Grocery Store Project

CMPS 342

Christian Trahan

Brandon Jones

DATABASE SYSTEMS PROJECT

Table of Contents

Phase III: Implementation of Relational Database

Phase IV: Stored Procedures

Phase V: Stored Procedures

1.1 Fact Finding Techniques and Information Gathering

Personal Experience:
We have all used grocery stores our entire life and this experience gives one a basic understanding of how the grocery system operates. This includes inventory, check-out, pricing, customer needs, etc. Also, Christian worked for a small grocery chain when he was younger which introduced him to grocery store inventory control and checking out. In addition, Brandon has worked at Albertsons grocery chain for many years and we will incorporate this experience into the design. These past and present experiences will qualify this team to build a grocery store prototype database system.

Enterprise Research:
Types of data needed to keep track of in this system will be the item information, checkout activity, inventory control, employee information, and customer activity. Item information is critical to a grocery store as a grocery store moves a tremendous amount of product each day. Item data that will need to be incorporated into the design include: UPC (manufacture's code), ID, brand, description, price, cost, weight, shape, size, and if it is taxable.

This fictitious grocery model will have multiple stores. Each store will have a record in the 'store' entity including a unique ID and address. Inventory will have a store ID component plus item information. This will allow a manager to see what items a store has, the quantity, and the current dollar value of items in inventory.

1.2 Data gathering, operations on data, and reports

Customer information is critical to any successful business and the grocery business is no exception. Each customer will have basic information stored and this will be linked to their purchases. Managers can run reports viewing which store they shopped at, what they bought, how many total transactions and how much money they spent.

Customers have to be able to select product for checkout. They also will need to have their purchases subtotaled and taxes added so they know what they will have to pay. After the transaction is complete, a receipt should be displayed. For this project, there will be a web form a user, customer, can choose items and add them to their checkout basket. Once the ‘checkout’ button is pressed, the items will be deducted from inventory. There will be no actual commerce mechanism in this mock up.

Employees work at a store. The employees will have a unique ID. Employees group contains cashiers and managers. Employees could have dependents.

Employees work at a store. The employees will have a unique ID. Employees group contains cashiers and managers.

Cashiers help checkout the customers. They work hourly and they each assigned one store. The cashiers log into the system with a password at the beginning of their shift. They have a unique ID, a hire date, and a password change date. In this project cashier information will be shown in certain management reports. The user entering the items into the checkout form will be both the customer and the cashier.

Managers are also employees and have a unique ID. Managers supervise other employees. They also work at specific stores. Managers have a higher security clearance that also them to run certain reports and adjust inventory.

Management reports needed for management will have to answer questions including the current status of inventory quantity per store, price per item, customer activity, and sales totals. Managers can query employee information like wage information, store worked, dependents and manager. Inventory report(s) will include quantity in each store and the value of the current inventory. The report will have a ‘restock’ button that will be used to reset all inventory items to a preset quantity and date so a user can simulate purchasing items multiple times. In the real world the store would order more inventory from their warehouse.

1.3 Introduction to Enterprise/Organization

The grocery store industry is a multi-billion dollar industry. It touches everyone on society since we all eat. Through tough competition a grocery chain must run efficient and reduce cost overruns. This includes using updated information in its purchasing decisions, inventory control, store stocking, customer satisfaction, buying trends, and a host of other business concerns.

The business of grocery stores is the ability to provide their customers with well priced and fresh food items everyday. This seemingly simple concept is very complex when put into practice. The company must source its inventory from the U.S. and foreign countries. They must buy at a competitive price then deliver the food items to the individual store quickly. They must handle food changes as the seasons change. They must provide quick checkout service to their customers and be able to keep stores stocked with high velocity items.

Grocery stores also have many employees under management to make the business work. Operationally these many moving parts plus many employees combine for a complex operation with tight profit margins. Managers must be able to use their employees wisely and know their employee costs. Inventory in the stores runs in real time and some store even have systems in place to automatically re-order items from their distribution warehousing. Many of these capabilities are only possible through the use of an advanced RDMS system. This RDMS model is tied into every aspect of the corporation to GPS on trucks to the checkout at one of the stores.

1.4 Project & Database scope

This project will focus on small aspect of the grocery enterprise simulating a customer buying items by selecting them on a form as if taking them from the shelves. They can then see their subtotal and tax and finish the transaction. They only are able to press ‘buy’ to finish, as there is no representation of money. Customers don’t log into the system and only have one checkout UI to interact with. The project removes the items once purchased by updating the inventory.

A manager level user can view certain reports on inventory, customer activity and personal information using data stored in the tables. A manger will have to login into the system to use it. Employees will also have to log in to use the system. Their information will appear in reports that managers run.

CUSTOMER: This entity type represents all the people that shop at the grocery store. A customer performs a checkout. The CUSTOMER entity relates to the CHECKOUT table via the BUY ITEM relationship. The Cust_ID primary key is a foreign key in the checkout table.

EMPLOYEES: This entity type represents all the people that work for the grocery chain. It is a superclass because both managers and salaried people are in this entity type. It related to the STORE entity with the WORKS FOR relationship. It has a recursive relationship with SUPERVISING as employees manage themselves. Stores are managed by employees via the MANAGER relationship. Employees can have dependents through the DEPENDENTS OF relationship.

DEPENDENTS: This is a weak entity type representing anyone who is a dependent of an EMPLOYEE via the DEPENDENTS OF relationship.

CHECKOUT: This entity type represents an atomic transaction of a customer purchasing items in the store. It relates to STORE via the CHECKOUT LOCATION relationship. It relates to CUSTOMERS via the BUY ITEMS relationship. It relates to the employee who performed the checkout with the EMPLOYEE CHECKOUT ACTION relationship. It is connected to the ITEMS entity through the CHECKOUT ACTION relationship. This relationship will become a table using the PK from CHECKOUT and ITEMS to join every item on each individual checkout transaction. CHECKOUT needs a ‘subtotal’ entity as it is calculated at the time of purchase with those specific item prices. If we try to derive this number later any price change would also change the subtotal.

ITEMS: This entity type represents the individual store items someone would purchase like milk, cheese, meat, etc. ITEMS is related to CHECKOUT as described above.

STORE: This entity type represents the actual brick and mortar buildings where food is placed and customers go to and buy. It has two relationships with EMPLOYEES. One is WORKS FOR and that relationship describes which EMPLOYEES are at which STORE. The second relationship is MANAGES FOR. This connects which manager supervises which store. The Store_ID primary key is useful as foreign key in multiple tables.

STORE CONTAINS: This entity type represents the many different types of reports that could be generated from this relationship. These reports include: inventory total, inventory per store, cost of inventory, resale price of inventory, and many more. The STORE relates to the INVENTORY through the STORE CONTAINS relationship.

ITEMS: This entity type represents how many and what type of items are in a checkout. It relates to CHECKOUT via the CHECKOUT ACTION relationship. It is also related to STORES through the STORE CONTAINS relationship. This relationship passes on information from the store where the transaction is process to the final report; the receipt.

1.5 Entity Set Description

CUSTOMER:

Name	Cust_ID	Name	Phone	Email	DateLastTrans	DateCreated
Description	PK	name	Cell phone	email	Last shopping date.	Date joined, first trans.
Domain/Type	Number(9)	Varchar(128)	Number(10)	Varchar(128)	Date	date
Value Range	Positive number	ASCII	Valid numeric chars	ASCII	Any date equal to or greater than DateCreated	Any date before present
Default Value	Next available	Passed value	null	null	null	Passed value
NULL Allowed?	NO	NO	YES	YES	YES	NO
Unique?	YES	NO	NO	NO	NO	NO
Single or Multi-value?	Single	Single	Single	Single	Single	Single
Simple or Composite?	Simple	Simple	Simple	Simple	Simple	Simple

Candidate keys: Cust_ID, Name, Phone, Email
Primary key: Cust_ID
Weak/Strong: strong
Fields to be indexed: Cust_ID

Notes: Customer ID is the ground work to enable customer tracking like most stores now have with their employee cards

DEPENDENTS:

Name	Bdate	Name	Relationship	Email	DateCreated	Emp_ID
Description	Birth date	Employee name	Relation to employee	email	Date added	ID of Employee
Domain/Type	date	Varchar(128)	Varchar(128)	Varchar(128)	date	Number(7)
Value Range	Any date before present	ASCII	ASCII	ASCII	Any date before present	integer
Default Value	null	Passed value	Passed value	null	Auto generated	Passed value
NULL Allowed?	YES	NO	NO	YES	NO	NO
Unique?	NO	NO	NO	NO	NO	NO
Single or Multi-value?	single	Single	Single	Single	Single	Single
Simple or Composite?	simple	Simple	Simple	Simple	Simple	Simple

Candidate keys: none
Primary key: none
Weak/Strong: weak
Partial key: Name
Fields to be indexed: Name

EMPLOYEES:

Name	Emp_ID	Name	SSN	Phone	Store_ID	Date_Start
Description	PK	name	Social security #	Phone number	Foreign key from Store table	Date of employment
Domain/Type	Number(7)	Varchar(128)	Number(9)	Number(10)	Number(3)	Date
Value Range	1 … 9999	ASCII	ASCII	ASCII	1 … 999	Date
Default Value	Next available	Passed value	Passed value	null	FK value	Current
NULL Allowed?	NO	NO	NO	YES	YES	NO
Unique?	YES	NO	YES	NO	NO	NO
Single or Multi-value?	single	Single	Single	Single	Single	Single
Simple or Composite?	simple	Simple	Simple	Simple	Simple	Simple

EMPLOYEES: (continued)

Name	Address	PayType	Password	Email	Pay	Date_End
Description	Home address	0 = Salaried & 1 = hourly	User password	email	Amount they make	Date
Domain/Type	Varchar(128)	Number(1)	Varchar(128)	Varchar(128)	Number(9)	Date
Value Range	ASCII	0 or 1	ASCII	ASCII	1…999999999	Date
Default Value	Null	Passed value	Passed value	null	Null	NULL
NULL Allowed?	YES	NO	YES	YES	YES	YES
Unique?	NO	NO	NO	NO	NO	NO
Single or Multi-value?	Single	single	Single	Single	Single	Single
Simple or Composite?	Simple	simple	Simple	Simple	Simple	Simple

Candidate keys: Emp_ID, SSN, Email, Name
Primary key: Emp_ID
Weak/Strong: strong

Fields to be indexed: Emp_ID

CHECKOUT:

Name	Checkout_ ID	Cust_ID	Date	Store_ID	Subtotal	Emp_ID	Tax
Description	Specific checkout number	FK from Customer ID	Date of checkout	Foreign key from Store table	Amount of all items on list	Employee who performed checkout	Tax amount on subtotal
Domain/Type	Number(9)	Number(7)	date	Number(3)	double	Number(7)	double
Value Range	integer	integer	Present date	integer	Positive number	integer	Positive number
Default Value	Next available	FK value	Auto generated	FK value	null	FK value	null
NULL Allowed?	NO	NO	NO	NO	NO	NO	NO
Unique?	YES	YES	NO	YES	NO	YES	NO
Single or Multi-value?	single	Single	Single	Single	Single	Single	Single
Simple or Composite?	simple	Simple	Simple	Simple	Simple	Simple	Simple

Candidate keys: Checkout_ID, Cust_ID, Store_ID
Primary key: Checkout_ID
Weak/Strong: strong
Fields to be indexed: Checkout_ID

Notes: Store_ID allows customer to be tracked at what store they are shopping in while Cust_ID allows the reports to trace what the customer has bought and at what data. Subtotal is needed because the price for items changes over time so this allows reports to be generated showing how much a customer has spent instead of how much they would spend were they to buy the products at that time.

ITEMS:

Name	Item_ID	Brand	Description	Price	Cost
Description	Sequence - Unique internal ID #	Brand of item	Item description	Current price	Wholesale cost
Domain/Type	Number(7)	Varchar(32)	Varchar(128)	Double	Double
Value Range	intiger	ASCII	ASCII	Positive number	Positive number
Default Value	Next available	Passed value	Passed value	Passed value	Passed value
NULL Allowed?	NO	NO	NO	NO	NO
Unique?	YES	NO	NO	NO	NO
Single or Multi-value?	single	Single	Single	Single	Single
Simple or Composite?	simple	Simple	Simple	Simple	Simple

ITEMS: (continued)

Name	Shape	Size	UPC	Weight	Taxable
Description	Basic shape description	Basic size description	Manufactures ID	Current weight	Is the item taxed
Domain/Type	Varchar(32)	Varchar(32)	Number(9)	Number(4,2)	integer
Value Range	ASCII	ASCII	Positive number	Positive number	1 for yes 0 for no
Default Value	Null	Null	Passed value	null	1
NULL Allowed?	YES	YES	NO	YES	NO
Unique?	NO	NO	YES	NO	NO
Single or Multi-value?	single	single	Single	Single	Single
Simple or Composite?	simple	simple	Simple	Simple	Single

Candidate keys: Item_ID, UPC
Primary key: Item_ID, UPC
Weak/Strong: strong
Fields to be indexed: Item_ID, UPC

STORE:

Name	Store_ID	Address
Description	Sequence - Unique internal ID #	Physical location
Domain/Type	Number(3)	Varchar(128)
Value Range	integer	ASCII
Default Value	Next available	Passed value
NULL Allowed?	NO	NO
Unique?	YES	NO
Single or Multi-value?	single	Single
Simple or Composite?	Simple	Simple

Candidate keys: Store_ID
Primary key: Store_ID
Weak/Strong: strong
Fields to be indexed: Store_ID

INVENTORY:

Name	Store_ID	Item_ID	Quantity
Description	What store is it in?	What item is in inventory?	How many?
Domain/Type	Number(3)	Varchar(128)	Number(5)
Value Range	integer	ASCII	Positive number
Default Value	FK from Store table, ID.	FK from Item table, ID.	Passed value
NULL Allowed?	NO	NO	NO
Unique?	NO	NO	NO
Single or Multi-value?	single	Single	Single
Simple or Composite?	simple	Simple	Simple

Candidate keys: Store_ID, Item_ID
Primary key: Store_ID, Item_ID
Weak/Strong: strong
Fields to be indexed: Store_ID, Item_ID

Notes: Store_ID is needed here to tie the store to the quantity for reports that are store specific

CHECKOUT ACTION:

Name	Checkout_ID	Quantity	Item_ID
Description	Specific checkout number.	Number of items purchased on that transaction.	Specific item that was checked out.
Domain/Type	sequence	Integer	sequence
Value Range	1 … 999999	Positive number	1 … 9,999,999
Default Value	FK from Checkout table, ID.	Passed value	FK from Items table, ID.
NULL Allowed?	NO	YES	NO
Unique?	NO	NO	NO
Single or Multi-value?	Single	Single	Single
Simple or Composite?	Simple	Simple	Simple

Candidate keys: Checkout_ID, Item_ID
Primary key: Checkout_ID, Item_ID
Weak/Strong: strong
Fields to be indexed: Checkout_ID, Item_ID

MANAGES FOR

Name	Store_ID	Emp_ID	Position
Description	Specific checkout number.	FK from Employee table	Level of management at the store.
Domain/Type	sequence	Number	Number
Value Range	1 … 999	1 … 9999999	1 … 999
Default Value	FK from Checkout table, ID.	FK from Employee table	1
NULL Allowed?	NO	NO	NO
Unique?	NO	NO	NO
Single or Multi-value?	Single	Single	Single
Simple or Composite?	Simple	Simple	Simple

Candidate keys: Checkout_ID, Item_ID
Primary key: Checkout_ID, Item_ID
Weak/Strong: strong
Fields to be indexed: Checkout_ID, Item_ID

1.6 Relationship Set Description

WORKS FOR:

The WORKS FOR relationship describes the interaction between the EMPLOYEE and STORE entity types. It is a binary relationship type. Many employees can work for one store. Every employee must work for a store, but only one store. Therefore there is total participation for the each side of the WORKS FOR relationship. The mapping cardinality of EMPLOYEE:STORE is M:0. The multiplicity of EMPLOYEE in WORKS FOR is 1..M as each employee must be at a store and more than one could be. . The multiplicity of STORE in WORKS FOR is 0..M as a STORE could be new and have no employees initially assigned.

MANAGES FOR:

The MANAGES FOR relationship describes the interaction between the EMPLOYEES entity and the STORE entity. It is a binary relationship. The employees table has an identification for which employees are also managers. These managers are assigned stores to manage. Many managers can be assigned to each store therefore it is a total participation relationship. The POSITION attribute describes the level of management that employee has at that store. The mapping cardinality of EMPLOYEES:STORE within this relationship is M:0. The multiplicity of EMPLOYEE in MANAGES FOR is 1..M as one to many managers can manage a store. . The multiplicity of STORE in MANAGES FOR is 0..M as a STORE could be new and have no managers initially assigned.

CHECKOUT ACTION:

The CHECKOUT ACTION relationship is a binary relationship between the CHECKOUT entity and the ITEMS entity. A checkout transaction can contain many separate items but each item is only contained once in a specific checkout. This relationship is constructed by the combination of the PK of CHECKOUT entity (checkout_ID) and the PK of ITEMS entity (items_ID). This PK merger is represented as a new CHECKOUT_ITEMS entity. The multiplicity between the CHECKOUT entity and the CHECKOUT_ITEMS entity is 1..M with full participation. The multiplicity between the ITEMS entity and the CHECKOUT_ITEMS entity is 1..M also. The ER multiplicity between the entities CHECKOUT and ITEMS is M..M as a checkout can contain many items and many checkouts can contain the same item.

STORE CONTAINER:

The STORE CONTAINER relationship is a binary relationship. It connects the STORE entity with the ITEMS entity. Many items are contained in a store and each item will be in multiple stores. Therefore, the ER multiplicity between the entities STORE and ITEMS is M..M. This relationship is constructed by the combination of the PK of STORE entity (checkout_ID) and the PK of ITEMS entity (items_ID). This PK merger is represented as a new INVENTORY entity. The multiplicity between the STORE entity and the INVENTORY entity is 1..M with full participation. The multiplicity between the ITEMS entity and the INVENTORY entity is 1..M also. There are two attributes added to this STORE CONTAINER relationship, ‘quantity’ and ‘date delivered’. These two attributes and the two PKs will be combined in the relational DB model to form the INVENTORY table.

CHECKOUT LOCATION:

This relationship connects the CHECKOUT entity and the STORE entity. It tells the manager where the checkout transaction happened. It is a binary relationship. Many checkouts can happen at each store and a checkout must have a store as part of it so there is a total participation between the CHECKOUT entity and the STORE entity. The mapping cardinality of CHECKOUT:STORE is M:0. The multiplicity between the CHECKOUT entity and the CHECKOUT LOCATION relationship entity is M..1 as many checkouts happen per location. The multiplicity between the STORE entity and the CHECKOUT LOCATION relationship entity is 1..0.

