5		2	
100006	9	7	12		12	
100007	55	23	18		4	
100008	32	21	19		5	
100009	33	35	28		1	
100010	485	476	400		0	
100000	96	32	64	6/01/2011 - 6/30/2011	2	100000002
100002	88	96	13		1	
100003	224	103	91		22	
100004	346	216	140		30	
100005	9	3	8		3	
100006	10	7	9		1	
100007	53	14	15		9	
100008	35	31	22		2	
100009	32	17	36		1	
100010	484	488	463		0	
100000	101	31	70	7/01/2011 - 7/31/2011	0	100000005
100002	17	99	4		1	
100003	233	100	96		32	
100004	345	255	100		50	
100005	13	4	6		1	
100006	9	7	7		2	
100007	55	23	17		12	
100008	32	21	24		3	
100009	33	35	31		0	
100010	485	476	495		0	
100000	115	32	65	8/01/2011 -	3	100000001

100002	108	103	4	8/31/2011	1	
100003	223	100	92		30	
100004	335	235	100		15	
100005	14	7	3		2	
100006	10	4	22		3	
100007	33	26	36		6	
100008	34	22	22		0	
100009	26	36	36		2	
100010	478	476	430		1	
100000	101	31	70	9/01/2011 - 9/30/2011	0	100000000
100002	17	99	4		1	
100003	233	100	96		32	
100004	345	255	100		50	
100005	13	4	6		1	
100006	9	7	7		2	
100007	55	23	17		12	
100008	32	21	24		3	
100009	33	35	31		0	
100010	485	476	495		0	

PURCHASE

Quantity	TID	Price	Sname	IID
100	1001	$2,570.36	Coffee Suppliers	10000000001
23				10000000003
32				10000000005
400				10000000007
360				10000000009
123	1002	$3,624.35	Coffee R Us	10000000002
395				10000000004
66				10000000006
54				10000000008
32				100000000010

71	1003	$3,732.77	Coffee Suppliers	10000000001
12				10000000002
32				10000000003
15				10000000005
64				10000000007
200	1004	$2,934.55	Coffee Suppliers	10000000008
25				10000000005
63				10000000004
22				10000000006
134				10000000002
78	1005	$2,876.32	Coffee Max	10000000001
493				10000000010
62				10000000002
300				10000000009
124				10000000003
123	1006	$3,151.60	Coffee R Us	10000000004
14				10000000003
36				10000000006
51				10000000009
12				10000000002
9	1007	$3,333.33	Coffee Suppliers	10000000002
101				10000000003
105				10000000004
64				10000000005
28				10000000006
37	1008	$2,432.23	Coffee R Us	10000000001
462				10000000010
398				10000000003
17				10000000007
77				10000000005
8	1009	$2,323.23	Coffee Suppliers	10000000004
72				10000000001

209				10000000006
370				10000000010
64				10000000005
223	1010	$4,264.78	Coffee Max	10000000010
47				10000000003
17				10000000002
23				10000000004
495				10000000008

(3.4/3.5) QUERIES

1.The most expensive purchase from a supplier.

Relational Algebra:

PROJECT (Purchase(p) - (Purchase(p1) **JOIN** Purchase(p2)))
p.* (p1.Sname = p2.Sname ^ p1.price < p2.price)

Tuple Calculus:

{ p.* | (Ep)purchase(p) ^ **NOT**((Ep2)purchase(p2) ^ (p2 > p1)}

Domain Calculus:

{<q,t,p,s,i> | purchase(q,t,p,s,i) ^ (Ep)(purchase(_,t,p,_,_) ^ **NOT** (Ep2)(purchase<_,t,p,_,_) ^ (p2 > p1)

2.All Employees who made no sales last week.

Relational Algebra:

PROJECT (Employee(e) - **PROJECT**(Employee(e1) **JOIN** Transaction))
e.* e1.* (SSN = ESSN ^ date > 10/7/2011 ^
date < 10/14/2011)

Tuple Calculus:

Domain Calculus:

{<s,n,ph,a,d,p> | employee(s,n,ph,a,d,p) ^ **NOT**((Es)employee(s,_,_,_,_,_) ^ (Ee)transaction(_,d,_,e)
^ s = e ^ (d > 8/31/1011 ^ d < 10/1/2011))}

{e.* | (Ee)employee(e) ^ **NOT**((Et)transaction(t) ^ e.SSN = t.ESSN ^ t.date > 8/31/2011 ^
t.date < 10/01/2011)}

3. Suppliers we have bought every item from at least once.

Relational Algebra:

PROJECT (Purchase) / **PROJECT** (Item)
Sname, ItemID ItemID

Tuple Calculus:

{s.* | (Es)supplier(s) ^ **NOT**((Ep)purchase(p) ^ (Ei)item(i) ^ i.ItemID != p.IID)}

Domain Calculus:

{<n,p,a> | supplier(n,p,a) ^ **NOT**((Ei)purchase(_,_,_,s,i) ^ (Ei2)item(i2,_,_,_,_,_,_) ^ i != i2)}

4. Most popular item in inventory last month.

Relational Algebra:

PROJECT (Inventory(in) - (Item **NATURAL JOIN** (Inventory(I1) **JOIN** Inventory(I2)))
Item.* (I1.date > 8/31/2011 ^
I1.date < 10/1/2011 ^
I2.date > 8/31/2011 ^
I2.date < 10/1/2011 ^
I1.soldsincelast < I2.soldsincelast)

Tuple Calculus:
{i.* | item(i) ^ inventory(y) ^ y.date >= 10/1/2011 ^ y.date <= 10/31/2011 ^ **NOT**(inventory(y2) ^
y2.sold > y.sold ^ (y2.date >= 10/1/2011 ^ y2.date <= 10/31/2011))}

Domain Calculus:

{<n,i,p> | item(n,i,p) ^ ((Es)inventory(i,_,_,s,d,_,_) ^ d >= 10/01/2011 ^ d <= 10/31/2011) ^ **NOT**
((Es2)inventory(i,_,_,s2,d2,_,_) ^ s2 > s ^ (d2 >= 10/01/2011 ^ d2 <= 10/31/2011))}

5. Smallest number of items purchased in one transaction.

Relational Algebra:

PROJECT (Purchase(p) - (Purchase(p1) **JOIN** Purchase(p2)))

p.quantity p1.quantity > p2.quality

Tuple Calculus:

{p.quantity | (Ep)purchase(p) ^ (Ep2)purchase(p2) ^ **NOT**((Ep3)purchase(p3) ^
p3.quantity < p2.quantity)}

Domain Calculus:

{<q> | purchase(q,t,p,s,i) ^ (Eq)(purchase(q,t,_,_,_)) ^ **NOT** ((Eq2)(purchase(q,_,_,_,_)) ^ q2 > q1)}

6. All employees who make exactly ten dollars an hour.

Relational Algebra:

PROJECT (Employee) / $10.00
name, pay

Tuple Calculus:

{e.* | (Ee)employee(e) ^ e.pay = $10.00}

Domain Calculus:

{<s,n,ph,a,d,p> | employee(s,n,ph,a,d,p) ^ ((Ep)employee(_,_,_,_,_,p) ^ p = $10.00)}

7. The second largest amount of money made in one sale

Relational Algebra:

AllButLargest <= (Sale(s1) **JOIN** Sale(s2)))
s1.price < s2.price

PROJECT (AllButLargest(a) - (AllButLargest(a1) **JOIN** AllButLargest(a2)))
a.price a1.price < a2.price

Tuple Calculus:

{s2.price | (Es)sale(s) ^ (Es2)sale(s2) ^ s2.price < s.price ^ **NOT**((Es3)sale(s3) ^ (s3.price < s.price)
^ (s3.price != s2.price))}

Domain Calculus:

{<p,t> | sale(i,q,p,t) ^ ((Ep)sale(_,_,p,_) ^ (Ep2sale(_,_,p2,t) ^ p2 < p^ **NOT**(Ep3)sale(_,_,p3,_)

^ (p3 < p) ^ (p3 != p2))}

8. The average price per sale last month.

Relational Algebra:

PROJECT (**AGGREGATE** (**SELECT** (Transaction **X** Sale)))
price AVERAGE Price date > 8/31/2011 ^ date < 10/01/2011

Tuple Calculus:

{ **AGGREGATE** Aggregate s.price | (Es)sale(s) ^ (Et)transaction(t) ^ t.date > 8/31/2011 ^
t.date < 10/01/2011 ^ t.ID = s.TID

Domain Calculus:

{<**AGGREGATE** Average p> | sale(_,_,p,t) ^ ((Ed)transaction(i,d,_,_) d > 8/31/2011 ^
d < 10/01/2011 ^ t = i)}

PHASE 3:

1. Normalization of Relations

(1)

A.