EMPLOYEE CHECKOUT ACTION

This relationship connects the CHECKOUT entity and the EMPLOYEE entity. It tells the manager who performed the checkout. It is a binary relationship. Many checkouts can happen per employee and a checkout must have a employee ID as part of it so there is a total participation between the CHECKOUT entity and the EMPLOYEE entity. The mapping cardinality of EMPLOYEE CHECKOUT ACTION:EMPLOYEE is M:1. The multiplicity between the EMPLOYEE CHECKOUT ACTION relationship and EMPLOYEE entity is M..1 as many checkouts can be performed by employee. The multiplicity between the EMPLOYEE entity and the CHECKOUT entity is 1…M.

BUY ITEM:

This relationship connects the CUSTOMERS entity to the CHECKOUT entity and is a binary relationship. Many customers can do many checkouts but each checkout has one customer assigned. It’s possible a customer has no checkouts yet if they just signed up. Therefore there is a 0..M mapping cardinality between CUSTOMERS:CHECKOUT. There is partial participation between the CUSTOMERS entity and the BUY ITEM relationship and full participation between the CHECKOUT entity and the BUY ITEM relationship. The multiplicity of CUSTOMERS in BUY ITEM is 0..M. The multiplicity of CHECKOUT in BUY ITEM is 1..M.

1.7 Related Entity Set

Specialization: is the process of defining a set of subclasses of an entity type; this entity type is called the superclass of the specialization. Though I did not use a specific example of specialization in this project, one could see that customers, dependents and employees are all people. Once could make a general super class called ‘people’ and extend these three groups from it.

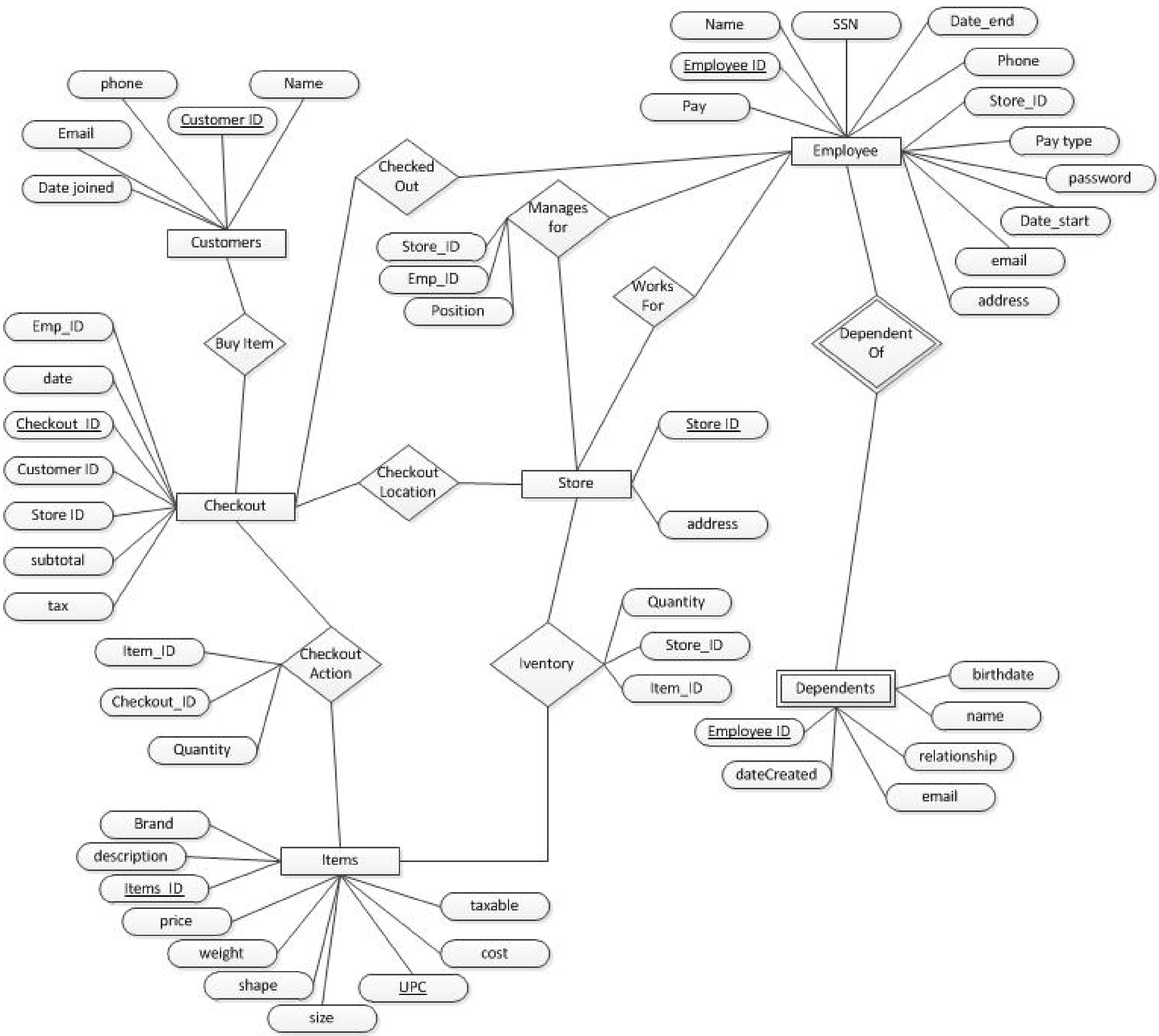
Generalization: is conceptually exactly the same as specialization, except that it is done in the opposite order. Common features from several entity types are identified and generalized into a single superclass. As stated above, one could generalize a superclass ‘people’ from the three independent groups.

Participation Constraints: Specifies whether the existence of an entity depends on its being related to another entity via the relationship type. This constraint specifies the minimum number of relationship instances that each entity can participate in. These constraints are describes above in each of the relationship examples given.

Disjointness Constraint: Specifies that the subclasses of the specialization must be disjoint. This means that an entity can be a member of at most one of the subclasses of the specialization. This project does not have this type of constraint. If the project had a superclass of employees and it broke out salary as one class and hourly as another class then this would be a disjointness constraint since one employee can not be both.

Aggregation: Represents a relationship between a whole object and its component parts. Aggregation and association do not have different structural properties, and are both represented as relationships in the ER model. This project uses aggregation between CHECKOUT & ITEMS entities. We modeled the checkout information that contains items in a separate ‘checkout_items’ aggregate entity.

1.8 E-R Diagram



2.1 ER Model and Relational Model Description

An **entity-relationship model** (ER Model) is a description of a high-level data model. ER Modeling is used in the conceptual design for a database. Entity-relationship modeling is a relational schema database modeling method, used in software engineering to produce a type of conceptual data model (or semantic data model) of a system. This system is usually a relational database with the requirements described in a top-down fashion. Diagrams created using this process are called entity-relationship diagrams, or ER diagrams. The definitive reference for entity relationship modeling is generally given as Peter Chen's 1976 paper. Dr. Chen is a Professor of Computer Science at Louisiana State University in Baton Rouge, LA. Dr. Chen did not in fact invent the concept; the basic ideas appear in earlier papers especially by practitioners, such as that by A. P. G. Brown. However, Chen did more than anyone to formalize and popularize the model, and to introduce it to the academic literature.

An entity may be defined as a thing in the real world which is recognized as being capable of an independent existence and which can be uniquely identified. A relationship captures how two or more entities are related to one another. Relationships can be thought of as verbs, linking two or more nouns. Entities and relationships can both have attributes. Attributes can be simple, composite, single valued or multivalued. Every entity (unless it is a weak entity) must have a minimal set of uniquely identifying attributes, which is called the entity's primary key.

The **relational model** for database management is a database model based on first-order predicate logic, first formulated and proposed in 1969 by Edgar Codd. First-order predicate logic is a system of deduction that extends propositional logic by allowing quantification over individuals of a given domain of discourse. Its core idea is to describe a database as a collection of predicates over a finite set of predicate variables, describing constraints on the possible values and combinations of values. The content of the database at any given time is a finite model of the database. The purpose of the relational model is to provide a declarative method for specifying data and queries

The relational model assumes that all data is represented as mathematical *n*-ary relations, an *n*-ary relation being a subset of the Cartesian product of *n* domains. The relational model has relationship type, or relationship sets, among entities from the entity types. Each relationship instance maps exactly one entity from each side of the relation and will be a subset of the Cartesian product. The number of participating entity types determines the degree of the relation. Reasoning about the data is done in two-value predicate logic, meaning a proposition may be true or false. The basic relational building block is the domain or data type. A tuple is an unordered set of attribute values. An attribute value is a specific valid value for the type of the attribute. A relation is also called a table and defined as a set of *n*-tuples. It is a result of a relational query.

The ER model is clear, easy to understand, and unambiguous. Though it is simple to create and understand, it lacks the details and structure necessary to be applicable in a database management system. This is why the conversion from ER model to relational model is needed.

2.2 Conversion from ER to Relational Model

Step 1: Mapping of Regular Entity Types: For each regular (strong) entity type *E* in the ER schema, create a relation *R* that includes all the simple attributes of *E.* Include simple components of composite attributes. Choose one of *E*s key attributes to be the primary key of *R*. Such relations are sometimes called entity relations because each tuple represents an entity instance.

Step 2: Mapping of Weak Entity Types: For each weak entity type *W* in the ER schema with owner entity type *E*, create a relation *R* and include all simple attributes (or simple components of composite attributes) of *W* as attributes of *R*. Include as foreign key attributes of *R*, the primary key attribute(s) of the relation(s) that correspond to the owner entity type(s). If there is a weak entity type E_2 whose owner is also a weak entity type E_1, then E_1 should be mapped before E_2 to determine its primary key first.

Step 3: Mapping of Binary 1:1 Relationship Types: For each binary 1:1 relationship type *R* in the ER schema, identify the relations *S* and *T* that correspond to the entity types participating in *R*. There are three possible approaches: 1. the foreign key approach, 2. the merged relationship approach, and 3. the cross-reference or relationship relation approach. The first approach is the most useful.

A. **Foreign key approach:** Choose one of the relations (we'll say *S*) and include as a foreign key in *S* the primary key of *T*. It is better to pick an entity type with total participation in *R* in the role of *S*. Include all the simple attributes (or the simple components of composite attributes) of the 1:1 relationship type *R* as attributes of *S*.

B. Merged relation approach: This involves merging the two entity types and the relationship into a single relation. This may be appropriate when both participations are total.

C. Cross-reference or relationship relation approach: Set up a third relation *R* for the purpose of cross-referencing the primary keys of the two relations *S* and *T* representing the entity types. The relation *R* is called a relationship relation or lookup table, because each tuple in *R* represents a relationship instance that relates one tuple from *S* with one tuple of *T*.

Step 4: Mapping of Binary 1:N Relationship Types: For each regular binary 1:N relationship *R*, identify the relation *S* that represents the participating entity type at the *N-side* of the relationship type. Include as foreign key in *S* the primary key of the relation *T* that represents the other entity type participating in *R*. (Include attributes as done previously).

Step 5: Mapping of Binary M:N Relationship Types: For each binary M:N relationship type *R*, create a new relation *S* to represent *R*. Include as foreign key attributes in *S* the primary keys of the relations that represent the participating entity types. Their combination will form the primary key of *S*. Include in *S* all the attributes from the M:N relationship type.

Step 6: Mapping of Multivalued Attributes: For each multivalued attribute *A*, create a new relation *R*. This relation will include an attribute corresponding to *A*, plus the primary key attribute *K*—as a foreign key in *R*—of the relation that represents the entity type or relationship type that has *A* as an attribute. The primary key of *R* is the combination of *A* and *K*. If the multivalued attribute is a composite, we include its simple components.

Step 7: Mapping of *N*-ary Relationship Types: For each *n*-ary relationship type *R*, where $n > 2$, create a new relation *S* to represent *R*. Include as foreign key attributes in *S* the primary keys of the relations that represent the participating entity types. Also include any simple attributes of the *n*-ary relationship type (or simple components of composite attributes) as attributes of *S*. The primary key of *S* is usually a combination of all the foreign keys that reference the relations representing the participating entity types. However, if the cardinality constraints on any of the entity types *E* participating in *R* is 1, then the primary key of *S* should not include the foreign key attribute that references the relation *E*' corresponding to *E*.

Step 8: Options for Mapping Specialization or Generalization: Convert each specialization with *m* subclasses $\{S_1, S_2, \ldots, S_m\}$ and (generalized superclass *C*, where the attributes of *C* are $\{k, a_1, \ldots, a_n\}$ and *k* is the (primary) key, into relation schemas.

- **A.** Create different relations (tables) for each sub class and one for the superclass.
- **B.** Integrate the superclass into each 'M' relation so there are no more relationships.
- **C.** Union all the attributes into one relation and add a indicator, like an integer, that specifies the type of entity it is. Obviously there would be nulls in cells that did not correspond to the type of class that entity was.
- **D.** Similar to option 'C' but use binary flags to determine the entity type after you union all the attributes.

Step 9: Mapping of Categories (Union Types): A category (or union type) is a subclass of the union of two or more superclasses that can have different keys because they can be of different entity types. For mapping a category whose defining superclasses have different keys, it is customary to specify a new key attribute, called a surrogate key, when creating the relation. Because the keys are different, we cannot use just one of them exclusively to identify all tuples in the relation. We can create a relation to correspond to the category, and include any attributes of the category in this relation. The primary key of the new relation is the surrogate key. We also include this surrogate key as a foreign key in each relation corresponding to a superclass of the category. For a category whose superclasses have the same key, there is no need for a surrogate key.

2.3 Constraints

Entity Integrity Constraint: States that no primary key value can be NULL. This is because the primary key value is used to identify individual tuples in a relation. Having NULL values for the primary key implies that we cannot identify some tuples.

Primary Key and Unique Key Constraints: An entity type usually has an attribute whose values are distinct for each individual entity in the entity set. Such an attribute is called a key attribute, and its values can be used to identify each entity uniquely. Sometimes several attributes together form a key. This is a composite key. The composite key should be minimal, but include all component attributes to have the uniqueness property.

Referential Integrity Constraint: Is specified between two relations and is used to maintain the consistency among tuples in the two relations. A tuple in one relation that refers to another relation must refer to an existing tuple in that relation. For this we use a foreign key to reference the primary key of the tuple in another relation.

Check Constraints and Business Rules: A check constraint is a condition that defines valid data when adding or updating an entry in a table of a relational database. A check constraint is applied to each tuple in the table. The constraint must be a predicate. It can refer to a single or multiple columns of the table. The result of the predicate can be TRUE, FALSE, or UNKNOWN, depending on the presence of NULLs. Business rules are constraints that cannot be directly expressed in schemas of the data model, and must be expressed and enforced by the application programs.

2.4 Conversion to Relational Model

CUSTOMERS

Cust_ID	Name	Phone	Email	Date_joined

- Cust_ID - auto-generated integer ID
- Name - 128-letter character string ranged 'a' through 'z' each
- Phone - 10-digit character string ranged '0' through '9'
- Email - 128-letter character string ranged 'a' through 'z' plus the '@' symbol
- DateCreated – valid date
- DateLastTrans – valid date

Candidate key: Checkout_ID, Cust_ID, Store_ID
Primary key: Checkout_ID

CHECKOUT

Checkout_ID	Cust_ID	Store_ID	Tax	Subtotal	Emp_ID	Date

- Checkout_ID - auto-generated integer ID
- Cust_ID - unique integer number identifying each store – FK from CUSTOMER entity
- Store_ID - unique integer number identifying each store – FK from STORE entity
- Emp_ID - unique integer number identifying each store – FK from EMPLOYEE entity
- Tax – double with two digit precision
- Subtotal – double with two digit precision
- Date – Date type

Candidate key: Checkout_ID, Cust_ID, Store_ID
Primary key: Checkout_ID

CHECKOUTACTION

Checkout_ID	Items_ID

- Checkout_ID - auto-generated integer ID – FK from CHECKOUT entity
- Items_ID - auto-generated integer ID – FK from ITEMS entity
- Quantity – integer describing number of items purchased on that transaction.

Candidate key: Checkout_ID, Items_ID
Primary key: Checkout_ ID, Items_ID

ITEMS

Items_ID	Description	Brand	Cost	Price	Weight	Shape	Taxable	Size	UPC

- Items_ID - auto-generated integer ID
- Description - 128-letter character string ranged ‘a’ through ‘z’ each plus special characters
- Brand – 32-letter character string ranged ‘a’ through ‘z’ each plus special characters
- Cost – factory cost of item
- Price – valid positive double with two digit precision
- Weight - valid positive double with two digit precision
- Shape - 32-letter character string ranged ‘a’ through ‘z’ each
- Taxable – 1 digit value decide if item is taxable or not
- Size- 32-letter character string ranged ‘a’ through ‘z’ each
- UPC – valid positive integer

Candidate key: Items_ID
Primary key: Items_ID

EMPLOYEES

Emp_ID	Name	SSN	Phone	Address	Pay Type	Password	Store_ID	Date_Start	Date_End	Pay	Email

- Emp_ID: auto-generated integer ID
- Name - 128-letter character string ranged ‘a’ through ‘z’ each
- SSN 9-digit character integer ranged ‘0’ through ‘9’
- Phone - 10-digit character integer ranged ‘0’ through ‘9’
- Address: physical address of employee
- Pay_Type – is employee a regular employee or a manager
- Password – encrypted password field
- Store_ID: unique integer number identifying each store – FK from STORE entity
- Date_Start – Date hired
- Date_End – Date employment done. If NULL still employed
- Pay – amount employee makes either annual or hour based on pay type
- Email - 128-letter character string ranged ‘a’ through ‘z’ each plus special character ‘@’

Candidate key: Emp_ID, SSN, Name, Email
Primary key: Emp_ID

DEPENDENTS

Emp_ID	Name	Relationship	Email	DateCreated	Birthdate

- Emp_ID: auto-generated integer ID – FK from EMPLOYEES entity
- Name - 128-letter character string ranged ‘a’ through ‘z’ each
- Relationship - 128-letter character string ranged ‘a’ through ‘z’ each plus special characters
- Email - 128-letter character string ranged ‘a’ through ‘z’ each plus special character ‘@’
- DateCreated – valid date
- Birthdate – valid date

Candidate key: Emp_ID
Primary key: Emp_ID

STORE

<u>Store_ID</u>	Address

- <u>Store_ID</u>: auto-generated integer ID
- Address - 128-letter character string ranged ‘a’ through ‘z’ each plus special character ‘@’

Candidate key: Store_ID, Address
Primary key: Store_ID

INVENTORY

<u>Items_ID</u>	Store_ID	Quantity

- <u>Items_ID</u>: auto-generated integer ID – FK from ITEMS entity
- Store_ID - auto-generated integer ID – FK from STORE entity
- Quantity – valid positive integer

Candidate key: Items_ID, Store_ID
Primary key: Items_ID, Store_ID

MANAGES_FOR

<u>Emp_ID</u>	Store_ID	Position

- <u>Emp_ID</u> - auto-generated integer ID– FK from EMPLOYEES entity
- Store_ID - auto-generated integer ID– FK from STORE entity
- Position – string position of the manager.