The First Normal Form (1NF) was historically defined to disallow multivalued attributes, composite attributes, and their combinations. It states that the domain of an attribute must include only atomic (simple, indivisible) values and that the value of any attribute in a tuple must be a single value from the domain of that attribute. In other words, 1NF disallows relations within relations or relations as attribute values within tuples.

The Second Normal Form (2NF) is based on the concept of full functional dependency. A functional dependency X -> Y is a full functional dependency if removal of any attribute A from X means that the dependency does not hold any more. A functional dependency X -> is a partial dependency if some attribute A E X can be removed from X and the dependency still holds. **Definition:** A relation schema R is in 2NF if every nonprime attribute A in R is fully functionally dependent on the primary key of R.

The Third Normal Form (3NF) is based on the concept of transitive dependency. A functional dependency X -> Y in a relation schema R is a transitive dependency if there exists a set of attributes Z in R that is neither a candidate key nor a subset of any key of R, and both X -> Z and Z -> Y hold.

Defintion: According to Codd's original definition, a relation schema R is in 3NF if it satisfies 2NF and no nonprime attribute of R is transitively dependent on the primary key.

Boyce-Codd normal form (BCNF) was proposed as a simpler form of 3NF, but it was found to be stricter than 3NF. That is, every relation in BCNF is also in 3 NF; however, a relation in 3NF is not necessarily in BCNF. **Definition**: A relation schema R is in BCNF if whenever a nontrivial functional dependency X _> holds in R, then X is a superkey of R.

B.

Storing natural joins of base relations leads to an additional problem referred to as update anomalies. these can be classified into insertion anomalies, deletion anomalies, and modification anomalies.

Insertion Anomalies: Place Null values where the primary key is not allowed.

Deletion Anomalies: If you delete a row from a table that information is lost from the Database.

Modification Anomalies: If you change an attribute in one row you must update that whole row or the information will become inconsistent.

(2)

A. Which normal forms are the relations in?

Employee:	First normal form (1NF)
Supplier:	First normal form (1NF)
Item:	2nd normal form (2NF)
Inventory:	2nd normal form (2NF)
Inventory Info:	2nd normal form (2NF)
Item Sold:	2nd normal form (2NF)
Item Sold Info:	2nd normal form (2NF)
Purchase:	2nd normal form (2NF)
Purchase Info:	2nd normal form (2NF)

B. Do any modification anomalies exist?

Yes, there will be Deletion/Modification anomalies when the Item Id attribute from the relation Item becomes deleted. Thus losing the info in the relations ItemSold Info and Purchase Info. Delete the ESSN from employee and Inventory/Item Sold/Purchase results in a loss of info. If the Date/Time is deleted in Inventory then that info will be lost in Inventory Info. Same goes for the attribute TID.

4.

(1)

SQL * Plus: an Oracle command-line utility program that can run SQL and PL/SQL commands interactively or from a script. SQL*Plus operates as a relatively simple tool with a basic command-line interface. Programmers and DBAs commonly use it as the default available fundamental interface in almost any Oracle Software installation.

(2)

A **schema** is a collection of logical structures of data, or schema objects. A schema is owned by a database user and has the same name as that user. Each user owns a single schema. Schema objects can be created and manipulated with SQL and include the following types of objects: Tables, Views, Indexes, and Clusters. **Tables** are the basic unit of data storage in an Oracle database. Data is stored in **rows** and **columns**. A view is a tailored presentation of the data contained in one or more tables or other views. A view takes the output of a query and treats it as a table. Therefore, a view can be thought of as a stored query or a virtual table. You can use views in most places where a table can be used.

PHASE 4:

A. Common Features in Oracle PL/SQL and MS Trans-SQL

(1) Components which consist of PL/SQL and Trans-SQL:

A Oracle PL/SQL is a block structured language in which functions, procedures and anonymous blocks are the basic blocks. Blocks can be defined with another block.

A block consists of three parts:

- **Declaration:** declare variables, constraints, cursors, and user-defined expressions.
- **Executable:** consists of SQL/SQLPLUS statements.
- **Exception Handling:** A predefined or user-defined warning or error handled by the PL/SQL program.

(2) Purposes of Stored subprograms:

A **stored procedure** is a subroutine available to applications that access a relational database system. A stored procedure is stored by the DBMS in the server. They are typically used to process data validation or access control mechanisms. Also, stored procedures can consolidate and centralize logic that was originally implemented in applications. Furthermore, stored procedures can receive variables, return results or modify variables and return them, depending on how and where the variable is declared.

(3) Benefits of calling stored subprogram over sending a dynamic SQL to front- end DBMS server:

- If a database program is needed by several applications, it can be stored at the server and invoked by any of the application programs. This reduces duplication of effort and improves software modularity.
- Executing a program at the server can reduce data transfer and communication cost between the client and server in certain situations.
- These procedures can enhance the modeling power provided by views by allowing more complex types of derived data to be made available to the database users. Additionally, they can be used to check for complex constraints that are beyond the specification power of assertions and triggers.

When comparing it with dynamic SQL a stored program will remove overhead, avoid network traffic, protection against SQL injection attacks, delegation of access-rights, and encapsulation of business logic.

B. Oracle PL/SQL

(1) PL/SQL program structure, control statements, cursors:

- **PL/SQL program structure:**

  A program structure is a block that consists of three parts:

  1. **Declaration:** declare variables, constraints, cursors, and use-defined expressions.
  2. **Executable:** consists of SQL/SQLPLUS statements.
  3. **Exception Handling:** A predefined or user-defined warning or error handled by the PL/SQL program.

- **Control statement:** Consists of conditional, iterative, and sequential controls.

  - **Conditional Controls:**

```
If condition THEN
        Statements;
END IF
IF condition THEN
        Statements;
ELSEIF condition THEN
        Statements;
ELSE
        Statements;
END IF;
EXIT-WHEN condition;
```

- **Iterative Controls:**

LOOP
Statements;
END LOOP
FOR I IN lowerbound .. upperbound LOOP
Statements;
END LOOP;
For cursor_variable IN cursor_name LOOP
Statements;
END LOOP;
WHILE condition LOOP
Statements;
END LOOP;

- **Sequential Control:**

GOTO label;
.
<<label>>

- **Cursors:**

Cursors are used by database programmers to process individual rows returned by database system queries. Cursors enable manipulation of whole result sets at once—a capability that most procedural programming languages lack. In this scenario, a cursor enables the rows in a result-set to be processed sequentially.

Syntax:

DECLARE

CURSOR cursor_name [(parameter_name TYPE [, parameter_name TYPE])]
IS select_statement;

(2) What is a stored procedure and syntax of creating a stored procedure:

A stored procedure is a saved section of code which handles a specific task which must be repeated regularly. It is similar to a function in a procedural language such as C. The syntax for creating a stored procedure is:

CREATE [OR REPLACE] PROCEDURE procedure_name ([parameters])

IS

[delclare any variables]

BEGIN

[enter procedural code]

EXCEPTION

[handle any exceptions]

END;

(3) What is a stored function and Syntax of creating a stored function:

A stored function is similar to a stored procedure except that it returns a value where a procedure does not. The syntax for creating a function is:

CREATE [OR REPLACE] FUNCTION function_name ([parameters])

RETURN

[specify datatype to return]

IS

[delclare any variables]

BEGIN

[enter procedural code]

EXCEPTION

[handle any exceptions]

END;

(4) What is a package and Syntax of creating a package:

A package in PL/SQL is a schema object which contains many different types, procedures, functions etc which can then be shared by many users who may need them for various different applications within the same company. The syntax for a package is:

CREATE [OR REPLACE] PACKAGE package_name

[AUTHID {CURRENT_USER | DEFINER}]

{IS | AS}

[PRAGMA SERIALLY_REUSABLE;]

[collection_type_definition]

[record_type_definition]

[subtype_definition]

[collection_declaration]

[constant_declaration]

[exception_declaration]

[object_declaration]

[record_declaration]

[variable_declaration]

[cursor_spec]

[function_spec]

[procedure_spec]

[call_spec]

[PRAGMA RESTRICT_REFERENCES(assertions) …]

END [package_name];

(5) What is a trigger and Syntax of creating a trigger:

A trigger in PL/SQL is a section of code that is excecuted when a statement such as delete or update is inserted. It is automatically done when the statement is excecuted. The syntax for a trigger is:

CREATE [OR REPLACE] TRIGGER trigger_name

{BEFORE | AFTER | INSTEAD OF }

{INSERT [OR] | UPDATE [OR] | DELETE}

[OF col_name]

ON table_name

[REFERENCING OLD AS o NEW AS n]

[FOR EACH ROW]

WHEN (condition)

BEGIN

--- sql statements

END;

C. Code and Documentation

Procedure for inserting an employee:

```
create or replace procedure ddjo_addemployee( p_ssn in number, p_name in varchar2,
        p_phone in number, p_address in varchar2,p_hdate in date, p_pay in number) as
        begin
       insert into ddjo_employee values(p_ssn, p_name, p_phone, p_address, p_hdate, p_pay);

       exception
       when others then
         raise_application_error( -40001, 'An error occurred in ' || sqlcode ||
                    '-ERROR-' || sqlerrm );
end ddjo_addemployee;
/
```

Procedure for deleting a supplier:

```
create or replace procedure ddjo_deletesupplier (supname in varchar2) is
   begin
      delete from ddjo_supplier where name = supname;

      exception
      when others then
         raise_application_error( -40001, 'An error occurred in ' || sqlcode
                        || '-ERROR-' || sqlerrm );

   end ddjo_deletesupplier;
/
```

Function to return average purchase price:

```
create or replace function ddjo_avgPrice (n in number) return number is

   s number(9, 2) := 0.0;
   p number(7, 2);

   cursor c1 is select price from ddjo_purchase order by price desc;

begin
```

```
    open c1;
    for i in 1 .. n loop

        fetch c1 into p;
        s := s + p;

    end loop;

    close c1;

    return s / n;

exception

when others then
    raise_application_error( -40001, 'An error occured in ' || sqlcode ||
                        '-ERROR-' || sqlerrm);
end;
/
```

Trigger to be fired on deletion of employee:

```
create or replace trigger ddjo_deleteEmployee

before update or delete on ddjo_employee

for each row

begin

   insert into ddjo_logtable

   values (to_char(:old.ssn) || ' ' || to_char(:old.name) || ' ' ||

      to_char(:old.phone#) || ' ' || to_char(:old.address) || ' ' ||

      to_char(:old.date_hired) || ' ' || to_char(:old.pay)

      ,to_char(:new.ssn) || ' ' || to_char(:new.name) || ' ' ||

      to_char(:new.phone#) || ' ' || to_char(:new.address) || ' ' ||

      to_char(:new.date_hired) || ' ' || to_char(:new.pay) );

  exception

      when others then

         raise_application_error( -40001, 'An error occurred in ' || sqlcode

                        || '-ERROR-' || sqlerrm );
```

end ddjo_deleteEmployee;

/

PHASE 5:

A. Description of Daily activities of the user group.

In our DBMS, the manager will be the only user group of the Coffee Shop. The manager will be able to perform the following functionalities: add, delete, or update tables/reports from any of the following tabs: employee, item, supplier, inventory, purchase, and sales.

B. Relations, views and subprograms related to the activities:

```
CREATE TABLE ddjo_employee(
    SSN NUMBER(9) not null,
    NAME VARCHAR2(30) not null,
    phone# varchar2(10) not null,
    address varchar(30) not null,
    date_hired date not null,
    pay number not null,
    CONSTRAINT pk_employee PRIMARY KEY(SSN)
    )
    PCTfree 5
    PCTUSED 15
    TABLESPACE cs342index
/


create table ddjo_inventoryinfo(
        ItemID number(9),
        Total number,
        PurchasedSinceLast number,
        SoldSinceLast number,
        unnaccountedfor number,
        period date,
        CONSTRAINT fk_inventory_item FOREIGN KEY (itemID) REFERENCES
ddjo_item(itemID),
        CONSTRAINT fk_inventory_info FOREIGN KEY (period) REFERENCES
ddjo_inventory(period)
)
    PCTFree 5
    PCTUSED 15
    TABLESPACE cs342index
/
```

```
create table ddjo_inventory(
    period date,
    ESSN number(9),
    CONSTRAINT pk_inventory PRIMARY KEY(period),
    CONSTRAINT fk_inventory_employee FOREIGN KEY (ESSN) REFERENCES
ddjo_employee(SSN)
)
    PCTFree 5
    PCTUSED 15
    TABLESPACE cs342index
/



create table ddjo_itemsoldinfo(
    TID number(9),
    quantity number,
    itemID number(9),
    CONSTRAINT fk_itemsold_info FOREIGN KEY (TID) REFERENCES ddjo_itemsold
(TID),
    CONSTRAINT fk_itemsold_item FOREIGN KEY(itemID) REFERENCES ddjo_item
(itemID)
)
    PCTFree 5
    PCTUSED 15
    TABLESPACE cs342index
/




create table ddjo_ItemSold(
    TID number(9),
    sdate_time date,
    ESSN number(9),
    Price number,
    CONSTRAINT pk_ItemSold PRIMARY KEY(TID),
    CONSTRAINT fk_ItemSold_employee FOREIGN KEY (ESSN) REFERENCES
ddjo_employee(SSN)
)
    PCTFree 5
    PCTUSED 15
    TABLESPACE cs342index
/



create table ddjo_item (
    name varchar2(30) not null,
    itemID number(9) not null,
    priceperunit number not null,
```

```
    CONSTRAINT pk_item PRIMARY KEY(itemID)
)
    PCTFree 5
    PCTUSED 15
    TABLESPACE cs342index
/



create table ddjo_logtable(
    oldVal varchar(40),
    newVal varchar2(40)
)
pctfree 5
pctused 15
tablespace cs342index
/




create table ddjo_purchaseinfo(
        IID number(9),
        quantity number,
        TID number(9),
        CONSTRAINT fk_purchase_item FOREIGN KEY (IID) REFERENCES
ddjo_item(itemID),
        CONSTRAINT fk_transaction FOREIGN KEY (TID) REFERENCES
ddjo_purchase(TID)
)
        PCTFree 5
        PCTUSED 15
        TABLESPACE cs342index
/



create table ddjo_Purchase(
    TID number(9),
    pdate_time date,
    essn number(9),
    price number,
    Sname varchar2(30),
    CONSTRAINT pk_Purchase_key PRIMARY KEY(TID),
    CONSTRAINT fk_purchase_employee FOREIGN KEY (ESSN) REFERENCES
ddjo_employee(SSN),
    CONSTRAINT fk_purchase_supplier FOREIGN KEY (Sname) REFERENCES
ddjo_supplier(name)
)
    PCTFree 5
    PCTUSED 15
    TABLESPACE cs342index
```

```
/



create table ddjo_supplier(
    name varchar2(30),
    phone# varchar2(10),
    address varchar2(30),
    CONSTRAINT pk_supplier PRIMARY KEY(name)
)
    PCTFree 5
    PCTUSED 15
    TABLESPACE cs342index
/



CREATE VIEW ddjo_PURCHASES AS
    SELECT p.tid,p.sname, e.name, p.pdate_time, p.price, i.itemid, pi.quantity
    from ddjo_purchase p, ddjo_purchaseinfo pi, ddjo_item i, ddjo_employee e
    where p.tid = pi.tid and e.ssn = p.essn and pi.iid = i.itemid
    order by p.pdate_time;

create view ddjo_inventory_view as
    select i.period, i.essn, it.name, ii.total, ii.purchasedsincelast
    ,ii.soldsincelast, ii.unnaccountedfor
    from ddjo_inventory i, ddjo_inventoryinfo ii, ddjo_item it
    where i.period = ii.period and it.itemid = ii.itemid
    order by i.period;

create view ddjo_sales as
    select i.tid, i.essn, i.sdate_time, i.price, it.name, ii.quantity
    from ddjo_itemsold i, ddjo_itemsoldinfo ii, ddjo_item it
    where i.tid = ii.tid and it.itemid = ii.itemid
    order by sdate_time;



--Function to return average purchase price
create or replace function ddjo_avgPrice (n in number) return number is

    s number(9, 2) := 0.0;
    p number(7, 2);

    cursor c1 is select price from ddjo_purchase order by price desc;

begin
```

```
    open c1;
    for i in 1 .. n loop

        fetch c1 into p;
        s := s + p;

    end loop;

    close c1;

    return s / n;

exception

when others then
    raise_application_error( -40001, 'An error occured in ' || sqlcode ||
                             '-ERROR-' || sqlerrm);
end;
/



--stored procedure for inserting an employee
create or replace procedure ddjo_addemployee( p_ssn in number, p_name in
varchar2,
    p_phone in number, p_address in varchar2,p_hdate in date, p_pay in number)
as
    begin
    insert into ddjo_employee values(p_ssn, p_name, p_phone, p_address,
p_hdate, p_pay);

exception
when others then
    raise_application_error( -40001, 'An error occurred in ' || sqlcode ||
                                '-ERROR-' || sqlerrm );
end ddjo_addemployee;
/




--stored procedure for deleting a record
create or replace procedure ddjo_deletesupplier (supname in varchar2) is
    begin
        delete from ddjo_purchaseinfo pi where exists (select * from
ddjo_purchase p where pi.tid = p.tid and p.sname = supname);
        delete from ddjo_purchase where sname = supname;
        delete from ddjo_supplier where name = supname;

        exception
        when others then
```

```
            raise_application_error( -40001, 'An error occurred in ' || sqlcode
                                    || '-ERROR-' || sqlerrm );

    end ddjo_deletesupplier;
/


create or replace procedure ddjo_deleteemployee (ename in varchar2) is
    begin
        delete from ddjo_inventoryinfo ii where exists (select * from
            ddjo_inventory i where exists (select * from ddjo_employee e where
                e.ssn = i.essn and ii.period = i.period and e.name = ename));
        delete from ddjo_inventory i where exists (select * from ddjo_employee
            e where e.name = ename and e.ssn = i.essn);
        delete from ddjo_itemsoldinfo ii where exists (select * from
            ddjo_itemsold  i where exists(select * from ddjo_employee e where
                e.name = ename and i.essn = e.ssn and ii.tid = i.tid));
        delete from ddjo_itemsold i where exists (select * from ddjo_employee e
            where ename = e.name and i.essn = e.ssn);
        delete from ddjo_purchaseinfo pi where exists (select * from
            ddjo_purchase p where exists (select * from ddjo_employee e where
                ename = e.name and p.essn = e.ssn and pi.tid = p.tid));
        delete from ddjo_purchase p where exists (select * from ddjo_employee e
            where ename = e.name and p.essn = e.ssn);
        delete from ddjo_employee e where e.name = ename;

        exception
        when others then
            raise_application_error( -40001, 'An error occurred in ' || sqlcode
                                    || '-ERROR-' || sqlerrm );

    end;
/




create or replace procedure ddjo_deleteinventory (idate in date) is
begin
    delete from ddjo_inventoryinfo i where i.period = idate;
    delete from ddjo_inventory i where i.period = idate;

        exception
        when others then
            raise_application_error( -40001, 'An error occurred in ' || sqlcode
                                    || '-ERROR-' || sqlerrm );
end ddjo_deleteinventory;
/
```

```
create or replace procedure ddjo_deleteitem (iname in varchar2) is
begin
    delete from ddjo_inventoryinfo ii where exists (select * from ddjo_item i
        where iname = i.name and i.itemid = i.itemid);
    delete from ddjo_purchaseinfo p where exists (select * from ddjo_item i
        where iname = i.name and p.iid = i.itemid);
    delete from ddjo_itemsoldinfo ii where exists (select * from ddjo_item i
        where iname = i.name and ii.itemid = i.itemid);
    delete from ddjo_item i where iname = i.name;
        exception
        when others then
            raise_application_error( -40001, 'An error occurred in ' || sqlcode
                                || '-ERROR-' || sqlerrm );

end;
/
```

```
create or replace procedure ddjo_deletepurchase (npid in number) is
begin
    delete from ddjo_purchaseinfo p where npid = p.tid;
    delete from ddjo_purchaseinfo p where npid = p.tid;

        exception
        when others then
            raise_application_error( -40001, 'An error occurred in ' || sqlcode
                                || '-ERROR-' || sqlerrm );

end ddjo_deletepurchase;
/
```

```
create or replace procedure ddjo_deletesale (sid in number) is
begin
    delete from ddjo_itemsoldinfo i where sid = i.tid;
    delete from ddjo_itemsold i where sid = i.tid;

        exception
        when others then
            raise_application_error( -40001, 'An error occurred in ' || sqlcode
                                || '-ERROR-' || sqlerrm );
end ddjo_deletesale;
/
```