Candidate key: Emp_ID, Store_ID
Primary key: Emp_ID, Store_ID

2.5 Sample Relation Instances

Customers

Cust_ID	Name	Phone	Email	DateCreated	DateLastTrans
50	Bob Hope	6615552485	Bobhope@gmail.com	2001-1-1	2001-5-7
51	Renee Hicks	4589854588	Dragonthing@aol.com	2005-5-5	2009-4-25
52	Scott Sheer	4176521425	Scotts@hotmail.com	2011-12-12	2012-3-4
53	Colleen Mctyre	NULL	CMcT@ct.com	2008-8-12	2009-5-9
58	Bart Simpson	NULL	NULL	2001-6-6	2007-8-25
67	Lisa Girl	6619755896	NULL	1999-4-9	2000-4-6
99	Jeremy Scott	4586895847	TheBigMan@gmail.com	2000-1-9	2001-10-10
105	Master Shake	5555555555	MixMaster@crimefighter.org	2000-8-25	2001-8-18
178	Bruce Wayne	6619872145	IamBatman@crimefighter.org	2000-1-9	2001-12-5
179	Seymoure Butes	4789582145	SButes@education.edu	2001-5-20	2001-8-18

Dependents

Bdate	Name	Relationship	Email	DateCreated	Emp_ID
1985-5-12	Brandon Jose	Husband	NULL	2011-2-30	1
2011-7-23	Dexter Jones	Son	NULL	2011-4-20	2
NULL	Katie Haitfield	Daughter	KaiteH@aol.com	2011-4-20	2
NULL	Reuben Sanders	Husband	SandersReuben@thething.com	2005-6-5	3
2004-3-24	Scott Alexander	Son	ScottA@makemoney.com	2009-8-20	4
1980-9-2	Jennifer McGuire	Wife	Jenn@thecompany.com	1990-6-8	6
1975-2-8	Amanda Hooginkiss	Wife	NULL	2008-19-8	8
2004-6-9	Betty Green	Daughter	NULL	2001-1-8	9
2004-3-18	Lester Recher	Wife	LesterR@nastynames.com	2001-6-8	9

Employees

Emp_ID	Name	SSN	Phone	Pay	Date_End	Store_ID
1	Darrel Philbin	654269856	5489659874	20	2011-2-2	854
2	Ricky Tanner	125651452	6988532587	10	1999-6-10	354
3	Susan Phillips	145969658	9856984523	15	NULL	696
4	George Scott	147589652	2586521452	42000	NULL	159
5	Erin Abernathy	256985698	5896583541	30	NULL	674
6	Ted Smith	352956587	4736593569	50000	NULL	369
7	Harry Buts	458521658	2586584763	12	NULL	778
8	Maynar Teener	256656521	2596573257	9.25	NULL	989
9	Matt Longfellow	958786548	5249868525	60000	NULL	247
10	Jerry Garcia	758965897	6521458569	52000	NULL	348
11	Lawarnce Tom	625458569	9658745632	15	2011-9-0	348
12	Dexter Robert	254125478	1111111111	12.25	NULL	854
13	Mark Nick	563258965	2225478512	8.25	NULL	854
14	Jeremy David	111111112	2356895654	16000	NULL	159
15	Luke Ted	111111144	2144544123	20000	NULL	778

Employees continue

Address	Pay_Type	Password	Email	Date_Start
258 Cumberland dr	0	1234	baldman@gmail.com	1985-4-5
1587 H st	0	Abcdef	omegaman@aol.com	1990-6-8
695 LMNOP st	0	Password	streetsmartkid@hampster.edu	1972-6-9
4521 Gold st	1	Alpha	NULL	1999-7-25
635 Number ln	0	Bottle	drinkerster@gmail.com	1998-12-20
12 S st	1	Worksucks	NULL	1989-6-8
1 wonder st	0	Password	NULL	1970-10-20
24 nice ln	0	Password	Meme585@gmail.com	2005-6-4
6144 Computer way	1	Password	thisisshort@az.com	2000-9-21
214 Q st	1	1234	govperson@gov.gov	1990-9-24
2154 Beech st	0	NULL	NULL	1989-1-20
365 Moon dr	0	NULL	NULL	1990-5-6
65412 B St	0	NULL	NULL	1998-2-5
2 Molly Way	1	NULL	NULL	2000-6-3
65 Southland Av	1	NULL	NULL	2004-9-9

Checkout

Checkout_ID	Cust_ID	Emp_ID	Date	Store_ID	Subtotal	Tax
201	50	2	2011-6-10	854	65.25	2.74
32	51	2	2011-6-9	354	115.25	5
6589	52	3	2010-8-12	696	66.52	.75
2147	99	4	2010-6-5	159	500.25	3.24
210	179	5	2009-11-5	674	41.35	1
2141	105	6	2007-4-5	369	64.25	3.25
3652	178	6	2011-12-12	778	14.25	1.2
125	58	7	2005-12-24	989	80.85	3.02

Items

Item_ID	Brand	Description	Price	Cost
12	Nabisco	Cookies	2.25	1.00
658	PhillpMorris	Cigarettes	5.00	3.00
4587	Kraft	Cheese	6.00	4.00
2365	Kellogg	Cereal	1.99	.50
84854	Quaker	Oatmeal	2.50	1.00
3521	Nabisco	Crackers	4.00	2.00
355	HomeBrand	Spagehetti	.99	.50
1566	DelMonte	Canned Fruit	.50	.10
256	Hersey	Candy	3.99	2.00
145	Kleenex	Tissues	2.99	1.00

Items (Continued)

Shape	Size HxWxD	UPC	Weight	Taxable
Oval	23x5x20	224852	22.4	1
Square	8x8x8	596543	89	0
Rectangle	6x12x3	845532	.11	0
Round	10x10x10	557858	18	1
Square	2x2x2x	556487	1	0
Round	4x4x4	254413	2	0
Round	3x3x3	698547	3	0
Square	3x3x3	411255	5.2	1
Rectangle	4x16x6	123058	52.8	0
Rectangle	3x19x4	178965	34	0

Store

Store ID	Address
854	22556 Elm St
354	820 Birch Rd
696	710 Edison Dr
159	13636 Fir St
674	14496 Maple Way
369	940 Green St
778	341 Main St
989	25459 Aspen Blvd
247	13695 Alder St
348	650 Beech St

ManagesFor

FK_Store ID	FK_Emp_ID	Position
854	1	Director
354	2	Director
696	3	Director
159	4	Director
674	5	Director
369	6	Director
778	7	Director
989	8	Director
247	9	Director
348	10	Director

Inventory

FK_Store ID	FK_Item ID	Quantity
854	12	10
854	658	10
354	1566	4
696	12	23
696	658	38
159	335	27
159	1566	31
674	4587	23
674	2365	28

Checkout Action

FK_Item_ID	Quantity	FK_Checkout_ID
12	2	210
658	2	210
84854	2	32
3521	2	32
4587	2	6589
2365	2	6589
4587	2	2147
2365	2	201
256	2	201

2.6 Query examples with SQL statements & Relational Notation

2.6.1 SQL Statements

1) Get all the customer's who was helped by employee Harry Buts

```
SELECT
      cus.Name
FROM
      [grocery].[dbo].[Customer] cus
      INNER JOIN [grocery].[dbo].[Checkout] cout on cout.FK_Cust_ID = cus.Cust_ID
      INNER JOIN [grocery].[dbo].[Employees] emps on emps.Emp_ID = cout.FK_Emp_ID
WHERE
      emps.Name = 'Harry Buts'
                              c
```

2) Get the item's name and id of every item with less a 75% mark up

```
SELECT Brand, Item_ID, cost, price
FROM [grocery].[dbo].[Items]
WHERE (cost*1.75)>price;
```

3) The total value of all the inventory by item. Group the items together and displays the total wholesake and retail amounts.

Created a view in the DB to represent the total count of each store's inventory using the 'Inventory' table. Then calculated the total retail and wholesale amounts by cross referencing the Items table.

```
CREATE VIEW inventoryview
AS
SELECT FK_Items_ID, COUNT(*) 'quantity'
FROM [grocery].[dbo].[Inventory]
GROUP BY FK_Items_ID
GO

SELECT  Brand, Item_ID, (Price * quantity) 'Retail', (Cost * quantity) 'Wholesale'
FROM [grocery].[dbo].[Items], [grocery].[dbo].[inventoryview]
WHERE Item_ID = FK_Items_ID
GO
```

4) Get all employee IDs that have a wife as dependant

```
SELECT Employees.Name, Relationship
FROM Dependents, Employees
WHERE Relationship = 'Wife'
```

5) Select all customer that have spent over $100 in a single transaction

```
SELECT
      Name
FROM
      [grocery].[dbo].[Customer] Cus
      INNER JOIN [grocery].[dbo].[Checkout] cout on Cus.Cust_ID = cout.FK_Cust_ID
WHERE cout.Subtotal > 100
```

6) List all of the addresses of grocery stores that have at least one item that retails at over $5.00.

```
SELECT
      s.Address
FROM
      [grocery].[dbo].[Store] s
      INNER JOIN [grocery].[dbo].[Inventory] inv on inv.FK_Store_ID = s.Store_ID
      INNER JOIN [grocery].[dbo].[Items] i on i.Item_ID = inv.FK_Items_ID
WHERE
      i.Price > 5
```

7) List all customers who bought more than 6 items on any signal transaction.

```
CREATE VIEW checkoutquantity
AS
SELECT [FK_Checkout_ID], COUNT(*) 'Item_Count'
  FROM [grocery].[dbo].[Checkout_Action]
  GROUP BY FK_Checkout_ID

SELECT Customer.Name
FROM
      [grocery].[dbo].[Customer]
      INNER JOIN [grocery].[dbo].[Checkout] co on co.FK_Cust_ID = Customer.Cust_ID
      INNER JOIN [grocery].[dbo].[checkoutquantity] coq on coq.FK_Checkout_ID =
co.Checkout_ID
WHERE
      coq.Item_Count > 6
```

8) List any manager who manages a store with more than 3 employees.

```
CREATE VIEW EmpsPerStore
AS
SELECT FK_Store_ID, COUNT(*) 'Emp_Count'
  FROM [grocery].[dbo].[Employees]
  GROUP BY FK_Store_ID


SELECT
      e.Name
FROM
      [grocery].[dbo].[Employees] e
      INNER JOIN [grocery].[dbo].[Manages_For] mf on mf.FK_Emp_ID = e.Emp_ID
      INNER JOIN [grocery].[dbo].[Store] s on s.Store_ID = mf.FK_Store_ID
      INNER JOIN [grocery].[dbo].[EmpsPerStore] eps on eps.FK_Store_ID = s.Store_ID
WHERE
      eps.Emp_Count > 4
```

2.6.2 Relational Notation

1) Get all the customer's who was helped by employee Harry Buts

$$\pi_{c.Name}\left(\sigma_{emps.Name = \text{'Harry Buts'}}\left(\underset{c}{Customer} * Checkout * \underset{emps}{Employees}\right)\right)$$

$$\{ c \mid Customer(c) \wedge (\exists c)(\exists e)(Checkout(c) \wedge Employees(e) \wedge e.name = \text{'Harry But'} \wedge c.CUST\_ID = checkout.FK\_CUST\_ID \wedge checkout.FK\_Emp\_ID = e.Emp\_ID) \}$$

$$\{ \langle -, n, -, -, -, - \rangle \mid Customer(-, n, -, -, -, -) \wedge (\exists CID)(\exists emp\_ID)(Checkout(cid, -, -, -, -, -, -) \wedge Employees(empID, n, -, -, -, -, -, -, -, -) \wedge e.name = \text{'Harry Buts'}) \}$$

2) Get the item's name and id of every item with less a 75% mark up

$$\pi_{i.*}\left(\sigma_{(i.cost \times 1.75) > i.price}\left(\underset{i}{Items}\right)\right)$$

$$\{ i \mid Items(i) \wedge (i.cost \times 1.75) > i.price \}$$

$$\{ \langle Item\_ID, b, d, p, c, s, s_3, u, w, T \rangle \mid Items(item\_ID, b, d, p, c, s, s_3, u, w, t) \wedge (c \times 1.75) > p \}$$

3) The total value of all the inventory by item. Group the items together and displays the total wholesake and ~~retial~~ retail amounts.

$$\sigma_{it.Items\_ID \wedge i.FK\_Items\_ID}\left(\left[\pi_{FK\_Items\_ID,\ Count(*)\ \text{'quantity'}}\left(\sigma_{Group\ By\ i.FK\_Items\_ID}\left(\underset{i}{Inventory}\right)\right)\right] \times \underset{it}{Items}\right)\ \pi_{i.brand,\ i.Item\_ID,\ (i.price \times i.quantity),\ (i.cost \times i.quantity)}$$

4) Get all employee IDs that have a wife as dependant

$$\pi_{e.name,\ d.Relationship}\left(\sigma_{d.Relationship = \text{'Wife'}}\left(\overset{e}{Employees} * \overset{d}{Dependents}\right)\right)$$

$$\{ e \mid Employees(e) \land \exists (d)(Dependent(d) \land d.Relationship = \text{'Wife'} \land e.emp\_ID = d.Emp\_ID)\}$$

$$\{\langle -, n, -, -, -, -, -, -, - \rangle \mid Employees(Emp\_ID, -, -, -, -, -, -, -, -) \land \exists (FK\_emp\_ID)(Dependents(FK\_emp\_ID, -, -, r, -, -) \land r = \text{'Wife'})\}$$

5) Select all customer that have spent over $100 in a single transaction

$$\pi_{c.name}\left(\sigma_{ch.Subtotal > 100}\left(\overset{c}{Customer} * \overset{ch}{Checkout}\right)\right)$$

$$\{c \mid Customer(c) \land \exists (ch)(Checkout(ch) \land c.Cust\_ID = ch.FK\_Cust\_ID)\}$$

$$\{\langle -, n, -, -, -, - \rangle \mid Customer(Cust\_ID, -, -, -, -, -) \land (\exists FK\_cust\_ID)(Checkout(-, -, S, -, -, FK\_cust\_ID, -) \land S > 100)\}$$

6) List all of the addresses of grocery stores that have at least one item that retails at over $5.00.

$$\pi_{s.Address}\left(\sigma_{i.price > 5}\left(\overset{s}{Store} * \overset{inv}{Inventory} * \overset{i}{Items}\right)\right)$$

$$\{s \mid Store(s) \land \exists (inv) \exists (i)(Inventory(inv) \land Items(i) \land s.Store\_ID = inv.FK\_Store\_ID \land inv.FK\_ItemsID = i.ItemsID)\}$$

$$\{\langle -, a \rangle \mid Store(Id, -) \land (\exists ID_1)(\exists ID_2)(Inventory(-, ID, ID_2) \land Items(ID_1, -, -, P, -, -, -, -, -, -, -) \land$$

7) List all customers who bought more than 6 items on any signal transaction.

$$\pi_{c.Name}\left(\sigma_{Item\_Count > 6}\left(\left(Customer^{c} * Checkout * \left(\pi_{ca.FK\_checkout\_ID,\ count(*)\ 'Item\_Count'}\left(\sigma\left(Checkout\_Action^{ca}\right)\ Group\ By\ ca.FK\_Checkout\_ID\right)\right)\right)\right)\right)$$

8) List any manager who manages a store with more than 3 employees.

$$\pi_{e.Name}\left(\sigma_{Emp\_Count > 4}\left(\left(Manages\_For^{m} * Store^{s} * \left(\pi_{e.FK\_Store\_ID,\ Count(*)\ 'Emp\_Count'}\left(\sigma\left(Employees^{e}\right)\ Group\ By\ FK\_Store\_ID\right)\right)\right)\right)\right)$$

Phase III: Implementation of Relational Database

3.1 Normalization of Your Relations

While attempting to normalize this database there are many forms that must be considered. The forms used in this application are First normal form, second normal form, third normal form and Boyce-Codd Normal form. There are more normal form but only the ones listed here are considered.

First normal form is a relation in a basic relational model that is generally part of the formal definition. The values in this form must be an atomic value and a set of values or tuple is not allowed. While utilizing this method simplicity must be taken into account in order to simplify the database and reduce memory usage. Remove nested relations and move them into a new relation if there are values that are not atomic.

The idea behind the second normalization form is full functional dependency. For this to remain true, X -> Y at all times. If at any time you are able to remove any attribute of X or Y and have the dependency remain true then full functional dependency is not happening. Not all databases are in second normal form and if not it can become second normalized by splitting into many second normalized form relations based on the dependencies of the primary key and the non-primary attributes in which they are fully functional dependent.

Transitive dependency is a functional dependency in a relation schema where X -> Y. This is called third normal form. A relation schema is said to be in third normal form if it follows all requirements of second normal form and there are no nonprime attributes that are dependent on primary keys. If it a relation is not in third normalization form it can be made into this relation by setting up a new relation that includes the non-key attributes that functionally determine other non-key attributes.

The form Boyce-Codd Normal Form is a simpler form of third normalization form. Although simpler, it is very strict. The strictness is because a relation schema is in Boyce-Codd Normal form when a nontrivial functional dependency X -> A holds in the relation schema meaning that X is a super key of that schema. This can also mean that any relation schema in Boyce-Codd form are in third normal form but relation schemas in third normal form are not always in Boyce-Codd form.

3.2 Check your relations

After checking over my relations they all appear to meet the first normalization form. This is because they all pass the test for first normalization form which is that the relation is to have no multivalued attributes or nested relations. Every relation that is present is in an atomic attribute meaning that there is no way to break down the attributes more than they already are.

The INVENTORY relation pulls information from other tables but the information pulled is a prime attribute. The relation CHECKOUT ACTION is a relation built using data populated from other tables as well but this relation is used to store the information of what customer bought what item.

The EMPLOYEE relation contains tuples that are generated by manual input while hiring a new employee. This information is only used when accessing information about an employee is needed and when a employee changes any information in this section. The same applies to CUSTOMER, ITEMS, AND DEPENDENTS. This information is input when a customer signs up or an employee requests information in this relation to be populated. Even though the information in DEPENDENTS is atomic, it is a weak entity because there is no primary key. New information for the ITEMS relations are inputted when information about an item changes or when a new item is added or old item removed. With this simplification all relations are put into First Normal Form.

Even though all the relations pass for first normal form due to their simplicity that does not mean they are all in second normal form. There are several tuples that rely on data from other relations. All of the relations seem to pass for second normal form as well. This is because the definition of second normal form is that every non-prime attribute is fully dependent on the primary key. All the relations that have attributes that are dependent on other relations would be a good candidate key and attributes that are not candidate keys are dependent on the primary key. The attributes are also transversely dependent on the primary key. For example in the CHECKOUT relation any non-prime attributes are dependent on the primary key. Without the primary key, Checkout_ID, there is no other way to pull up the tax, date, or subtitle and be sure it is the correct relation.