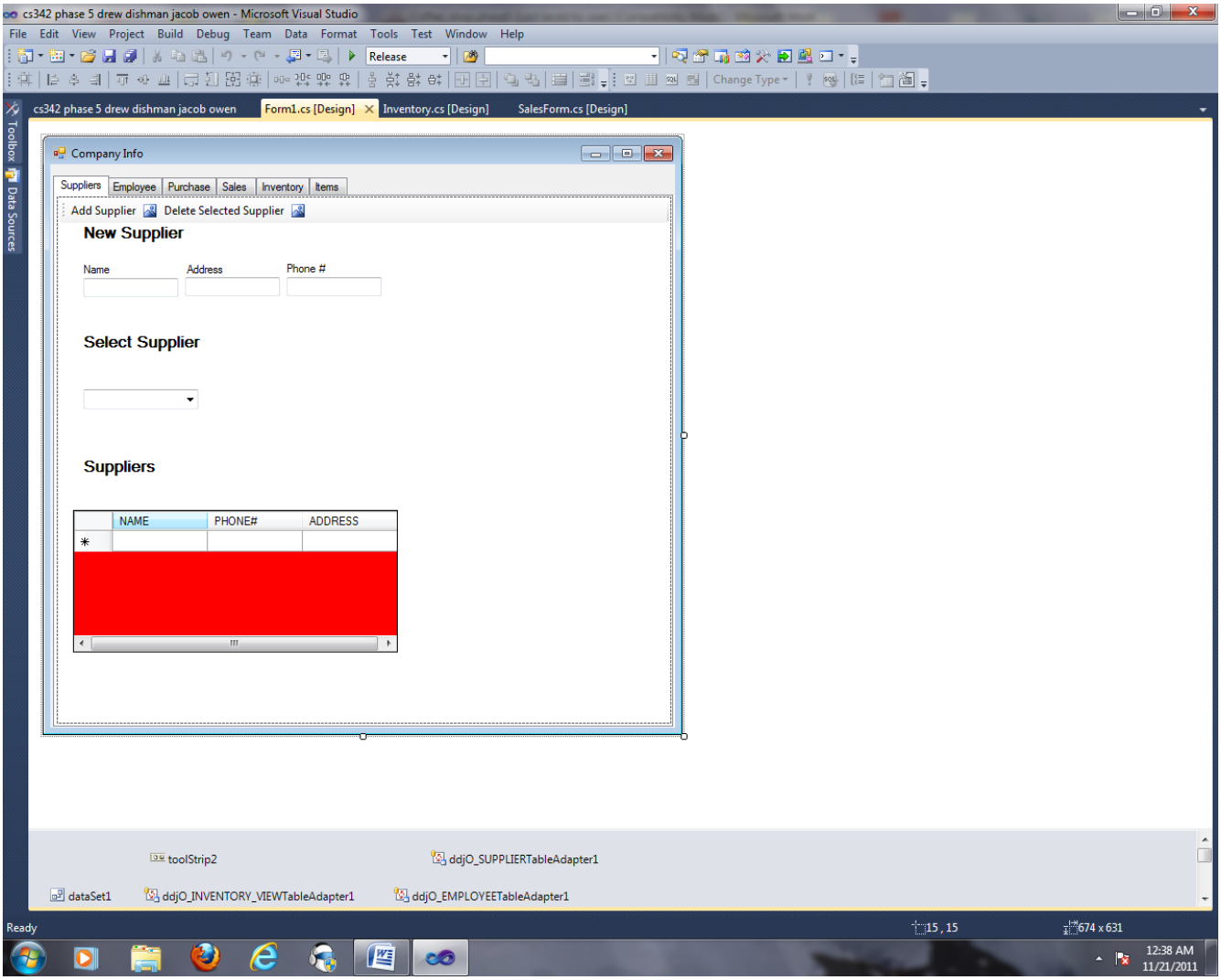
```
--Trigger to be fired on deletion of employee
create or replace trigger ddjo_deleteEmployee
before update or delete on ddjo_employee
for each row
begin
    insert into ddjo_logtable
    values (to_char(:old.ssn) || ' ' || to_char(:old.name) || ' ' ||
        to_char(:old.phone#) || ' ' || to_char(:old.address) || ' ' ||
        to_char(:old.date_hired) || ' ' || to_char(:old.pay)
        ,to_char(:new.ssn) || ' ' || to_char(:new.name) || ' ' ||
        to_char(:new.phone#) || ' ' || to_char(:new.address) || ' ' ||
        to_char(:new.date_hired) || ' ' || to_char(:new.pay) );
   exception
        when others then
            raise_application_error( -40001, 'An error occurred in ' || sqlcode
                                    || '-ERROR-' || sqlerrm );
end ddjo_deleteEmployee;
/



--stored procedure for inserting an employee
create or replace procedure ddjo_addemployee( p_ssn in number, p_name in
varchar2,
    p_phone in number, p_address in varchar2,p_hdate in date, p_pay in number)
as
    begin
    insert into ddjo_employee values(p_ssn, p_name, p_phone, p_address,
p_hdate, p_pay);

exception
when others then
    raise_application_error( -40001, 'An error occurred in ' || sqlcode ||
                                '-ERROR-' || sqlerrm );
end ddjo_addemployee;
/



--The most expensive purchase from a supplier
select * from ddjo_purchase p where not exists(select * from ddjo_purchase p2
where (p2.price > p.price))
/
```

```
--employees who have made no sales last month
select * from ddjo_employee e where exists(select * from ddjo_itemsold i where
e.SSN = i.ESSN
                                                                          and
i.sdate_time > to_date(

'08/31/2011','mm/dd/yyyy')
                                                                        and
i.sdate_time < to_date(

'10/01/2011','mm/dd/yyyy'))
/



--Suppliers from whom we've bought every item
select * from ddjo_supplier s where not exists(select * from ddjo_item i where
exists(select * from ddjo_purchaseinfo p where not exists(select * from
ddjo_purchase p1

where

(p1.tid = p.tid and s.name = p1.sname) and ( i.itemid = p.iid))))
/




--most popular item last month
select i.name
       from ddjo_item i
       where (select max(s.itemid)
              from ddjo_sales s
              where (s.sdate_time > to_date('08/31/2011','mm/dd/yyyy') and
s.sdate_time < to_date('10/01/2011','mm/dd/yyyy'))) = i.itemid
/




--smallest number of items purchased in one transaction

select min(quantity)
from ddjo_purchaseinfo
/
```