Being in the first, second, and third normal forms already there is a lot of data is dependent on other relations. Attributes from one tuple can often be referencing other tuples ensuring that the information is kept up to date in an attempt to isolate modification anomalies. Not only are foreign keys set up but they are all set to cascade on delete. This is because were an employee be deleted, which should not happen for many years after their termination if at all, there would be no further need to know their dependents. All relations but Checkout are CASCADE. This is because when an employee is deleted out of the system the company still wants the information on what the user purchased for later use.

None of the current relations appear to meet Boyce-Codd normal form because of how dependent the data is. For example if someone wanted to get all orders done by a certain employee and all they have is the employees name the must first get the employee number from the EMPLOYEE relation. From the EMPLOYEE relation they can then use the CHECKOUT and CHECKOUT_ACTION relation to get all orders done by the employee as well as each item that was purchased from the CHECKOUT_ACTION. This can cause modification abnormalities if the data isn’t updated correctly as time progresses. Without proper updates and SQL cleanup routines data could be deleted and not properly updated in other tables causes the foreign keys to mess up. Although hopefully rare it is a possibility that must be taken into account.

3.3 Describe the main purpose of SQL*PLUS and functionality provided by SQL*PLUS.

There are many ways to connect to and manipulate an oracle database. SQL*PLUS is one of the many ways to control this access to the database. SQL*PLUS can be used to connect to the database and execute commands that are required to create, alter, and destroy a table, sequence, or trigger.

3. 4 Describe schema objects allowed in Oracle DBMS

An Oracle schema is usually tied to a user. In this case the user, cs342, is associated with the tables that are within the schema of cs342. Oracle supports many different objects such sequences, triggers, tables, views, and functions. Currently in our tables we are using tables and foreign keys.

The syntax to create these statements is relatively consistent for all of them: *statements* value *statement* value *statement* value. The statements can be referencing a select statement, a value for inputting values into tables, or specifying the specific values that are desired. A basic example of a select query is SELECT “Cust_ID” FROM TMJ_CUSTOMER. This would get the customer’s id from the tmj_customer table.

3.5. List its relation schema

All current relations are within the schema cs342. Because of this all syntax is made simpler. Shown below are the responses from SELECT and DESC queries done on each database.

SELECT * FROM TMJ_CHECKOUT;

```
SQL> SELECT * FROM TMJ_CHECKOUT;

Checkout_ID    Cust_ID Date        Store_ID   Subtotal        Tax Employee_ID
----------- ---------- --------- ---------- ---------- ---------- -----------
        201         50 10-JUN-11        854      65.25       2.74           2
         32         51 09-JUN-11        354     115.25          5           2
       6589         52 12-AUG-10        696      66.52        .75           3
       2147         99 05-JUN-10        159     500.25       3.24           4
        210        179 05-NOV-09        674      41.35          1           5
       2141        105 05-APR-07        369      64.25       3.25           6
       3652        178 12-DEC-11        778      14.25        1.2           6
        125         58 24-DEC-05        989      80.85       3.02           7
```

DESC TMJ_CHECKOUT;

```
SQL> DESC TMJ_CHECKOUT
 Name                                      Null?    Type
 ----------------------------------------- -------- ---------------

 Checkout_ID                               NOT NULL NUMBER(6)
 Cust_ID                                   NOT NULL NUMBER(6)
 Date                                      NOT NULL DATE
 Store_ID                                  NOT NULL NUMBER(3)
 Subtotal                                  NOT NULL NUMBER(5,2)
 Tax                                       NOT NULL NUMBER(5,2)
 Employee_ID                               NOT NULL NUMBER(7)
```

SELECT * FROM TMJ_CHECKOUT_ACTION;

```
SQL> SELECT * FROM TMJ_CHECKOUT_ACTION;

Checkout_ID    Item_ID
----------- ----------
         32       3521
         32      84854
        201        256
        201       2365
        210         12
        210        658
       2147       4587
       6589       2365
       6589       4587
```

DESC TMJ_CHECKOUT_ACTION;

```
SQL> DESC TMJ_CHECKOUT_ACTION
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------

 Checkout_ID                               NOT NULL NUMBER(7)
 Item_ID                                   NOT NULL NUMBER(7)
```

SELECT * FROM TMJ_CUSTOMER;

```
   Cust_ID
----------
Name
------------------------

     Phone
----------
Email
------------------------

Date_Join
---------
Dragonthing@aol.com
05-MAY-05

   Cust_ID
----------
Name
------------------------

     Phone
----------
Email
------------------------
```

DESC TMJ_CUSTOMER;

```
SQL> DESC TMJ_CUSTOMER
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------

 Cust_ID                                   NOT NULL NUMBER(7)
 Name                                      NOT NULL VARCHAR2(128)
 Phone                                              NUMBER(10)
 Email                                              VARCHAR2(128)
 Date_Joined                               NOT NULL DATE
```

SELECT * FROM TMJ_DEPENDANTS;

```
DateCreat      Emp_ID
--------- ----------
KatieH@aol.com
20-APR-11          2


Bdate
---------
Name
---------------------------

Relationship
---------------------------

Email
---------------------------

DateCreat      Emp_ID
--------- ----------
08-FEB-75
Amanda Hooginkiss
Wife

Bdate
---------
```

DESC TMJ_DEPENDANTS;

```
SQL> DESC TMJ_DEPENDANTS
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------

 Bdate                                              DATE
 Name                                      NOT NULL VARCHAR2(128)
 Relationship                              NOT NULL VARCHAR2(128)
 Email                                              VARCHAR2(128)
 DateCreated                               NOT NULL DATE
 Emp_ID                                    NOT NULL NUMBER(7)
```

SELECT * FROM TMJ_INVENTORY;

```
SQL> SELECT * FROM TMJ_INVENTORY;

  Store_ID    Item_ID   Quantity
---------- ---------- ----------
       854         12         10
       854        658         10
       354       1566          4
       696         12         23
       696        658         38
       159        335         27
       159       1566         31
       674       4587         23
       674       2365          0
```

DESC TMJ_INVENTORY;

```
SQL> DESC TMJ_INVENTORY
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------

 Store_ID                                  NOT NULL NUMBER(3)
 Item_ID                                   NOT NULL NUMBER(7)
 Quantity                                  NOT NULL NUMBER(7)
```

DESC TMJ_EMPLOYEES;

```
SQL> DESC TMJ_EMPLOYEES
 Name                                      Null?    Type
 ----------------------------------------- -------- --------------------

 Emp_ID                                    NOT NULL NUMBER(7)
 EmpName                                   NOT NULL VARCHAR2(128)
 SSN                                       NOT NULL NUMBER(9)
 Phone                                              NUMBER(10)
 StoreRef_ID                               NOT NULL NUMBER(3)
 Address                                            VARCHAR2(128)
 PayType                                   NOT NULL NUMBER(1)
 Password                                           VARCHAR2(128)
 Email                                              VARCHAR2(128)
 Date_start                                NOT NULL DATE
 Date_end                                           DATE
 Pay                                       NOT NULL NUMBER(9)
```

SELECT * FROM TMJ_EMPLOYEES;

```
   PayType
----------
Password
--------------------------------------------------------------------------------
Email
--------------------------------------------------------------------------------
Date_star Date_end         Pay
--------- --------- ----------
NULL
    Emp_ID
----------
EmpName
--------------------------------------------------------------------------------
       SSN      Phone StoreRef_ID
---------- ---------- -----------
Address
--------------------------------------------------------------------------------
   PayType
----------
Password
--------------------------------------------------------------------------------
Email
--------------------------------------------------------------------------------
Date_star Date_end         Pay
--------- --------- ----------
01-JAN-01 01-JAN-01      20000
    Emp_ID
----------
EmpName
--------------------------------------------------------------------------------
       SSN      Phone StoreRef_ID
---------- ---------- -----------
Address
--------------------------------------------------------------------------------
   PayType
----------
Password
--------------------------------------------------------------------------------
Email
--------------------------------------------------------------------------------
Date_star Date_end         Pay
--------- --------- ----------

14 rows selected.
```

SELECT * FROM TMJ_ITEMS; (not all items can be shown in this view)

```
---------- ----------
      1566 HomeBrand
Spagehetti
       .99         .5 Round                           3x3x3      698547
         3          0


   Item_ID Brand
---------- --------------------------------
Description
----------------------------------------------------------------------------

     Price       Cost Shape                           Size       UPC
---------- ---------- ------------------------------- ---------- ---------
    Weight    Taxable
---------- ----------
       335 DelMonte
Canned Fruit
        .5         .1 Square                          3x3x3      411255
       5.2          1


   Item_ID Brand
---------- --------------------------------
Description
----------------------------------------------------------------------------

     Price       Cost Shape                           Size       UPC
---------- ---------- ------------------------------- ---------- ---------
    Weight    Taxable
---------- ----------
       256 Hersey
Candy
      3.99          2 Rectangle                       4x16x6     123058
      52.8          0


   Item_ID Brand
---------- --------------------------------
Description
----------------------------------------------------------------------------
```

DESC TMJ_ITEMS;

```
SQL> DESC TMJ_ITEMS
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------

 Item_ID                                   NOT NULL NUMBER(7)
 Brand                                     NOT NULL VARCHAR2(32)
 Description                               NOT NULL VARCHAR2(128)
 Price                                     NOT NULL NUMBER(5,2)
 Cost                                      NOT NULL NUMBER(5,2)
 Shape                                              VARCHAR2(32)
 Size                                               VARCHAR2(10)
 UPC                                       NOT NULL VARCHAR2(9)
 Weight                                             NUMBER(4,2)
 Taxable                                   NOT NULL NUMBER(1)
```

SELECT * FROM TMJ_MANAGESFOR;

StoreRef_ID	EmpRef_ID	Position
854	1	Director
354	2	Director
696	3	Director
159	4	Director
674	5	Director
369	6	Director
778	7	Director
989	8	Director
247	9	Director

DESC TMJ_MANAGESFOR;

```
SQL> DESC TMJ_MANAGESFOR
 Name                                      Null?    Type
 ----------------------------------------- -------- ---------------

 StoreRef_ID                               NOT NULL NUMBER(3)
 EmpRef_ID                                 NOT NULL NUMBER(7)
 Position                                  NOT NULL VARCHAR2(32)
```

SELECT * FROM TMJ_STORE; (not all items can be shown in this view)

```
       854
22556 Elm St

       354
820 Birch Rd

       696
710 Edison Dr


  Store_ID
----------
Address
------------------------

       159
13636 Fir St

       674
14496 Maple Way

       369
940 Green St


  Store_ID
----------
Address
------------------------

       778
341 Main St

       989
25459 Aspen Blvd

       247
13695 Alder St


  Store_ID
----------
Address
------------------------

       348
650 Beech St
```

DESC TMJ_STORE;

```
SQL> DESC TMJ_STORE
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------

 Store_ID                                  NOT NULL NUMBER(3)
 Address                                   NOT NULL VARCHAR2(128)
```

3.6. Write queries designed in previous phase in SQL language.

```
Save the SQL statement in a file. Run each if the files to generate the report.
       The following SQL features will be used in some of your queries (if not, add more new
        queries to the previous phase):
                - IS [not] NULL,
                -[NOT] EXISTS,
                - GROUP BY and Having,
                - aggregate functions,
                - sub-select statement,
                - create a new table from existing table(s)
                  using CREATE TABLE .... AS SELECT ...
                - outer join.
```

1) Get all the customer's who was helped by employee Harry Buts

```
SELECT
      cus.Name
FROM
      [grocery].[dbo].[Customer] cus
      INNER JOIN [grocery].[dbo].[Checkout] cout on cout.FK_Cust_ID = cus.Cust_ID
      INNER JOIN [grocery].[dbo].[Employees] emps on emps.Emp_ID = cout.FK_Emp_ID
WHERE
      emps.Name = 'Harry Buts'
```

Query

```
SELECT
      cus."CustName"
FROM
      TMJ_CUSTOMER cus
      INNER JOIN TMJ_CHECKOUT cout on cout."CustRef_ID" = cus."Cust_ID"
      INNER JOIN TMJ_EMPLOYEES emps on emps."Emp_ID" = cout."EmpRef_ID"
WHERE
      emps."EmpName" = 'Harry Buts';
```

Results: Bart Simpson

2) Get the item's name and id of every item with less a 75% mark up

```
SELECT Brand, Item_ID, cost, price
FROM [grocery].[dbo].[Items]
WHERE (cost*1.75)>price;
```

Query

```
SELECT "Brand", "Item_ID", "Cost", "Price"
FROM TMJ_ITEMS
WHERE ("Cost"*1.75)>"Price";
```

Results

```
PhillpMorris      658   3     5
Kraft 4587  4     6
```

3) The total value of all the inventory by item. Group the items together and displays the total wholesake and retail amounts.
Created a view in the DB to represent the total count of each store's inventory using the 'Inventory' table. Then calculated the total retail and wholesale amounts by cross referencing the Items table.

```
CREATE VIEW inventoryview
AS
SELECT FK_Items_ID, COUNT(*) 'quantity'
FROM [grocery].[dbo].[Inventory]
GROUP BY FK_Items_ID
GO

Create View
SELECT
CS342.TMJ_INVENTORY."ItemRef_ID",
Count (*) "Quantity"
FROM
CS342.TMJ_INVENTORY
GROUP BY
CS342.TMJ_INVENTORY."ItemRef_ID"


SELECT  Brand, Item_ID, (Price * quantity) 'Retail', (Cost * quantity) 'Wholesale'
FROM [grocery].[dbo].[Items], [grocery].[dbo].[inventoryview]
WHERE Item_ID = FK_Items_ID
GO
```

<u>Query</u>

SELECT "Brand", "Item_ID", ("Price" * "Quantity") "Retail", ("Cost" * "Quantity") "Wholesale"
FROM TMJ_ITEMS, TMJ_INVENTORYVIEW
WHERE TMJ_ITEMS."Item_ID" = TMJ_INVENTORYVIEW."ItemRef_ID";

Result

DelMonte	335	0.5	0.1
PhillpMorris	658	10	6
HomeBrand	1566	1.98	1
Kellogg 2365	1.99	0.5	
Kraft 4587	6	4	
Nabisco12	4.5	2	

4) Get all employee IDs that have a wife as dependent

```
SELECT Employees.Name, Relationship
FROM Dependents, Employees
WHERE Relationship = 'Wife'

Orcale Command

SELECT "Emp_ID"
FROM TMJ_DEPENDANTS,TMJ_EMPLOYEES
WHERE "Relationship" = 'Wife' AND "EmpRef_ID"="Emp_ID";
```

```
Results
```

6
2
8

5) Select all customer that have spent over $100 in a single transaction

```
SELECT
       Name
FROM
       [grocery].[dbo].[Customer] Cus
       INNER JOIN [grocery].[dbo].[Checkout] cout on Cus.Cust_ID = cout.FK_Cust_ID
WHERE cout.Subtotal > 100
```

Query

```
SELECT
       "CustName"
FROM
       TMJ_CUSTOMER, TMJ_CHECKOUT
WHERE ("Subtotal" + "Tax") > 100 AND "Cust_ID" = "CustRef_ID";
```

Results

Renee Hicks
Jeremy Scott

6) List all of the addresses of grocery stores that have at least one item that retails at over $5.00.

```
SELECT
       s.Address
FROM
       [grocery].[dbo].[Store] s
       INNER JOIN [grocery].[dbo].[Inventory] inv on inv.FK_Store_ID = s.Store_ID
       INNER JOIN [grocery].[dbo].[Items] i on i.Item_ID = inv.FK_Items_ID
WHERE
       i.Price > 5
```

Query

```
SELECT
       s."Address"
FROM
       TMJ_STORE s
       INNER JOIN TMJ_INVENTORY inv on inv."StoreRef_ID" = s."Store_ID"
       INNER JOIN TMJ_ITEMS i on i."Item_ID" = inv."ItemRef_ID"
WHERE
       i."Price" > 5;
```

7) List all customers who bought more than 6 items on any signal transaction.

```
CREATE VIEW checkoutquantity
AS
SELECT [FK_Checkout_ID], COUNT(*) 'Item_Count'
  FROM [grocery].[dbo].[Checkout_Action]
  GROUP BY FK_Checkout_ID

SELECT Customer.Name
FROM
      [grocery].[dbo].[Customer]
      INNER JOIN [grocery].[dbo].[Checkout] co on co.FK_Cust_ID = Customer.Cust_ID
      INNER JOIN [grocery].[dbo].[checkoutquantity] coq on coq.FK_Checkout_ID =
co.Checkout_ID
WHERE
      coq.Item_Count > 6
```

Create the View

```
SELECT "CheckoutRef_ID", COUNT(*) "Item_Count"
 FROM TMJ_CHECKOUT_ACTION
 GROUP BY "CheckoutRef_ID"
```

Query

```
SELECT TMJ_CUSTOMER."CustName"
FROM
      TMJ_CUSTOMER
      INNER JOIN TMJ_CHECKOUT co on co."CustRef_ID" = TMJ_CUSTOMER."Cust_ID"
      INNER JOIN TMJ_CHECKOUTQUANTITY coq on coq."CheckoutRef_ID" = co."Checkout_ID"
WHERE
      coq."Item_Count" > 1;
```

8) List any manager who manages a store with more than 3 employees.

```
CREATE VIEW EmpsPerStore
AS
SELECT FK_Store_ID, COUNT(*) 'Emp_Count'
  FROM [grocery].[dbo].[Employees]
  GROUP BY FK_Store_ID
SELECT
      e.Name
FROM
      [grocery].[dbo].[Employees] e
      INNER JOIN [grocery].[dbo].[Manages_For] mf on mf.FK_Emp_ID = e.Emp_ID
      INNER JOIN [grocery].[dbo].[Store] s on s.Store_ID = mf.FK_Store_ID
      INNER JOIN [grocery].[dbo].[EmpsPerStore] eps on eps.FK_Store_ID = s.Store_ID
WHERE
      eps.Emp_Count > 4
```

Create the View

```
SELECT "StoreRef_ID", COUNT(*) "Emp_Count"
  FROM CS342.TMJ_EMPLOYEES
  GROUP BY "StoreRef_ID"
```

<u>Query</u>

```
SELECT e."EmpName"
FROM   TMJ_EMPLOYEES e
INNER JOIN TMJ_MANAGESFOR mf ON mf."EmpRef_ID" =e."Emp_ID"
INNER JOIN TMJ_STORE s on s."Store_ID" = mf."StoreRef_ID"
INNER JOIN TMJ_EMPSPERSTORE eps on eps."StoreRef_ID" = s."Store_ID";
WHERE  eps."Emp_Count" > 4;
```

9) Create a duplicate table of an existent table

```
CREATE TABLE TMJ_CUSTOMERSDUP AS (SELECT *
         FROM TMJ_CUSTOMERS;
```

3.7. Data Loader

There are several different types of methods to load data into the database. Different people will end up using different DMBS. Our team used Navicat lite as a DMBS because the interface to interact with the database system was simple and clean ensuring ease of use. Creating the tables can be done using the command line or the GUI interface. The same goes for creating triggers or sequences for use in databases. Using this DMBS we can also export the SQL script to be used latter to build other databases replicating the current one.

```
-- Description of Java DataLoader Program. Add additional features
   into the program to make it more user friendly.
```

All of these DBM Systems use the same basics ideas for modifying values. For example they use INTERT INTO and SELECT statements , however, they are being used behind the scenes to simplify the programmer's and database manager's life. The would be helpful for anyone trying to manage a database small or large as the tools are simplified and often use a more generic form of English rather than the strict the SQL calls.

```
-- Document the features that you have added into the original
   DataLoader Program.
```

A free program found on the internet to help interact with database is the java data loader program. This program allows a user to load data into the database using CSV, or XML files. This speeds up the process of populating the original database with values. Granted the files must be in the correct setup to ensure an accurate transfer but once they are it is much faster to load a database with values this way rather than by hand.

Currently our group has no plans to use the java Dataloader program due to its limitations. There are many other free programs out there that allow greater access and manipulation of the data base. TOAD, Navicat lite and SQL Express Management Studio are examples of free DBMS. The DBMS that this group is using is Navicat lite. We are using this program because it allows the user to connect to the database and create tables, sequences, triggers, and even input or import information. Besides importing and data manipulation Navicat also allows for data exporting into a CSV file or a basic text file.

Phase IV: Stored Procedure

4.1 Write the following stored procedure or function in Oracle PL/SQL:

For The Grocery Store Project a Procedure to insert new rows into TMJ_CHECKOUT was created. The syntax is:

```
CREATE OR REPLACE
PROCEDURE "InsertCheckoutP" (Cust_ID IN NUMBER, Store_ID IN NUMBER, SubTotal IN NUMBER, Tax
IN NUMBER, Emp_ID IN NUMBER)
AS
BEGIN
        -- routine body goes here, e.g.
        -- DBMS_OUTPUT.PUT_LINE('Navicat for Oracle');
INSERT INTO TMJ_CHECKOUT VALUES       ('22',Cust_ID,
        SYSDATE(),Store_ID,SubTotal,Tax,Emp_ID);
END;
```

A stored procedure was created to delete a customer's orders history. This can be used when a customer is requesting a return. This way the item is taken off the order and if the customer returns the item they cannot try to return it again. The syntax:

```
CREATE OR REPLACE
PROCEDURE "TMJ_DeleteCheckout" (pri_key IN NUMBER)
AS
BEGIN
        -- routine body goes here, e.g.
        -- DBMS_OUTPUT.PUT_LINE('Navicat for Oracle');
DELETE TMJ_CHECKOUT WHERE "Checkout_ID"=pri_key;
END;
```

4.2 Common Features in Oracle PL/SQL and MS Trans-SQL

SQL components are split into many different subtypes. The subtypes are: clauses, expressions, predicates, queries and statements. Breaking it down further clauses are the components of statements and queries. Expressions are statements which can produce tables that can also be filled with data using the expressions. In order to specify conditions that SQL can use and evaluate, also known as truth values, predicates are used.

Those are the more complicated structures of SQL. The basics are the Queries and statements. Statements can effect data and schema or can control transaction, flow, connections, sessions while queries simply get data based on specified criteria and are also the most import part of any database.

Subprogram, also known as procedures and functions, are used within a database to simply the calling procedures from the front end. They also use the CPU on the server and return just what the user is requesting instead of returning all the tables and forcing the users CPU to process the data.

The benefits of calling a stored subprogram over sending data from the interacting software is simplicity and ensuring correct data is used. Calling a subprogram also ensures that the information needed is correct preventing sending wrong data from the front end user to the database.

4.3 Oracle PL/SQL

A stored procedure is useful within any database application as it simplifies the programmer's job and ensures fewer mistakes while parsing information because the information is parsed on the server side. The syntax on a stored procedure is:

```
CREATE OR REPLACE
PROCEDURE <procedure name> (<value name> IN NUMBER, <value name> IN OUT NUMBER)
AS
BEGIN
        -- routine body goes here, e.g.
END;
```

A stored function is very similar to a store procedure expect that a function will return a value and a procedure modifies data without returning anything. Also like a procedure the work is done on the server side instead of bogging down the CPU of the end user. The stored function syntax is:

```
CREATE OR REPLACE
FUNCTION <Name> RETURN NUMBER (<value name> IN NUMBER, <value name> IN OUT NUMBER)
AS
BEGIN
        -- routine body goes here, e.g.
        -- DBMS_OUTPUT.PUT_LINE
        RETURN NULL;
END;
```

Packages are objects that groups related types, items, and subprograms logically. Typically packages have a body and a specification. The body is where the cursors and subprograms are defined and the specification is the interface to the program. The syntax to create or replace a package and its body 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];

[CREATE [OR REPLACE] PACKAGE BODY package_name {IS | AS}
   [PRAGMA SERIALLY_REUSABLE;]
   [collection_type_definition ...]
```

```
   [record_type_definition ...]
   [subtype_definition ...]
   [collection_declaration ...]
   [constant_declaration ...]
   [exception_declaration ...]
   [object_declaration ...]
   [record_declaration ...]
   [variable_declaration ...]
   [cursor_body ...]
   [function_spec ...]
   [procedure_spec ...]
   [call_spec ...]
[BEGIN
   sequence_of_statements]
END [package_name];]
```

A trigger is a piece of code that is executed once a certain condition is met. For example a trigger can execute when a user adds a new row to a table. The relation CUSTOMERS has a trigger that fills in the attribute Cust_ID with an auto increment number.

```
create or replace trigger CUSTOMERS_ID_SEQ
before insert on TMJ_CUSTOMERS
for each row
begin
   select CUSTOMERS_ID_SEQ.nextval into :new.Cust=ID from dual;
end;
/
```

4.4 Oracle PL/SQL Subprogram

/*

```
Navicat Oracle Data Transfer
Oracle Client Version : 10.2.0.5.0

Source Server        : CSUB
Source Server Version : 110200
Source Host          : delphi.cs.csub.edu:1521
Source Schema        : CS342

Target Server Type    : ORACLE
Target Server Version : 110200
File Encoding        : 65001

Date: 2011-11-07 20:43:47
*/


-- ----------------------------
-- Table structure for "CS342"."TMJ_CUSTOMER"
-- ----------------------------
DROP TABLE "CS342"."TMJ_CUSTOMER";
CREATE TABLE "CS342"."TMJ_CUSTOMER" (
"Cust_ID" NUMBER(7) NOT NULL ,
"CustName" VARCHAR2(128 BYTE) NOT NULL ,
"Phone" NUMBER(10) NULL ,
"Email" VARCHAR2(128 BYTE) NULL ,
"Date_Joined" DATE NOT NULL
)
LOGGING
NOCOMPRESS
NOCACHE

;

-- ----------------------------
-- Records of TMJ_CUSTOMER
-- ----------------------------
INSERT INTO "CS342"."TMJ_CUSTOMER" VALUES ('50', 'Bob Hope', '6615552485', 'Bobhope@gmail.com',
TO_DATE('2001-01-01 00:00:00', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO "CS342"."TMJ_CUSTOMER" VALUES ('51', 'Renee Hicks', '4589854588',
'Dragonthing@aol.com', TO_DATE('2005-05-05 00:00:00', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO "CS342"."TMJ_CUSTOMER" VALUES ('52', 'Scott Sheer', '4176521425', 'Scotts@hotmail.com',
TO_DATE('2011-12-12 00:00:00', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO "CS342"."TMJ_CUSTOMER" VALUES ('53', 'Colleen Mctyre', null, 'CMcT@ct.com',
TO_DATE('2008-08-12 00:00:00', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO "CS342"."TMJ_CUSTOMER" VALUES ('58', 'Bart Simpson', null, 'NULL', TO_DATE('2001-06-
06 00:00:00', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO "CS342"."TMJ_CUSTOMER" VALUES ('67', 'Lisa Girl', '6619755896', 'NULL',
TO_DATE('1999-04-09 00:00:00', 'YYYY-MM-DD HH24:MI:SS'));
```

```
INSERT INTO "CS342"."TMJ_CUSTOMER" VALUES ('99', 'Jeremy Scott', '4586895847',
'TheBigMan@gmail.com', TO_DATE('2000-01-09 00:00:00', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO "CS342"."TMJ_CUSTOMER" VALUES ('105', 'Master Shake', '5555555555',
'MixMaster@crimefighter.org', TO_DATE('2000-08-25 00:00:00', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO "CS342"."TMJ_CUSTOMER" VALUES ('178', 'Bruce Wayne', '6619872145',
'IamBatman@crimefighter.org', TO_DATE('2000-01-09 00:00:00', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO "CS342"."TMJ_CUSTOMER" VALUES ('179', 'Seymoure Butes', '4789582145',
'SButes@education.edu', TO_DATE('1-01-01 00:00:00', 'YYYY-MM-DD HH24:MI:SS'));

-- ----------------------------
-- Indexes structure for table TMJ_CUSTOMER
-- ----------------------------

-- ----------------------------
-- Triggers structure for table "CS342"."TMJ_CUSTOMER"
-- ----------------------------
CREATE OR REPLACE TRIGGER "CS342"."TMJ_CUSTOMER_UPDATE" BEFORE INSERT ON
"CS342"."TMJ_CUSTOMER" REFERENCING OLD AS "OLD" NEW AS "NEW" FOR EACH ROW ENABLE
BEGIN
  SELECT TMJ_CUSTOMER_ID_SEQ.nextval
values :new."Cust_ID" from dual;
END;;

-- ----------------------------
-- Checks structure for table "CS342"."TMJ_CUSTOMER"
-- ----------------------------
ALTER TABLE "CS342"."TMJ_CUSTOMER" ADD CHECK ("CustName" IS NOT NULL);
ALTER TABLE "CS342"."TMJ_CUSTOMER" ADD CHECK ("Date_Joined" IS NOT NULL);

-- ----------------------------
-- Primary Key structure for table "CS342"."TMJ_CUSTOMER"
-- ----------------------------
ALTER TABLE "CS342"."TMJ_CUSTOMER" ADD PRIMARY KEY ("Cust_ID");

/*
Navicat Oracle Data Transfer
Oracle Client Version : 10.2.0.5.0

Source Server         : CSUB
Source Server Version : 110200
Source Host           : delphi.cs.csub.edu:1521
Source Schema         : CS342

Target Server Type    : ORACLE
Target Server Version : 110200
File Encoding         : 65001

Date: 2011-11-07 20:44:03
```

```
*/


-- ----------------------------
-- Table structure for "CS342"."TMJ_ITEMS"
-- ----------------------------
DROP TABLE "CS342"."TMJ_ITEMS";
CREATE TABLE "CS342"."TMJ_ITEMS" (
"Item_ID" NUMBER(7) NOT NULL ,
"Brand" VARCHAR2(32 BYTE) NOT NULL ,
"Description" VARCHAR2(128 BYTE) NOT NULL ,
"Price" NUMBER(5,2) NOT NULL ,
"Cost" NUMBER(5,2) NOT NULL ,
"Shape" VARCHAR2(32 BYTE) DEFAULT NULL  NULL ,
"Size" VARCHAR2(10 BYTE) DEFAULT NULL  NULL ,
"UPC" VARCHAR2(9 BYTE) NOT NULL ,
"Weight" NUMBER(4,2) DEFAULT NULL  NULL ,
"Taxable" NUMBER(1) DEFAULT 1  NOT NULL
)
LOGGING
NOCOMPRESS
NOCACHE

;

-- ----------------------------
-- Records of TMJ_ITEMS
-- ----------------------------
INSERT INTO "CS342"."TMJ_ITEMS" VALUES ('12', 'Nabisco', 'Cookies', '2.25', '1', 'Oval', '23x5x20', '224852',
'22.40', '1');
INSERT INTO "CS342"."TMJ_ITEMS" VALUES ('658', 'PhillpMorris', 'Cigarettes', '5', '3', 'Square', '8x8x8',
'596543', '89', '0');
INSERT INTO "CS342"."TMJ_ITEMS" VALUES ('4587', 'Kraft', 'Cheese', '6', '4', 'Rectangle', '6x12x3', '845532',
'0.11', '0');
INSERT INTO "CS342"."TMJ_ITEMS" VALUES ('2365', 'Kellogg', 'Cereal', '1.99', '0.50', 'Round', '10x10x10',
'557858', '18', '1');
INSERT INTO "CS342"."TMJ_ITEMS" VALUES ('84854', 'Quaker', 'Oatmeal', '2.50', '1', 'Square', '2x2x2x',
'556487', '1', '0');
INSERT INTO "CS342"."TMJ_ITEMS" VALUES ('3521', 'Nabisco', 'Crackers', '4', '2', 'Round', '4x4x4', '254413',
'2', '0');
INSERT INTO "CS342"."TMJ_ITEMS" VALUES ('1566', 'HomeBrand', 'Spagehetti', '0.99', '0.50', 'Round',
'3x3x3', '698547', '3', '0');
INSERT INTO "CS342"."TMJ_ITEMS" VALUES ('335', 'DelMonte', 'Canned Fruit', '0.50', '0.10', 'Square',
'3x3x3', '411255', '5.20', '1');
INSERT INTO "CS342"."TMJ_ITEMS" VALUES ('256', 'Hersey', 'Candy', '3.99', '2', 'Rectangle', '4x16x6',
'123058', '52.80', '0');
INSERT INTO "CS342"."TMJ_ITEMS" VALUES ('145', 'Kleenex', 'Tissues', '2.99', '1', 'Rectangle', '3x19x4',
'178965', '34', '0');
```

```
-- ----------------------------
-- Indexes structure for table TMJ_ITEMS
-- ----------------------------
CREATE UNIQUE INDEX "CS342"."Items_index"
ON "CS342"."TMJ_ITEMS" ("Item_ID" ASC, "UPC" ASC)
LOGGING
VISIBLE;

-- ----------------------------
-- Triggers structure for table "CS342"."TMJ_ITEMS"
-- ----------------------------
CREATE OR REPLACE TRIGGER "CS342"."TMJ_ITEMDELETE" AFTER INSERT OR DELETE OR
UPDATE OF "Item_ID" ON "CS342"."TMJ_ITEMS" REFERENCING OLD AS "OLD" NEW AS "NEW" FOR
EACH ROW ENABLE
BEGIN
DECLARE
NUMBER X;
SELECT "Item_ID" INTO X
FROM TMJ_ITEMS
INSERT INTO TMJ_VALUES VALUES ( X,new."Item_ID")
END;;

-- ----------------------------
-- Checks structure for table "CS342"."TMJ_ITEMS"
-- ----------------------------
ALTER TABLE "CS342"."TMJ_ITEMS" ADD CHECK ("Item_ID" IS NOT NULL);
ALTER TABLE "CS342"."TMJ_ITEMS" ADD CHECK ("Brand" IS NOT NULL);
ALTER TABLE "CS342"."TMJ_ITEMS" ADD CHECK ("Description" IS NOT NULL);
ALTER TABLE "CS342"."TMJ_ITEMS" ADD CHECK ("Price" IS NOT NULL);
ALTER TABLE "CS342"."TMJ_ITEMS" ADD CHECK ("Cost" IS NOT NULL);
ALTER TABLE "CS342"."TMJ_ITEMS" ADD CHECK ("Taxable" IS NOT NULL);
ALTER TABLE "CS342"."TMJ_ITEMS" ADD CHECK ("UPC" IS NOT NULL);

-- ----------------------------
-- Primary Key structure for table "CS342"."TMJ_ITEMS"
-- ----------------------------
ALTER TABLE "CS342"."TMJ_ITEMS" ADD PRIMARY KEY ("Item_ID");

/*
Navicat Oracle Data Transfer
Oracle Client Version : 10.2.0.5.0

Source Server         : CSUB
Source Server Version : 110200
Source Host           : delphi.cs.csub.edu:1521
Source Schema         : CS342
```

```
Target Server Type    : ORACLE
Target Server Version : 110200
File Encoding         : 65001

Date: 2011-11-07 20:44:16
*/


-- ----------------------------
-- Table structure for "CS342"."TMJ_STORE"
-- ----------------------------
DROP TABLE "CS342"."TMJ_STORE";
CREATE TABLE "CS342"."TMJ_STORE" (
"Store_ID" NUMBER(3) NOT NULL ,
"Address" VARCHAR2(128 BYTE) NOT NULL
)
LOGGING
NOCOMPRESS
NOCACHE

;

-- ----------------------------
-- Records of TMJ_STORE
-- ----------------------------
INSERT INTO "CS342"."TMJ_STORE" VALUES ('159', '13636 Fir St');
INSERT INTO "CS342"."TMJ_STORE" VALUES ('247', '13695 Alder St');
INSERT INTO "CS342"."TMJ_STORE" VALUES ('674', '14496 Maple Way');
INSERT INTO "CS342"."TMJ_STORE" VALUES ('854', '22556 Elm St');
INSERT INTO "CS342"."TMJ_STORE" VALUES ('989', '25459 Aspen Blvd');
INSERT INTO "CS342"."TMJ_STORE" VALUES ('778', '341 Main St');
INSERT INTO "CS342"."TMJ_STORE" VALUES ('348', '650 Beech St');
INSERT INTO "CS342"."TMJ_STORE" VALUES ('696', '710 Edison Dr');
INSERT INTO "CS342"."TMJ_STORE" VALUES ('354', '820 Birch Rd');
INSERT INTO "CS342"."TMJ_STORE" VALUES ('369', '940 Green St');

-- ----------------------------
-- Indexes structure for table TMJ_STORE
-- ----------------------------

-- ----------------------------
-- Uniques structure for table "CS342"."TMJ_STORE"
-- ----------------------------
ALTER TABLE "CS342"."TMJ_STORE" ADD UNIQUE ("Address");

-- ----------------------------
-- Checks structure for table "CS342"."TMJ_STORE"
-- ----------------------------
```

```
ALTER TABLE "CS342"."TMJ_STORE" ADD CHECK ("Store_ID" IS NOT NULL);
ALTER TABLE "CS342"."TMJ_STORE" ADD CHECK ("Address" IS NOT NULL);

-- ----------------------------
-- Primary Key structure for table "CS342"."TMJ_STORE"
-- ----------------------------
ALTER TABLE "CS342"."TMJ_STORE" ADD PRIMARY KEY ("Store_ID");