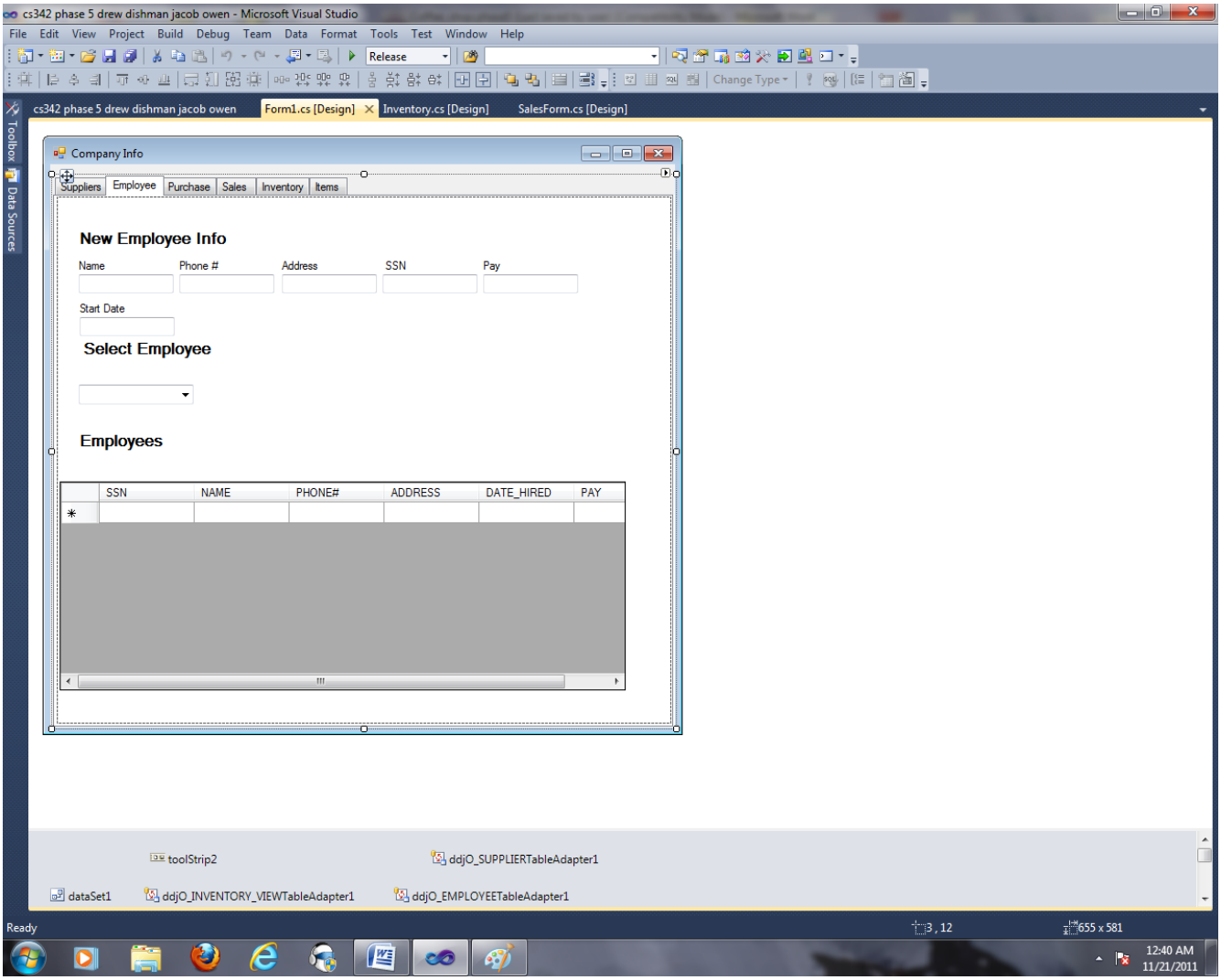
```
--employees who make 10 dollars an hour

select *
from ddjo_employee
where pay = 10
/




--least expensive sale
select s.*
from ddjo_sales s
where not exists(select s2.price
    from ddjo_sales s2 where s2.price < s.price)
/




select avg(i.price)
from ddjo_itemsold i
where i.sdate_time > to_date('08/01/2011','mm/dd/yyyy') and i.sdate_time <
to_date('10/01/2011','mm/dd/yyyy')
/




--The most expensive purchase from a supplier

CS342 SQL> @q1

       TID PDATE_TIM       ESSN      PRICE SNAME
---------- --------- ---------- ---------- ------------------------------
        11 04-JAN-11  100000000    4264.78 Chevrolet Movie Theater

CS342 SQL> @q2

       SSN NAME                           PHONE#     ADDRESS
DATE_HIRE        PAY
---------- ------------------------------ ---------- -------------------------
---- --------- ----------
```

```
 111111111 Teapot Sally                    6615185205 921 Mothergoose Ave
30-JAN-10        8.5

CS342 SQL> @q3

no rows selected

CS342 SQL> @q5

MIN(QUANTITY)
-------------
            8

CS342 SQL> @q6


       SSN NAME                            PHONE#     ADDRESS
DATE_HIRE        PAY
---------- ------------------------------ ---------- -------------------------
---- --------- ----------
 111111000 Trashcan James                  6614326507 2301 Alley St
17-FEB-04         10
 111111110 Scuba Steve                     6618341968 109 Seapines Ln
22-NOV-01         10

CS342 SQL> @q7

       TID       ESSN SDATE_TIM      PRICE NAME
QUANTITY
---------- ---------- --------- ---------- ------------------------------ -----
-----
         2  110000000 23-FEB-11          2 Vanilla Frappuccino
1
         2  110000000 23-FEB-11          2 Pastry
1

CS342 SQL> @q8

AVG(I.PRICE)
------------
         3.5
```

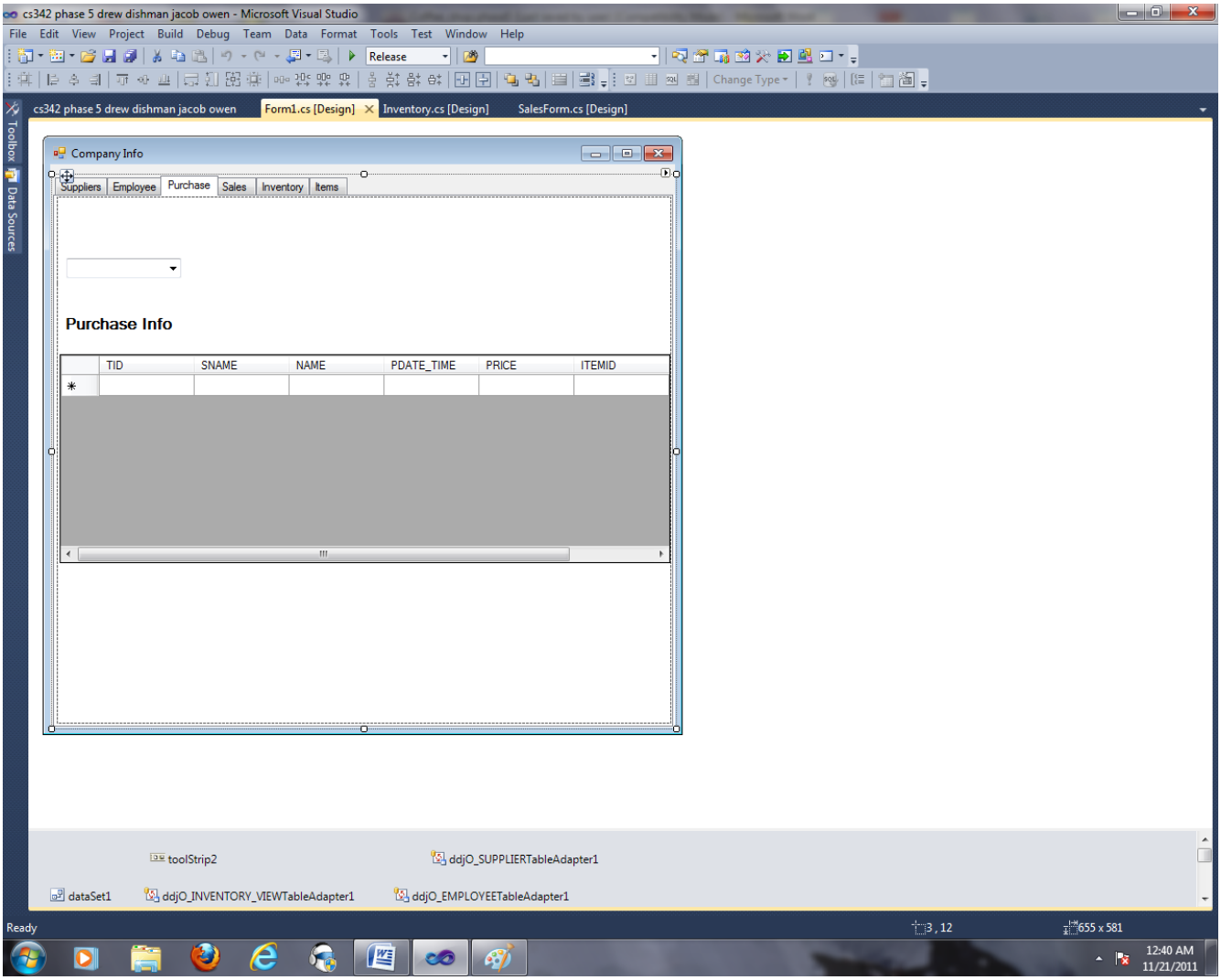
C. Screen shots of our Menu Display/Descriptions of the screen shots:



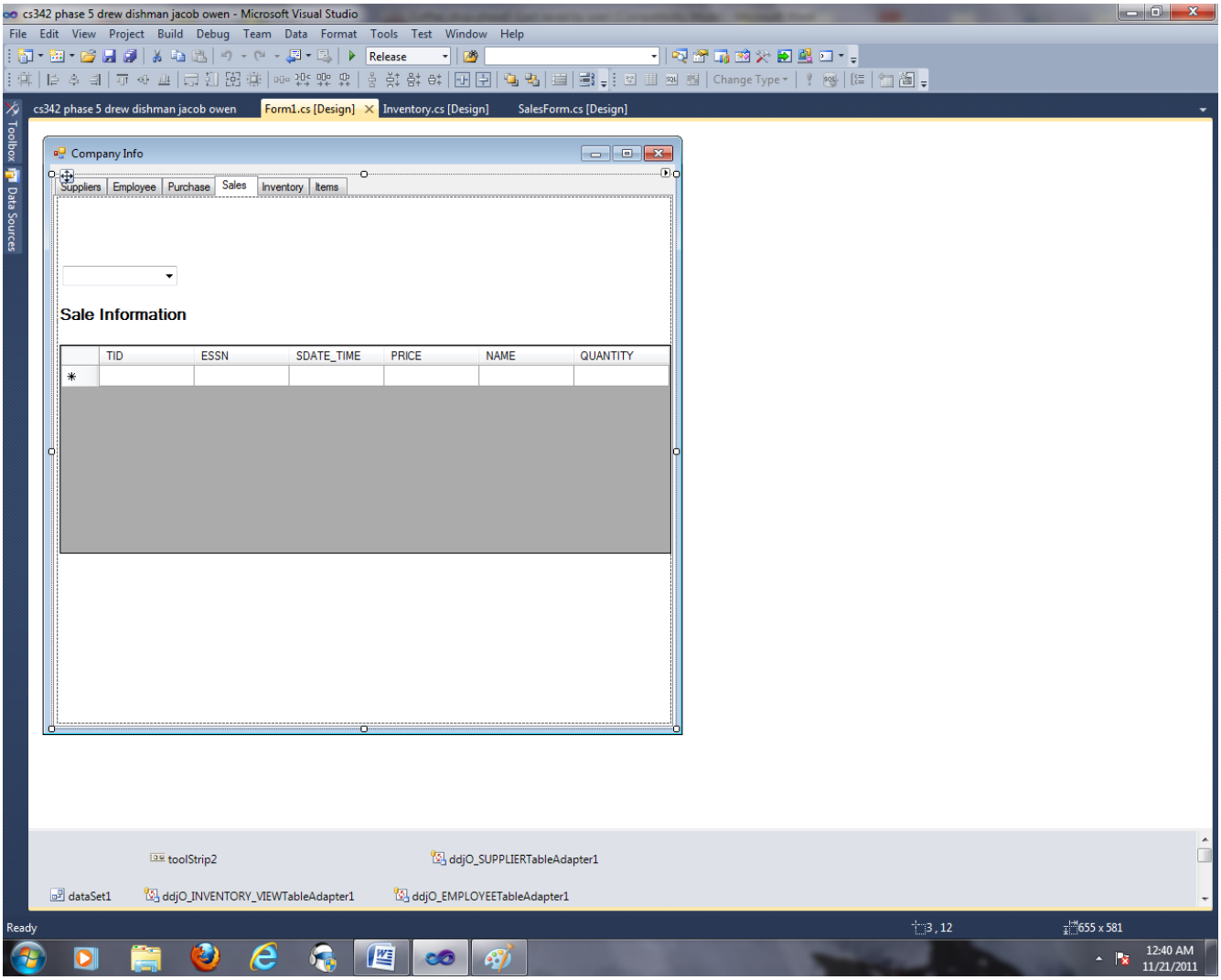
This the Suppliers tab of our GUI where you can add/delete suppliers from the Database.



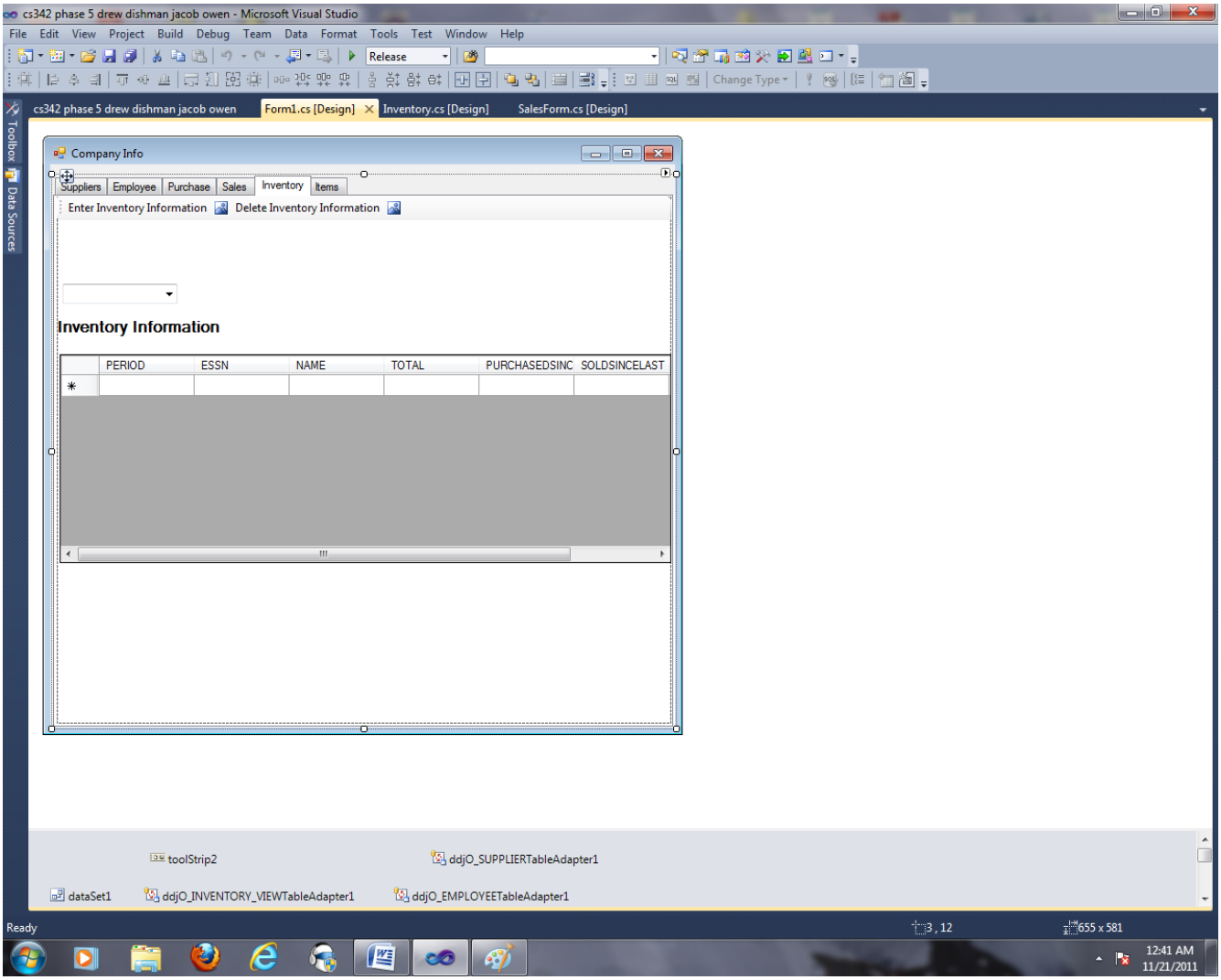
This is the Employee tab where you can add/delete Employees from the database.