/*
Navicat Oracle Data Transfer
Oracle Client Version : 10.2.0.5.0

Source Server         : CSUB
Source Server Version : 110200
Source Host           : delphi.cs.csub.edu:1521
Source Schema         : CS342

Target Server Type    : ORACLE
Target Server Version : 110200
File Encoding         : 65001

Date: 2011-11-07 20:43:54
*/


-- ----------------------------
-- Table structure for "CS342"."TMJ_EMPLOYEES"
-- ----------------------------
DROP TABLE "CS342"."TMJ_EMPLOYEES";
CREATE TABLE "CS342"."TMJ_EMPLOYEES" (
"Emp_ID" NUMBER(7) NOT NULL ,
"EmpName" VARCHAR2(128 BYTE) NOT NULL ,
"SSN" VARCHAR2(9 BYTE) NOT NULL ,
"Phone" VARCHAR2(10 BYTE) NULL ,
"StoreRef_ID" NUMBER(3) NOT NULL ,
"Address" VARCHAR2(128 BYTE) NULL ,
"PayType" NUMBER(1) NOT NULL ,
"Password" VARCHAR2(128 BYTE) NULL ,
"Manager" NUMBER(1) NOT NULL ,
"Email" VARCHAR2(128 BYTE) NULL ,
"Date_hired" DATE DEFAULT sysdate  NULL
)
LOGGING
NOCOMPRESS
NOCACHE

;
```

```
-- ----------------------------
-- Records of TMJ_EMPLOYEES
-- ----------------------------
INSERT INTO "CS342"."TMJ_EMPLOYEES" VALUES ('1', 'Darrel Philbin', '654269856', '5489659874', '854', '258 Cumberland dr', '0', '1234', '0', 'NULL', TO_DATE('1985-04-05 00:00:00', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO "CS342"."TMJ_EMPLOYEES" VALUES ('2', 'Ricky Tanner', '125651452', '6988532587', '354', '1587 H st', '0', 'Abcdef', '0', 'omegaman@aol.com', TO_DATE('1990-06-08 00:00:00', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO "CS342"."TMJ_EMPLOYEES" VALUES ('3', 'Susan Phillips', '145969658', '9856984523', '696', '695 LMNOP st', '0', 'Password', '0', 'streetsmartkid@hampster.edu', TO_DATE('1972-06-09 00:00:00', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO "CS342"."TMJ_EMPLOYEES" VALUES ('4', 'George Scott', '147589652', '2586521452', '159', '4521 Gold st', '1', 'Alpha', '1', 'NULL', TO_DATE('1999-07-25 00:00:00', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO "CS342"."TMJ_EMPLOYEES" VALUES ('5', 'Erin Abernathy', '256985698', '5896583541', '674', '635 Number ln', '0', 'Bottle', '0', 'drinkerster@gmail.com', TO_DATE('1998-12-20 00:00:00', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO "CS342"."TMJ_EMPLOYEES" VALUES ('6', 'Ted Smith', '352956587', '4736593569', '369', '12 S st', '1', 'Worksucks', '1', 'NULL', TO_DATE('1989-06-08 00:00:00', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO "CS342"."TMJ_EMPLOYEES" VALUES ('7', 'Harry Buts', '458521658', '2586584763', '778', '1 wonder st', '0', 'Password', '0', 'NULL', TO_DATE('1970-10-20 00:00:00', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO "CS342"."TMJ_EMPLOYEES" VALUES ('8', 'Maynar Teener', '256656521', '2596573257', '989', '24 nice ln', '0', 'Password', '0', 'Meme585@gmail.com', TO_DATE('2005-06-04 00:00:00', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO "CS342"."TMJ_EMPLOYEES" VALUES ('9', 'Matt Longfellow', '958786548', '5249868525', '247', '6144 Computer way', '1', 'Password', '1', 'thisisshort@az.com', TO_DATE('2000-09-21 00:00:00', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO "CS342"."TMJ_EMPLOYEES" VALUES ('10', 'Jerry Garcia', '758965897', '6521458569', '348', '214 Q st', '1', '1234', '1', 'govperson@gov.gov', TO_DATE('1-01-01 00:00:00', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO "CS342"."TMJ_EMPLOYEES" VALUES ('11', 'Bret Roberts', '222458521', '6619724848', '674', '669 Backroad RD', '0', '1234', '0', 'NULL', TO_DATE('2011-11-06 22:00:55', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO "CS342"."TMJ_EMPLOYEES" VALUES ('12', 'Bret Roberts', '222458522', '6619724849', '674', '667 Backroad RD', '0', '1234', '0', 'NULL', TO_DATE('2011-11-06 22:01:28', 'YYYY-MM-DD HH24:MI:SS'));

-- ----------------------------
-- Indexes structure for table TMJ_EMPLOYEES
-- ----------------------------
CREATE INDEX "CS342"."Employees__index"
ON "CS342"."TMJ_EMPLOYEES" ("SSN" ASC, "StoreRef_ID" ASC)
LOGGING
VISIBLE;

-- ----------------------------
-- Uniques structure for table "CS342"."TMJ_EMPLOYEES"
-- ----------------------------
ALTER TABLE "CS342"."TMJ_EMPLOYEES" ADD UNIQUE ("SSN");

-- ----------------------------
-- Checks structure for table "CS342"."TMJ_EMPLOYEES"
```

```
-- ----------------------------
ALTER TABLE "CS342"."TMJ_EMPLOYEES" ADD CHECK ("Emp_ID" IS NOT NULL);
ALTER TABLE "CS342"."TMJ_EMPLOYEES" ADD CHECK ("EmpName" IS NOT NULL);
ALTER TABLE "CS342"."TMJ_EMPLOYEES" ADD CHECK ("SSN" IS NOT NULL);
ALTER TABLE "CS342"."TMJ_EMPLOYEES" ADD CHECK ("PayType" IS NOT NULL);
ALTER TABLE "CS342"."TMJ_EMPLOYEES" ADD CHECK ("Manager" IS NOT NULL);
ALTER TABLE "CS342"."TMJ_EMPLOYEES" ADD CHECK ("StoreRef_ID" IS NOT NULL);

-- ----------------------------
-- Primary Key structure for table "CS342"."TMJ_EMPLOYEES"
-- ----------------------------
ALTER TABLE "CS342"."TMJ_EMPLOYEES" ADD PRIMARY KEY ("Emp_ID");

-- ----------------------------
-- Foreign Key structure for table "CS342"."TMJ_EMPLOYEES"
-- ----------------------------
ALTER TABLE "CS342"."TMJ_EMPLOYEES" ADD FOREIGN KEY ("StoreRef_ID") REFERENCES
"CS342"."TMJ_STORE" ("Store_ID") ON DELETE CASCADE;

/*
Navicat Oracle Data Transfer
Oracle Client Version : 10.2.0.5.0

Source Server         : CSUB
Source Server Version : 110200
Source Host           : delphi.cs.csub.edu:1521
Source Schema         : CS342

Target Server Type    : ORACLE
Target Server Version : 110200
File Encoding         : 65001

Date: 2011-11-07 20:43:50
*/


-- ----------------------------
-- Table structure for "CS342"."TMJ_DEPENDANTS"
-- ----------------------------
DROP TABLE "CS342"."TMJ_DEPENDANTS";
CREATE TABLE "CS342"."TMJ_DEPENDANTS" (
"Bdate" DATE NULL ,
"Name" VARCHAR2(128 BYTE) NOT NULL ,
"Relationship" VARCHAR2(128 BYTE) NOT NULL ,
"Email" VARCHAR2(128 BYTE) NULL ,
"DateCreated" DATE NOT NULL ,
"EmpRef_ID" NUMBER(7) NOT NULL
)
```

```
LOGGING
NOCOMPRESS
NOCACHE

;

-- ----------------------------
-- Records of TMJ_DEPENDANTS
-- ----------------------------
INSERT INTO "CS342"."TMJ_DEPENDANTS" VALUES (TO_DATE('2011-07-23 00:00:00', 'YYYY-MM-DD HH24:MI:SS'), 'Dexter Jones', 'Son', null, TO_DATE('2011-04-20 00:00:00', 'YYYY-MM-DD HH24:MI:SS'), '2');
INSERT INTO "CS342"."TMJ_DEPENDANTS" VALUES (TO_DATE('1-01-01 00:00:00', 'YYYY-MM-DD HH24:MI:SS'), 'Reuben Sanders', 'Husband', 'SandersReuben@thething.com', TO_DATE('2005-06-05 00:00:00', 'YYYY-MM-DD HH24:MI:SS'), '3');
INSERT INTO "CS342"."TMJ_DEPENDANTS" VALUES (TO_DATE('2004-03-24 00:00:00', 'YYYY-MM-DD HH24:MI:SS'), 'Scott Alexander', 'Son', 'ScottA@makemoney.com', TO_DATE('2009-08-20 00:00:00', 'YYYY-MM-DD HH24:MI:SS'), '4');
INSERT INTO "CS342"."TMJ_DEPENDANTS" VALUES (TO_DATE('1980-09-02 00:00:00', 'YYYY-MM-DD HH24:MI:SS'), 'Jennifer McGuire', 'Wife', 'Jenn@thecompany.com', TO_DATE('1990-06-08 00:00:00', 'YYYY-MM-DD HH24:MI:SS'), '6');
INSERT INTO "CS342"."TMJ_DEPENDANTS" VALUES (TO_DATE('2004-06-09 00:00:00', 'YYYY-MM-DD HH24:MI:SS'), 'Betty Green', 'Daughter', null, TO_DATE('2001-01-08 00:00:00', 'YYYY-MM-DD HH24:MI:SS'), '9');
INSERT INTO "CS342"."TMJ_DEPENDANTS" VALUES (TO_DATE('2004-03-18 00:00:00', 'YYYY-MM-DD HH24:MI:SS'), 'Lester Recher', 'Wife', 'LesterR@nastynames.com', TO_DATE('2001-06-08 00:00:00', 'YYYY-MM-DD HH24:MI:SS'), '2');
INSERT INTO "CS342"."TMJ_DEPENDANTS" VALUES (TO_DATE('1985-05-12 19:16:23', 'YYYY-MM-DD HH24:MI:SS'), 'Brandon Jose', 'Husband', null, TO_DATE('2011-02-28 19:17:12', 'YYYY-MM-DD HH24:MI:SS'), '1');
INSERT INTO "CS342"."TMJ_DEPENDANTS" VALUES (null, 'Katie Haitfield', 'Daughter', 'KatieH@aol.com', TO_DATE('2011-04-20 19:17:58', 'YYYY-MM-DD HH24:MI:SS'), '2');
INSERT INTO "CS342"."TMJ_DEPENDANTS" VALUES (TO_DATE('1975-02-08 19:20:17', 'YYYY-MM-DD HH24:MI:SS'), 'Amanda Hooginkiss', 'Wife', null, TO_DATE('2008-08-18 19:21:10', 'YYYY-MM-DD HH24:MI:SS'), '8');

-- ----------------------------
-- Checks structure for table "CS342"."TMJ_DEPENDANTS"
-- ----------------------------
ALTER TABLE "CS342"."TMJ_DEPENDANTS" ADD CHECK ("Name" IS NOT NULL);
ALTER TABLE "CS342"."TMJ_DEPENDANTS" ADD CHECK ("Relationship" IS NOT NULL);
ALTER TABLE "CS342"."TMJ_DEPENDANTS" ADD CHECK ("DateCreated" IS NOT NULL);
ALTER TABLE "CS342"."TMJ_DEPENDANTS" ADD CHECK ("EmpRef_ID" IS NOT NULL);

-- ----------------------------
-- Foreign Key structure for table "CS342"."TMJ_DEPENDANTS"
-- ----------------------------
ALTER TABLE "CS342"."TMJ_DEPENDANTS" ADD FOREIGN KEY ("EmpRef_ID") REFERENCES "CS342"."TMJ_EMPLOYEES" ("Emp_ID") ON DELETE CASCADE DISABLE;
```

```
/*
Navicat Oracle Data Transfer
Oracle Client Version : 10.2.0.5.0

Source Server         : CSUB
Source Server Version : 110200
Source Host           : delphi.cs.csub.edu:1521
Source Schema         : CS342

Target Server Type    : ORACLE
Target Server Version : 110200
File Encoding         : 65001

Date: 2011-11-07 20:44:12
*/


-- ----------------------------
-- Table structure for "CS342"."TMJ_MANAGESFOR"
-- ----------------------------
DROP TABLE "CS342"."TMJ_MANAGESFOR";
CREATE TABLE "CS342"."TMJ_MANAGESFOR" (
"StoreRef_ID" NUMBER(3) NOT NULL ,
"EmpRef_ID" NUMBER(7) NOT NULL
)
LOGGING
NOCOMPRESS
NOCACHE

;

-- ----------------------------
-- Records of TMJ_MANAGESFOR
-- ----------------------------
INSERT INTO "CS342"."TMJ_MANAGESFOR" VALUES ('159', '4');
INSERT INTO "CS342"."TMJ_MANAGESFOR" VALUES ('247', '9');
INSERT INTO "CS342"."TMJ_MANAGESFOR" VALUES ('348', '9');
INSERT INTO "CS342"."TMJ_MANAGESFOR" VALUES ('354', '2');
INSERT INTO "CS342"."TMJ_MANAGESFOR" VALUES ('369', '6');
INSERT INTO "CS342"."TMJ_MANAGESFOR" VALUES ('674', '5');
INSERT INTO "CS342"."TMJ_MANAGESFOR" VALUES ('696', '3');
INSERT INTO "CS342"."TMJ_MANAGESFOR" VALUES ('778', '7');
INSERT INTO "CS342"."TMJ_MANAGESFOR" VALUES ('854', '1');
INSERT INTO "CS342"."TMJ_MANAGESFOR" VALUES ('989', '8');

-- ----------------------------
-- Indexes structure for table TMJ_MANAGESFOR
```

```
-- ----------------------------

-- ----------------------------
-- Checks structure for table "CS342"."TMJ_MANAGESFOR"
-- ----------------------------
ALTER TABLE "CS342"."TMJ_MANAGESFOR" ADD CHECK ("StoreRef_ID" IS NOT NULL);
ALTER TABLE "CS342"."TMJ_MANAGESFOR" ADD CHECK ("EmpRef_ID" IS NOT NULL);

-- ----------------------------
-- Primary Key structure for table "CS342"."TMJ_MANAGESFOR"
-- ----------------------------
ALTER TABLE "CS342"."TMJ_MANAGESFOR" ADD PRIMARY KEY ("StoreRef_ID", "EmpRef_ID");

-- ----------------------------
-- Foreign Key structure for table "CS342"."TMJ_MANAGESFOR"
-- ----------------------------
ALTER TABLE "CS342"."TMJ_MANAGESFOR" ADD FOREIGN KEY ("EmpRef_ID") REFERENCES
"CS342"."TMJ_EMPLOYEES" ("Emp_ID") ON DELETE CASCADE;
ALTER TABLE "CS342"."TMJ_MANAGESFOR" ADD FOREIGN KEY ("StoreRef_ID") REFERENCES
"CS342"."TMJ_STORE" ("Store_ID") ON DELETE CASCADE;

/*
Navicat Oracle Data Transfer
Oracle Client Version : 10.2.0.5.0

Source Server         : CSUB
Source Server Version : 110200
Source Host           : delphi.cs.csub.edu:1521
Source Schema         : CS342

Target Server Type    : ORACLE
Target Server Version : 110200
File Encoding         : 65001

Date: 2011-11-07 20:43:58
*/


-- ----------------------------
-- Table structure for "CS342"."TMJ_INVENTORY"
-- ----------------------------
DROP TABLE "CS342"."TMJ_INVENTORY";
CREATE TABLE "CS342"."TMJ_INVENTORY" (
"StoreRef_ID" NUMBER(3) NOT NULL ,
"ItemRef_ID" NUMBER(7) NOT NULL ,
"Quantity" NUMBER(7) NOT NULL
)
LOGGING
```

```
NOCOMPRESS
NOCACHE

;

-- ----------------------------
-- Records of TMJ_INVENTORY
-- ----------------------------
INSERT INTO "CS342"."TMJ_INVENTORY" VALUES ('854', '12', '10');
INSERT INTO "CS342"."TMJ_INVENTORY" VALUES ('854', '658', '10');
INSERT INTO "CS342"."TMJ_INVENTORY" VALUES ('354', '1566', '4');
INSERT INTO "CS342"."TMJ_INVENTORY" VALUES ('696', '12', '23');
INSERT INTO "CS342"."TMJ_INVENTORY" VALUES ('696', '658', '38');
INSERT INTO "CS342"."TMJ_INVENTORY" VALUES ('159', '335', '27');
INSERT INTO "CS342"."TMJ_INVENTORY" VALUES ('159', '1566', '31');
INSERT INTO "CS342"."TMJ_INVENTORY" VALUES ('674', '4587', '23');
INSERT INTO "CS342"."TMJ_INVENTORY" VALUES ('674', '2365', '0');

-- ----------------------------
-- Indexes structure for table TMJ_INVENTORY
-- ----------------------------

-- ----------------------------
-- Checks structure for table "CS342"."TMJ_INVENTORY"
-- ----------------------------
ALTER TABLE "CS342"."TMJ_INVENTORY" ADD CHECK ("Quantity" IS NOT NULL);

-- ----------------------------
-- Primary Key structure for table "CS342"."TMJ_INVENTORY"
-- ----------------------------
ALTER TABLE "CS342"."TMJ_INVENTORY" ADD PRIMARY KEY ("StoreRef_ID", "ItemRef_ID");

-- ----------------------------
-- Foreign Key structure for table "CS342"."TMJ_INVENTORY"
-- ----------------------------
ALTER TABLE "CS342"."TMJ_INVENTORY" ADD FOREIGN KEY ("StoreRef_ID") REFERENCES
"CS342"."TMJ_STORE" ("Store_ID") ON DELETE CASCADE;
ALTER TABLE "CS342"."TMJ_INVENTORY" ADD FOREIGN KEY ("ItemRef_ID") REFERENCES
"CS342"."TMJ_ITEMS" ("Item_ID") ON DELETE CASCADE;

/*
Navicat Oracle Data Transfer
Oracle Client Version : 10.2.0.5.0

Source Server        : CSUB
Source Server Version : 110200
Source Host          : delphi.cs.csub.edu:1521
Source Schema        : CS342
```

```
Target Server Type    : ORACLE
Target Server Version : 110200
File Encoding         : 65001

Date: 2011-11-07 20:43:38
*/


-- ----------------------------
-- Table structure for "CS342"."TMJ_CHECKOUT"
-- ----------------------------
DROP TABLE "CS342"."TMJ_CHECKOUT";
CREATE TABLE "CS342"."TMJ_CHECKOUT" (
"Checkout_ID" NUMBER(6) NOT NULL ,
"CustRef_ID" NUMBER(6) NOT NULL ,
"Date" DATE NOT NULL ,
"StoreRef_ID" NUMBER(3) NOT NULL ,
"Subtotal" NUMBER(5,2) NOT NULL ,
"Tax" NUMBER(5,2) NOT NULL ,
"EmpRef_ID" NUMBER(7) NOT NULL
)
LOGGING
NOCOMPRESS
NOCACHE

;
COMMENT ON TABLE "CS342"."TMJ_CHECKOUT" IS 'FK do not CASCADE because do not want to delete record if employee no longer works here';

-- ----------------------------
-- Records of TMJ_CHECKOUT
-- ----------------------------
INSERT INTO "CS342"."TMJ_CHECKOUT" VALUES ('32', '51', TO_DATE('2011-06-09 00:00:00', 'YYYY-MM-DD HH24:MI:SS'), '354', '115.25', '5', '2');
INSERT INTO "CS342"."TMJ_CHECKOUT" VALUES ('6589', '52', TO_DATE('2010-08-12 00:00:00', 'YYYY-MM-DD HH24:MI:SS'), '696', '66.52', '0.75', '3');
INSERT INTO "CS342"."TMJ_CHECKOUT" VALUES ('2147', '99', TO_DATE('2010-06-05 00:00:00', 'YYYY-MM-DD HH24:MI:SS'), '159', '500.25', '3.24', '4');
INSERT INTO "CS342"."TMJ_CHECKOUT" VALUES ('210', '179', TO_DATE('2009-11-05 00:00:00', 'YYYY-MM-DD HH24:MI:SS'), '674', '41.35', '1', '5');
INSERT INTO "CS342"."TMJ_CHECKOUT" VALUES ('2141', '105', TO_DATE('2007-04-05 00:00:00', 'YYYY-MM-DD HH24:MI:SS'), '369', '64.25', '3.25', '6');
INSERT INTO "CS342"."TMJ_CHECKOUT" VALUES ('3652', '178', TO_DATE('2011-12-12 00:00:00', 'YYYY-MM-DD HH24:MI:SS'), '778', '14.25', '1.20', '6');
INSERT INTO "CS342"."TMJ_CHECKOUT" VALUES ('125', '58', TO_DATE('2005-12-24 00:00:00', 'YYYY-MM-DD HH24:MI:SS'), '989', '80.85', '3.02', '7');
```

```
INSERT INTO "CS342"."TMJ_CHECKOUT" VALUES ('22', '50', TO_DATE('2011-11-06 16:08:23', 'YYYY-MM-DD HH24:MI:SS'), '854', '25', '1.25', '2');

-- ----------------------------
-- Indexes structure for table TMJ_CHECKOUT
-- ----------------------------

-- ----------------------------
-- Checks structure for table "CS342"."TMJ_CHECKOUT"
-- ----------------------------
ALTER TABLE "CS342"."TMJ_CHECKOUT" ADD CHECK ("Checkout_ID" IS NOT NULL);
ALTER TABLE "CS342"."TMJ_CHECKOUT" ADD CHECK ("CustRef_ID" IS NOT NULL);
ALTER TABLE "CS342"."TMJ_CHECKOUT" ADD CHECK ("Date" IS NOT NULL);
ALTER TABLE "CS342"."TMJ_CHECKOUT" ADD CHECK ("StoreRef_ID" IS NOT NULL);
ALTER TABLE "CS342"."TMJ_CHECKOUT" ADD CHECK ("Subtotal" IS NOT NULL);
ALTER TABLE "CS342"."TMJ_CHECKOUT" ADD CHECK ("Tax" IS NOT NULL);
ALTER TABLE "CS342"."TMJ_CHECKOUT" ADD CHECK ("EmpRef_ID" IS NOT NULL);

-- ----------------------------
-- Primary Key structure for table "CS342"."TMJ_CHECKOUT"
-- ----------------------------
ALTER TABLE "CS342"."TMJ_CHECKOUT" ADD PRIMARY KEY ("Checkout_ID");

-- ----------------------------
-- Foreign Key structure for table "CS342"."TMJ_CHECKOUT"
-- ----------------------------
ALTER TABLE "CS342"."TMJ_CHECKOUT" ADD FOREIGN KEY ("CustRef_ID") REFERENCES "CS342"."TMJ_CUSTOMER" ("Cust_ID");
ALTER TABLE "CS342"."TMJ_CHECKOUT" ADD FOREIGN KEY ("StoreRef_ID") REFERENCES "CS342"."TMJ_STORE" ("Store_ID");
ALTER TABLE "CS342"."TMJ_CHECKOUT" ADD FOREIGN KEY ("EmpRef_ID") REFERENCES "CS342"."TMJ_EMPLOYEES" ("Emp_ID");

/*
Navicat Oracle Data Transfer
Oracle Client Version : 10.2.0.5.0

Source Server         : CSUB
Source Server Version : 110200
Source Host           : delphi.cs.csub.edu:1521
Source Schema         : CS342

Target Server Type    : ORACLE
Target Server Version : 110200
File Encoding         : 65001

Date: 2011-11-07 20:43:43
*/
```

```
-- ----------------------------
-- Table structure for "CS342"."TMJ_CHECKOUT_ACTION"
-- ----------------------------
DROP TABLE "CS342"."TMJ_CHECKOUT_ACTION";
CREATE TABLE "CS342"."TMJ_CHECKOUT_ACTION" (
"CheckoutRef_ID" NUMBER(7) NOT NULL ,
"ItemRef_ID" NUMBER(7) NOT NULL
)
LOGGING
NOCOMPRESS
NOCACHE

;

-- ----------------------------
-- Records of TMJ_CHECKOUT_ACTION
-- ----------------------------
INSERT INTO "CS342"."TMJ_CHECKOUT_ACTION" VALUES ('32', '3521');
INSERT INTO "CS342"."TMJ_CHECKOUT_ACTION" VALUES ('32', '84854');
INSERT INTO "CS342"."TMJ_CHECKOUT_ACTION" VALUES ('210', '12');
INSERT INTO "CS342"."TMJ_CHECKOUT_ACTION" VALUES ('210', '658');
INSERT INTO "CS342"."TMJ_CHECKOUT_ACTION" VALUES ('2147', '4587');
INSERT INTO "CS342"."TMJ_CHECKOUT_ACTION" VALUES ('6589', '2365');
INSERT INTO "CS342"."TMJ_CHECKOUT_ACTION" VALUES ('6589', '4587');

-- ----------------------------
-- Indexes structure for table TMJ_CHECKOUT_ACTION
-- ----------------------------

-- ----------------------------
-- Checks structure for table "CS342"."TMJ_CHECKOUT_ACTION"
-- ----------------------------
ALTER TABLE "CS342"."TMJ_CHECKOUT_ACTION" ADD CHECK ("CheckoutRef_ID" IS NOT NULL);
ALTER TABLE "CS342"."TMJ_CHECKOUT_ACTION" ADD CHECK ("ItemRef_ID" IS NOT NULL);

-- ----------------------------
-- Primary Key structure for table "CS342"."TMJ_CHECKOUT_ACTION"
-- ----------------------------
ALTER TABLE "CS342"."TMJ_CHECKOUT_ACTION" ADD PRIMARY KEY ("CheckoutRef_ID",
"ItemRef_ID");

-- ----------------------------
-- Foreign Key structure for table "CS342"."TMJ_CHECKOUT_ACTION"
-- ----------------------------
ALTER TABLE "CS342"."TMJ_CHECKOUT_ACTION" ADD FOREIGN KEY ("CheckoutRef_ID")
REFERENCES "CS342"."TMJ_CHECKOUT" ("Checkout_ID") ON DELETE CASCADE;
```

```
ALTER TABLE "CS342"."TMJ_CHECKOUT_ACTION" ADD FOREIGN KEY ("ItemRef_ID") REFERENCES
"CS342"."TMJ_ITEMS" ("Item_ID") ON DELETE CASCADE;

/*
Navicat Oracle Data Transfer
Oracle Client Version : 10.2.0.5.0

Source Server         : CSUB
Source Server Version : 110200
Source Host           : delphi.cs.csub.edu:1521
Source Schema         : CS342

Target Server Type    : ORACLE
Target Server Version : 110200
File Encoding         : 65001

Date: 2011-11-07 20:44:08
*/


-- ----------------------------
-- Table structure for "CS342"."TMJ_LOGTABLE"
-- ----------------------------
DROP TABLE "CS342"."TMJ_LOGTABLE";
CREATE TABLE "CS342"."TMJ_LOGTABLE" (
"oldVal" NUMBER(7) NOT NULL ,
"newVal" NUMBER(7) NOT NULL
)
LOGGING
NOCOMPRESS
NOCACHE
;
-- ----------------------------
-- Records of TMJ_LOGTABLE
-- ----------------------------

-- ----------------------------
-- Checks structure for table "CS342"."TMJ_LOGTABLE"
-- ----------------------------
ALTER TABLE "CS342"."TMJ_LOGTABLE" ADD CHECK ("oldVal" IS NOT NULL);
ALTER TABLE "CS342"."TMJ_LOGTABLE" ADD CHECK ("newVal" IS NOT NULL);
```

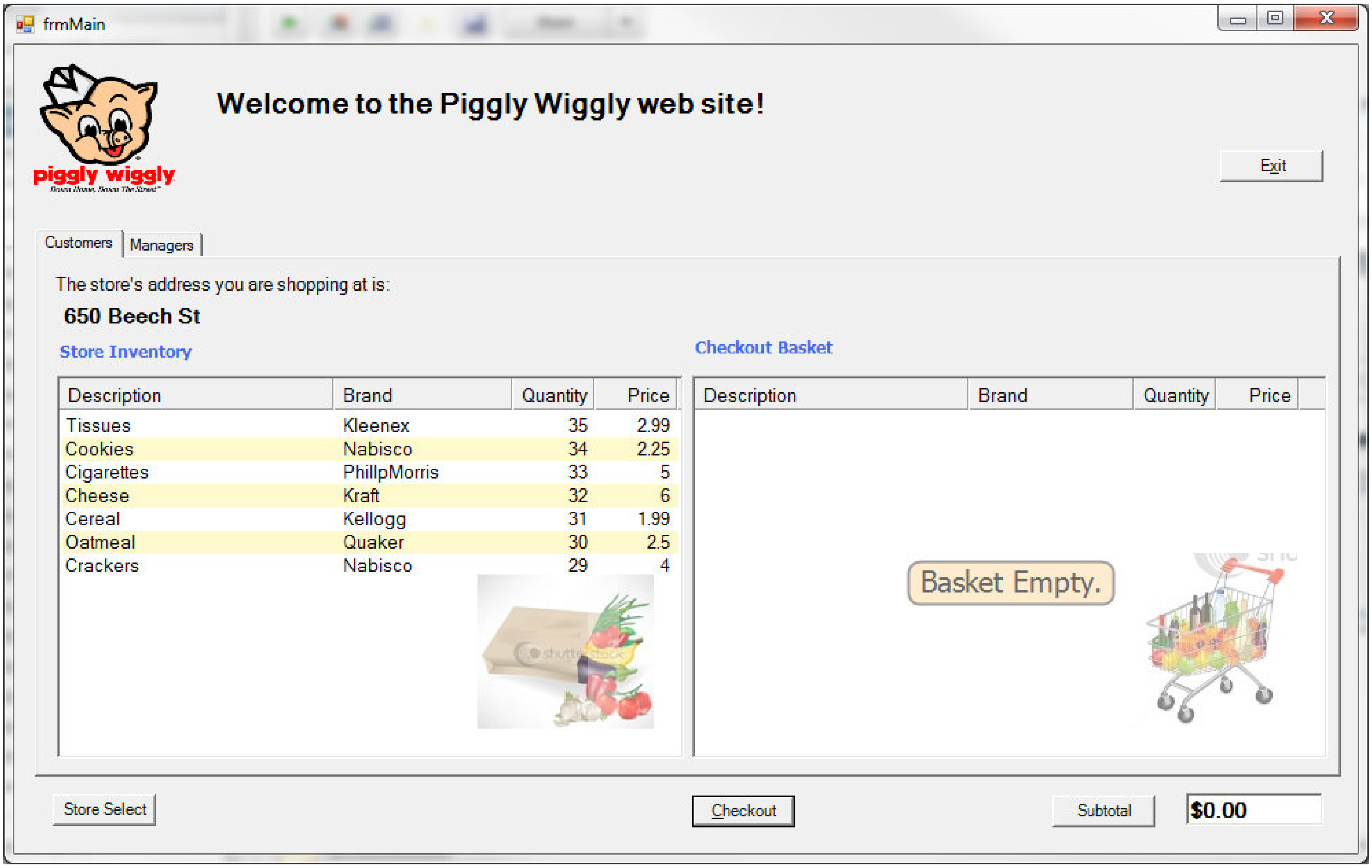
5.1 Daily Activities of the User Group

Daily activities of the Piggly Wiggly UI and the backend database can be broken into two functional areas. The first functional area fulfills the main point of the grocery business, customer's buying product and checking out. The customer accesses the 'Customer' tab and then can select the store address to shop in. In this fictional program, the user also has to select the customer name and the employee that will check them out. These parameters are needed by the application later in the checkout section to properly insert the checkout record in to the database.

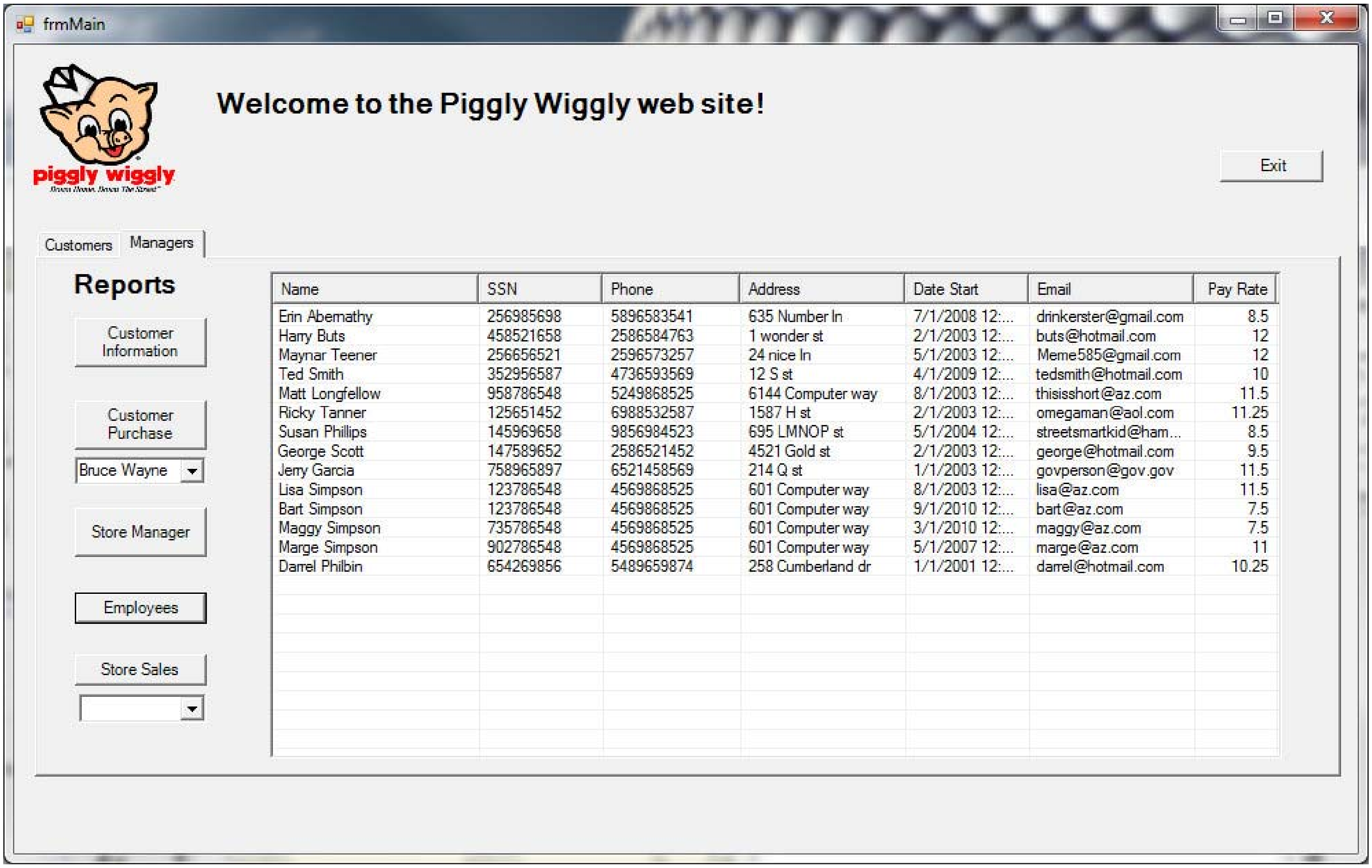
Once these are selected and the 'Shop' button is selected, the user can add items to the checkout cart and then proceed to checkout. The user can change the quantities and remove items from their basket. All these are the basic functions expected by a shopper at a grocery store.

The second part of the application allows the managers of the store to run reports and view the status of the store's sales, employee information, customer information & shopping history, and a store manager list. Managers will use a company's database for a variety of reports to get their job done. This application tries to realistically simulate a reporting interface for managers.

The system should not require much in the way of technical ability by the users to run the application or maintain it. We have used a simple tabbed interface. Customers can shop by simply dragging and dropping items into or out of their cart and modifying the quantity to purchase. When they checkout, they will be presented a list of purchased items and the total price in a separate form. In addition, the checkout action properly updates the DB to allow the managers to review the checkout activity, store inventory and customer sales history. This well designed application should increase sales and allow easier management of the stores.



Item selection screen for customers with the checkout basket.

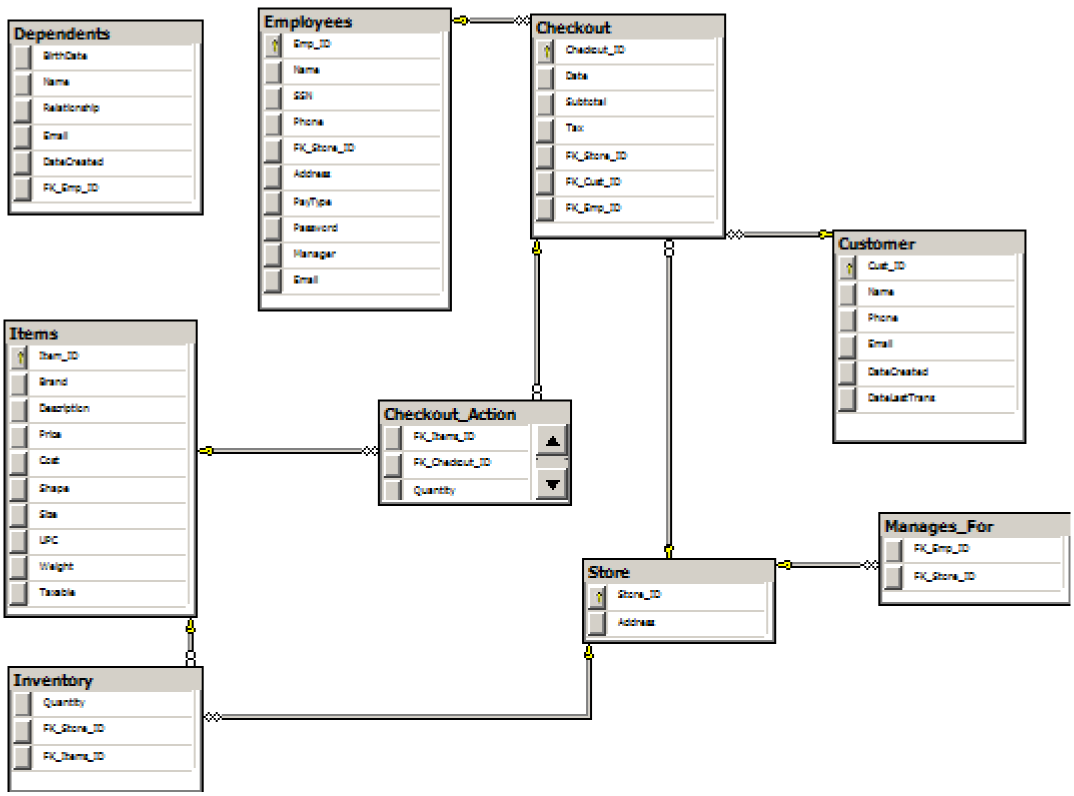


Name	SSN	Phone	Address	Date Start	Email	Pay Rate
Erin Abernathy	256985698	5896583541	635 Number ln	7/1/2008 12:...	drinkerster@gmail.com	8.5
Harry Buts	458521658	2586584763	1 wonder st	2/1/2003 12:...	buts@hotmail.com	12
Maynar Teener	256656521	2596573257	24 nice ln	5/1/2003 12:...	Meme585@gmail.com	12
Ted Smith	352956587	4736593569	12 S st	4/1/2009 12:...	tedsmith@hotmail.com	10
Matt Longfellow	958786548	5249868525	6144 Computer way	8/1/2003 12:...	thisisshort@az.com	11.5
Ricky Tanner	125651452	6988532587	1587 H st	2/1/2003 12:...	omegaman@aol.com	11.25
Susan Phillips	145969658	9856984523	695 LMNOP st	5/1/2004 12:...	streetsmartkid@ham...	8.5
George Scott	147589652	2586521452	4521 Gold st	2/1/2003 12:...	george@hotmail.com	9.5
Jerry Garcia	758965897	6521458569	214 Q st	1/1/2003 12:...	govperson@gov.gov	11.5
Lisa Simpson	123786548	4569868525	601 Computer way	8/1/2003 12:...	lisa@az.com	11.5
Bart Simpson	123786548	4569868525	601 Computer way	9/1/2010 12:...	bart@az.com	7.5
Maggy Simpson	735786548	4569868525	601 Computer way	3/1/2010 12:...	maggy@az.com	7.5
Marge Simpson	902786548	4569868525	601 Computer way	5/1/2007 12:...	marge@az.com	11
Darrel Philbin	654269856	5489659874	258 Cumberland dr	1/1/2001 12:...	darrel@hotmail.com	10.25

Employee report screenshot.

5.2 Relations, views and subprograms

The relational database has nine tables, two stored procedures, and eight constraints. It uses an auto-increment primary key feature in the ‘Checkout’ table. We built a prototype db in MS SQL 2008 in order to rapid prototype the C# UI design. Then we built the SQL structure on Oracle and worked on porting the database connectivity from MS SQL to Oracle. The MS SQL front end provided a friendly interface to build and script the tables, relations and stored procedures. This made development and debugging easier.



The database functions, or stored procedures, are shown below.