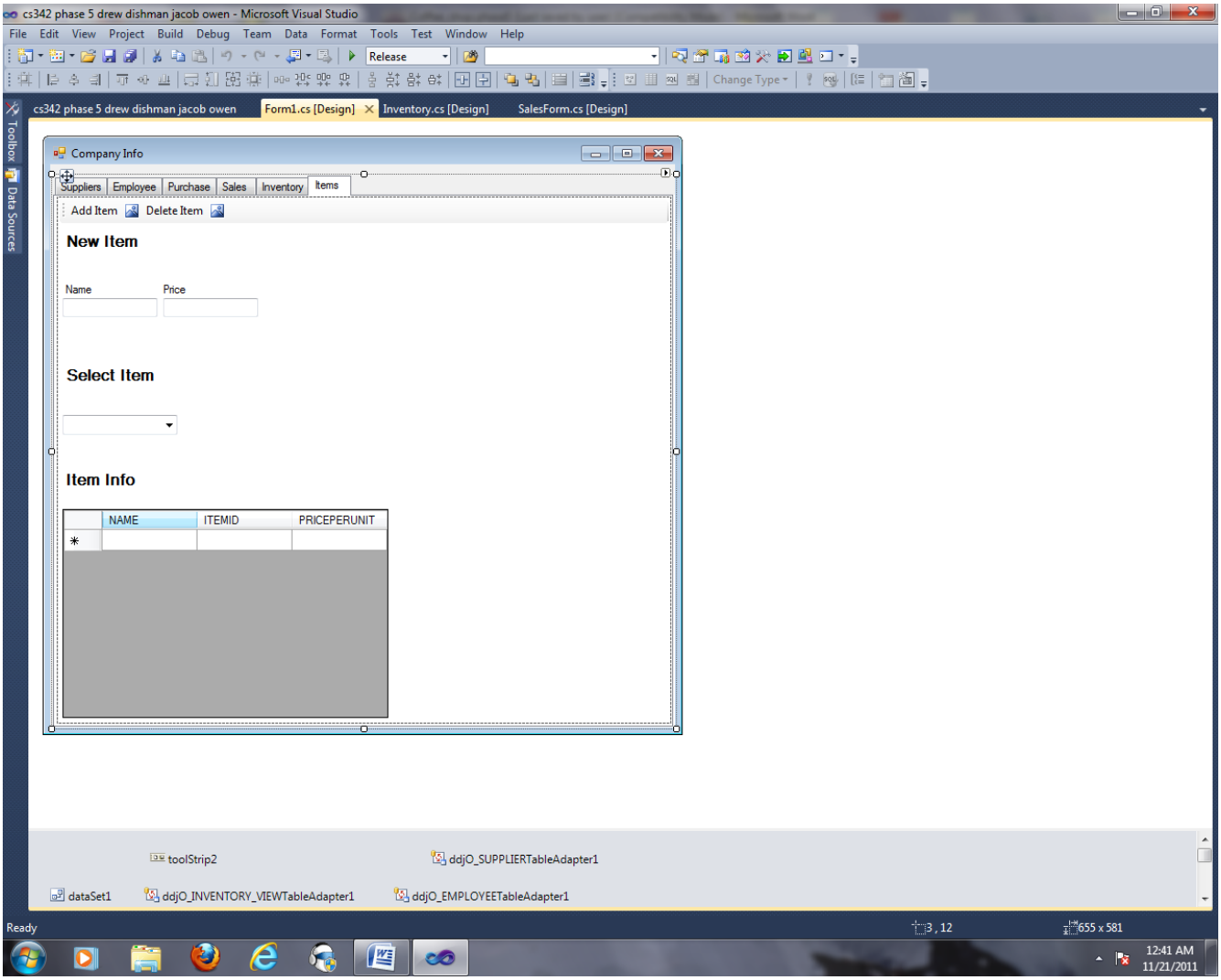
This is the Purchase tab, where you can make purchases/delete purchase info from the database.



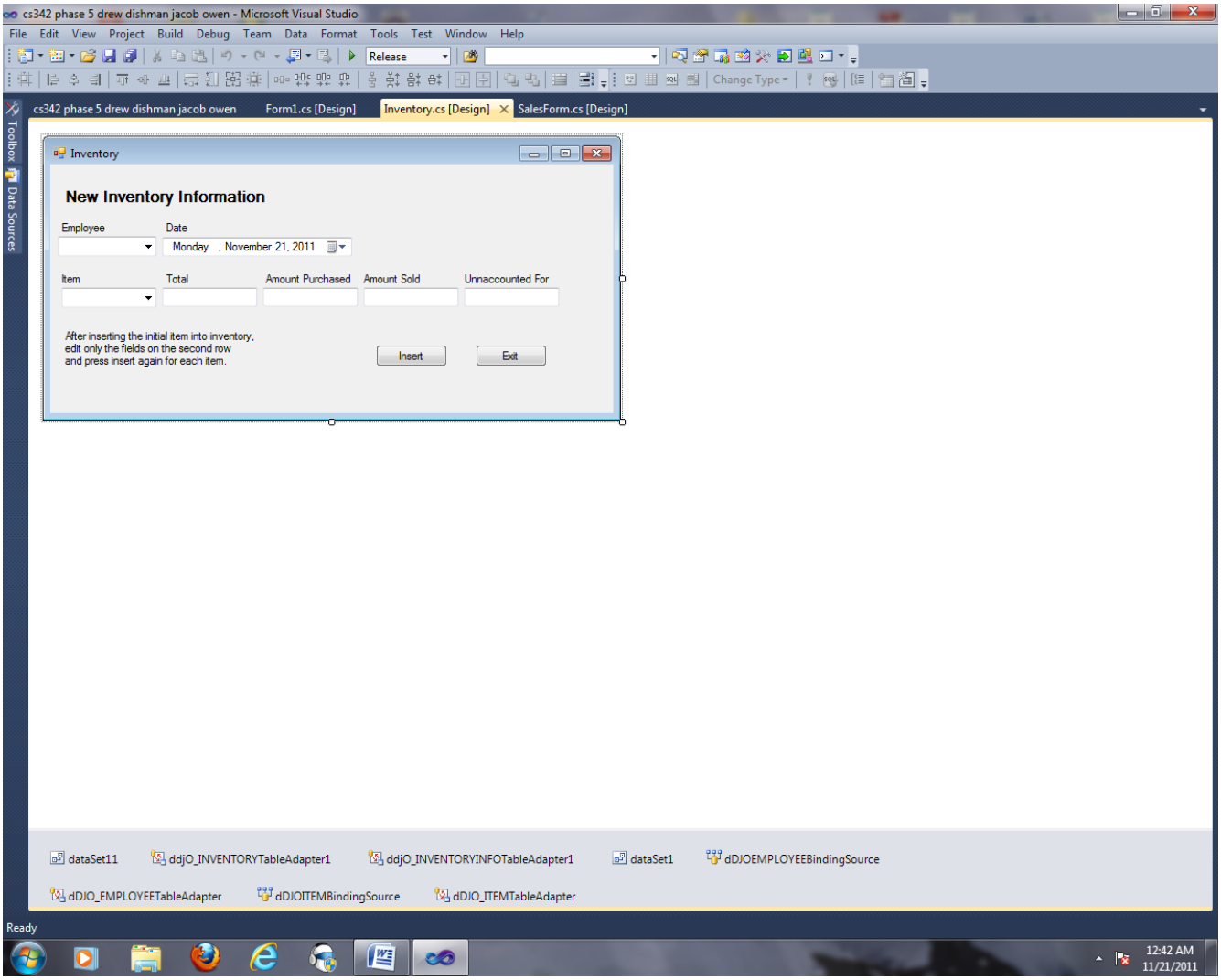
This is the Sales tab, where you can add new sale/delete sale info from the database.



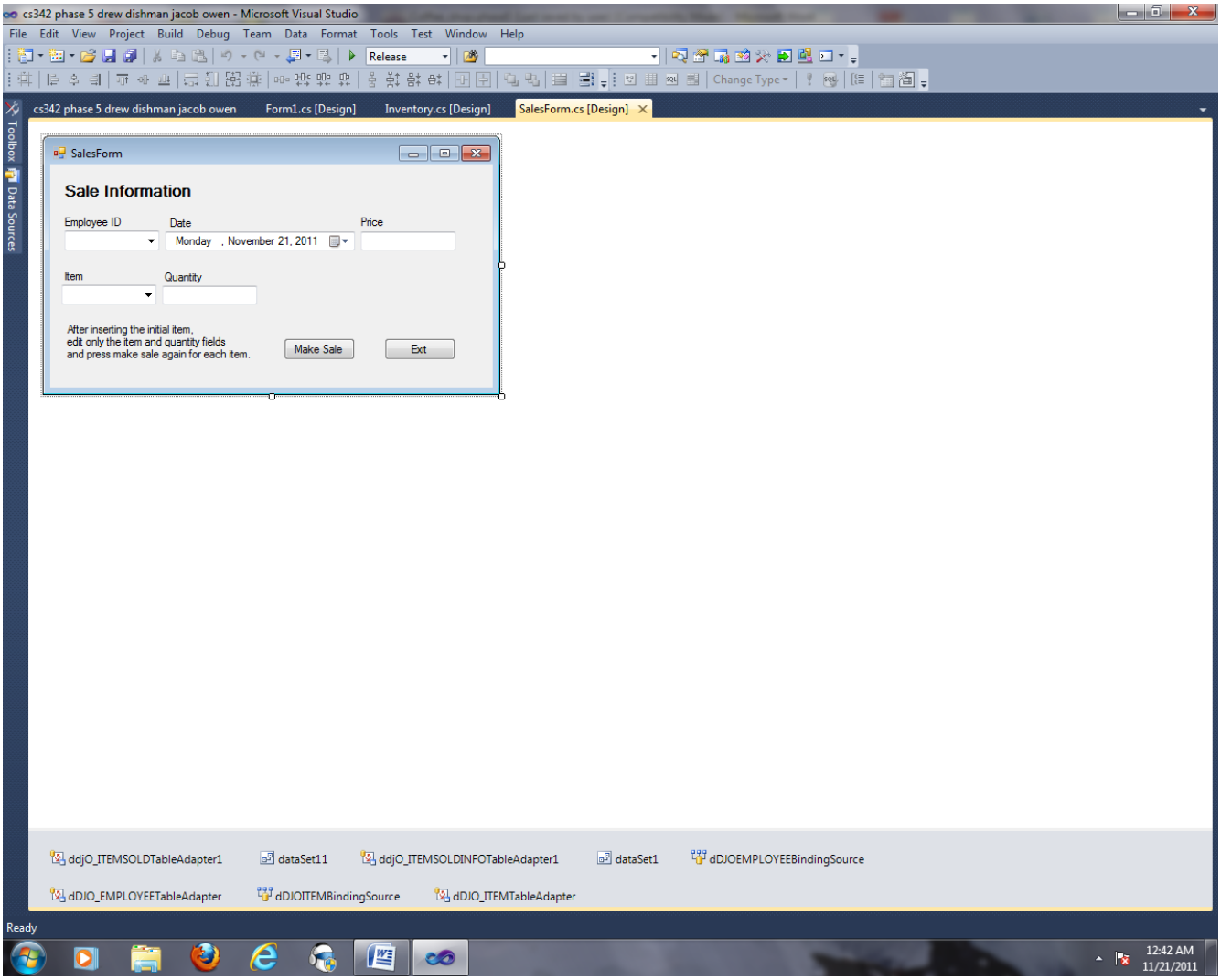
This is the Inventory Tab, where you can enter inventory information/delete from the Database.



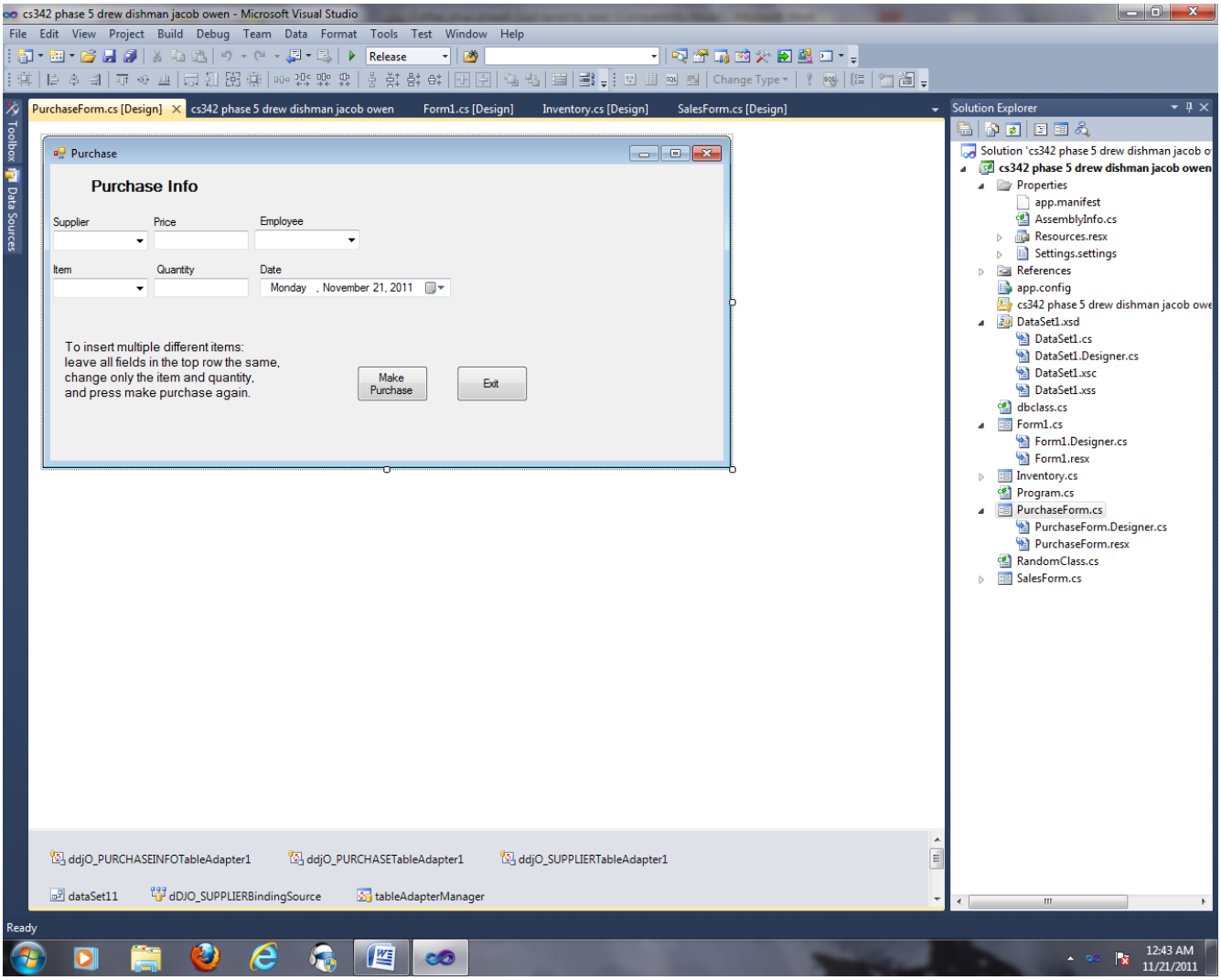
This is the Items tab, where you can add/ delete items from the Database.



This is the Inventory Information tab, where you insert new inventory information.



This is the Sale Information tab, where you insert new sales.



This is the Purchase Info tab, where you can

D. Description of our Code:

2. Descriptions of major classes: We do not have most of the classes asked for in phase 5 because we were not able to establish a connection on Microsoft Visual Studios 2010 to the Oracle Database, so we added Oracle as a datasource.

3. Major features of our GUI program:

Our GUI program features a single form with 6 different tabs: supplier, employee, item, inventory, purchase, and sales. Then we have 3 other forms with a single tab: sale info, purchase info, and inventory info. Users can reload data by clicking on the data grid.

4. Studied a C# book in order to learn more about Microsoft Visual Studio and C# language.

E. /D(1.) Major steps of designing and implementing a database application:

In 6 Phases:

Phase 1: Requirements Collection and Analysis: Analyze the expectations of the users and the intended uses of the database in as much detail as possible.

Phase 2: Conceptual Database Design: Examine the data requirements resulting from Phase 1 and produce a conceptual database schema.

Phase 3: Choice of a DBMS

Phase 4: Logical Database Design (Data Model Mapping): Create a conceptual schema and external schemas in the data model of the selected DBMS by mapping those schemas produced in Phase 2. The result of this phase should be DDL (data definition language) statements in the language of the chosen DBMS that specify the conceptual and external level schemas of the database system.

Phase 5: Physical Database Design: The process of choosing specific file storage structures and access paths for the database files to achieve good performance for the various database applications.

Phase 6: Database System Implementation and Tuning: Language statements in the DDL, including the SDL of the selected DBMS, are compiled and used to create the database schemas and database files. The database can then be loaded (populated with the data. If data is to be converted from an earlier computerized system, conversion routines may be needed to reformat the data for loading in the new database.