```
ALTER PROCEDURE [dbo].[insertCheckoutRecord]
      -- Add the parameters for the stored procedure here
      @Date datetime,
      @Subtotal money,
      @Tax money,
      @Store_ID int,
      @Cust_ID int,
      @Emp_ID int,
      @ID int output
AS
BEGIN
```

```
	-- SET NOCOUNT ON added to prevent extra result sets from
	-- interfering with SELECT statements.
	SET NOCOUNT ON;

    -- Insert statements for procedure here
	INSERT INTO [grocery].[dbo].[Checkout] ([Date], [Subtotal], [Tax],
	[FK_Store_ID], [FK_Cust_ID], [FK_Emp_ID])
    VALUES ( GETUTCDATE(), @Subtotal, @Tax, @Store_ID, @Cust_ID, @Emp_ID)
    SELECT @ID = SCOPE_IDENTITY()
	RETURN @ID

END
```

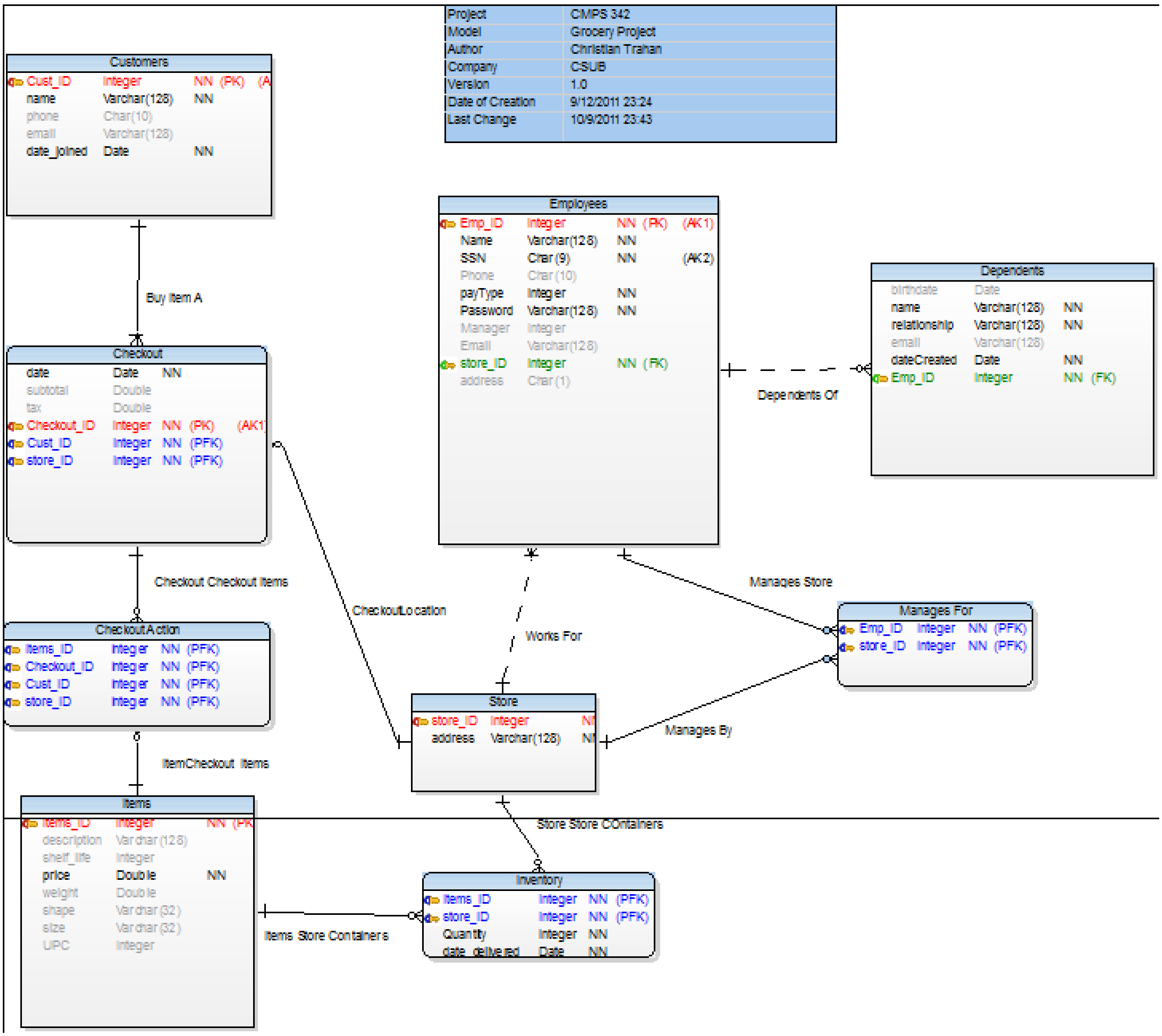
Notice the input and output variables used. These match up with the C# Sqlparameters object in the client application. This stored procedure uses an auto-increment column for the primary key. In order to return this column ID at the time of a new record being inserted into the table, you have to use the 'SCOPE_IDENTITY()' function. By returning the Checkout_ID in this fashion, the application can use that return value to properly insert another record into the Checkout_Action table.

The second stored procedure is:

```
	SET NOCOUNT ON;
	DECLARE @tempQuantity int
	SET @tempQuantity = (SELECT quantity from Inventory WHERE FK_Items_ID = @ID and
FK_Store_ID = @StoreID)
	IF ( @Quantity > @tempQuantity)
	BEGIN
		SET @Quantity = 0
	END
	ELSE
	BEGIN
		SET @Quantity = @tempQuantity - @Quantity
	END
	UPDATE [grocery].[dbo].[Inventory]
		SET [Quantity] = @Quantity
		WHERE FK_Items_ID = @ID and FK_Store_ID = @StoreID
```

This second stored procedure shows an example of multiple statements in one query and some logic processing of the returned value within the stored procedure. This is faster and more secure than doing multiple client requests and handling the code in the client application.

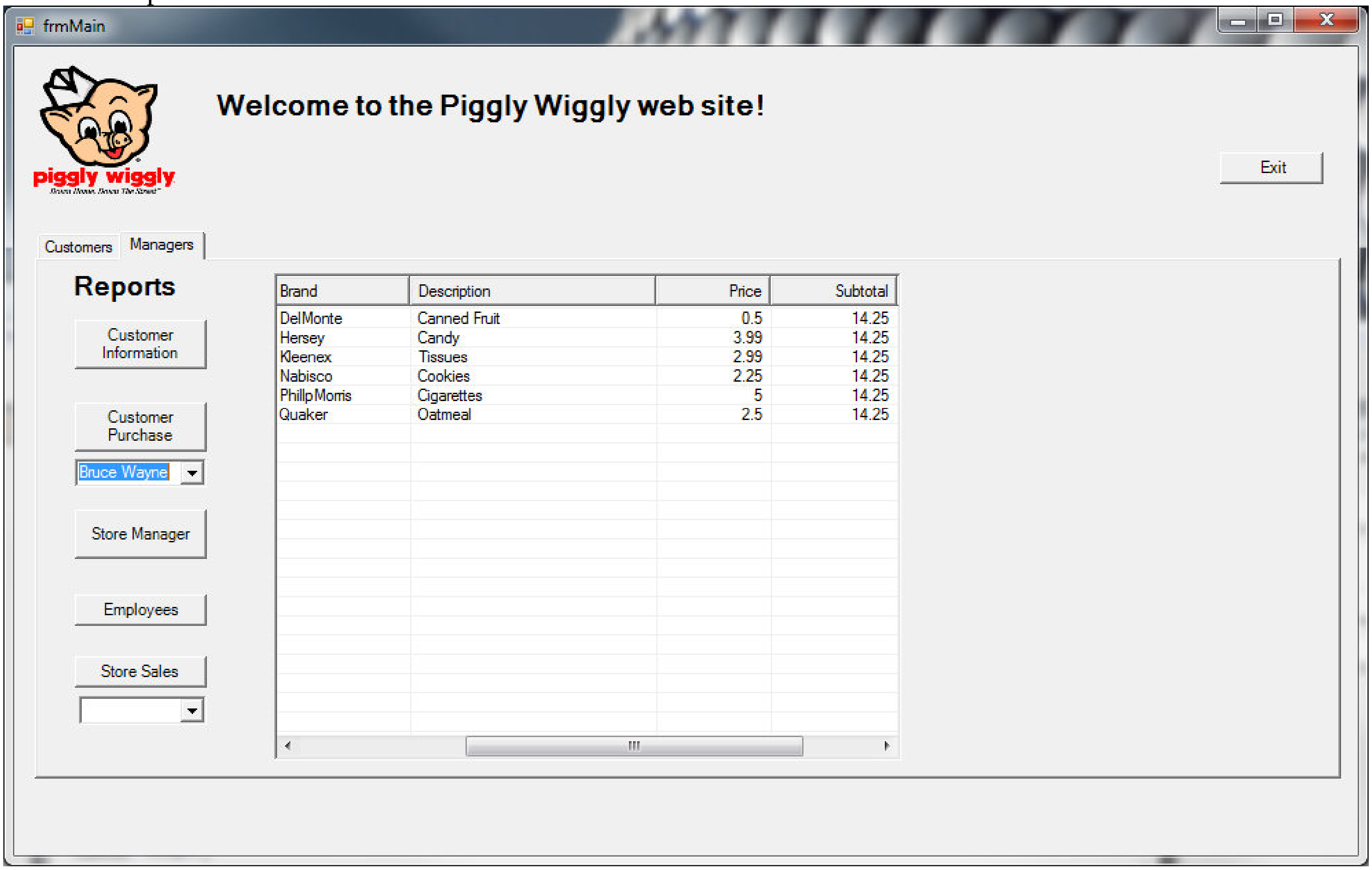
Relational Model Diagram



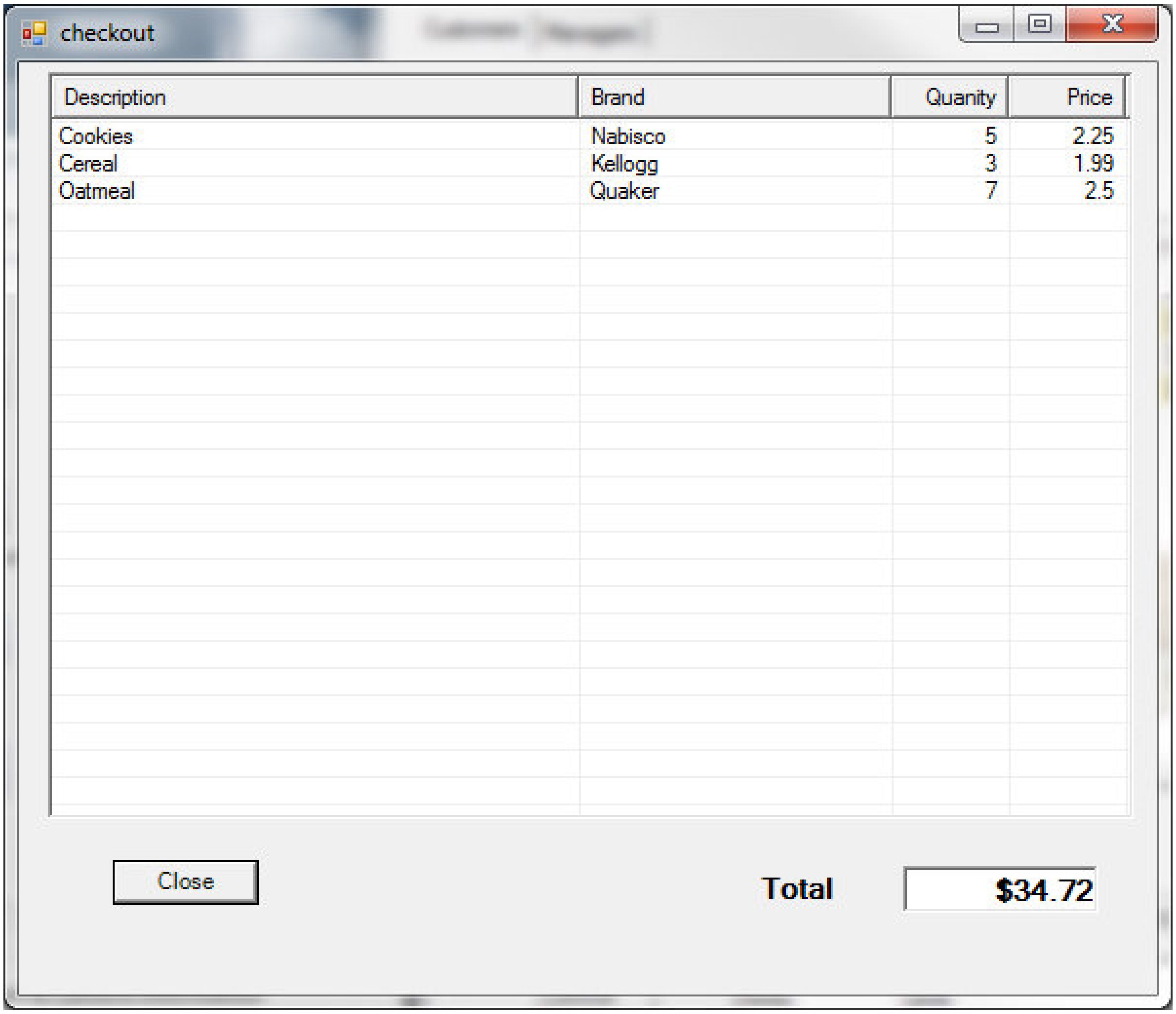
5.3 Screenshots

Customer Report:

Purchase Report:



Checkout Invoice:



5.4 Code description

The code was written in C# using MS Visual Studio 2008. The project includes two other external projects from CodeProject (http://www.codeproject.com). One of these projects is a splash screen that utilizes multi-threading and transparency to display a startup splash screen. This is a professional feature that allows other modules to load in the background and tells the user the application is loading. The second included project is an Object List View (OLV) component that allows a much more sophisticated and polished UI component. It is a replacement for the stock C# .NET object ListView. The OLV component was used in every tab of the program. It is used for the store inventory and shopping cart screen and also used for all the report screens. This OLV required extensive use of C# objects in the development of the application. Also used extensive object casting to retrieve data from the OLV objects.

The application uses the .NET data connection objects to read data from the database. These objects include SqlCommand, SqlConnection, SqlDataReader. They allow the client .NET application to connect to the SQL database and execute SQL commands or to run stored procedures.

The splash screen utilizes a multi-threaded capability so as to load the splash screen while still loading the back end application modules. The main form object loads and displays the multi-tabbed object. The application uses the standard Windows events to trigger on certain events like button clicks, mouse drag-and-drop, form load, etc. The application passes array structures and objects between form objects utilizing object constructors. An example of this is:

```
        public frmCheckout(ObjectListView inputOLV, int[] selectionInputs)
        {
            InitializeComponent();
            _inputOLV = inputOLV;
            _storeID = selectionInputs[0];
            _empID = selectionInputs[1];
            _custID = selectionInputs[2];
        }
```

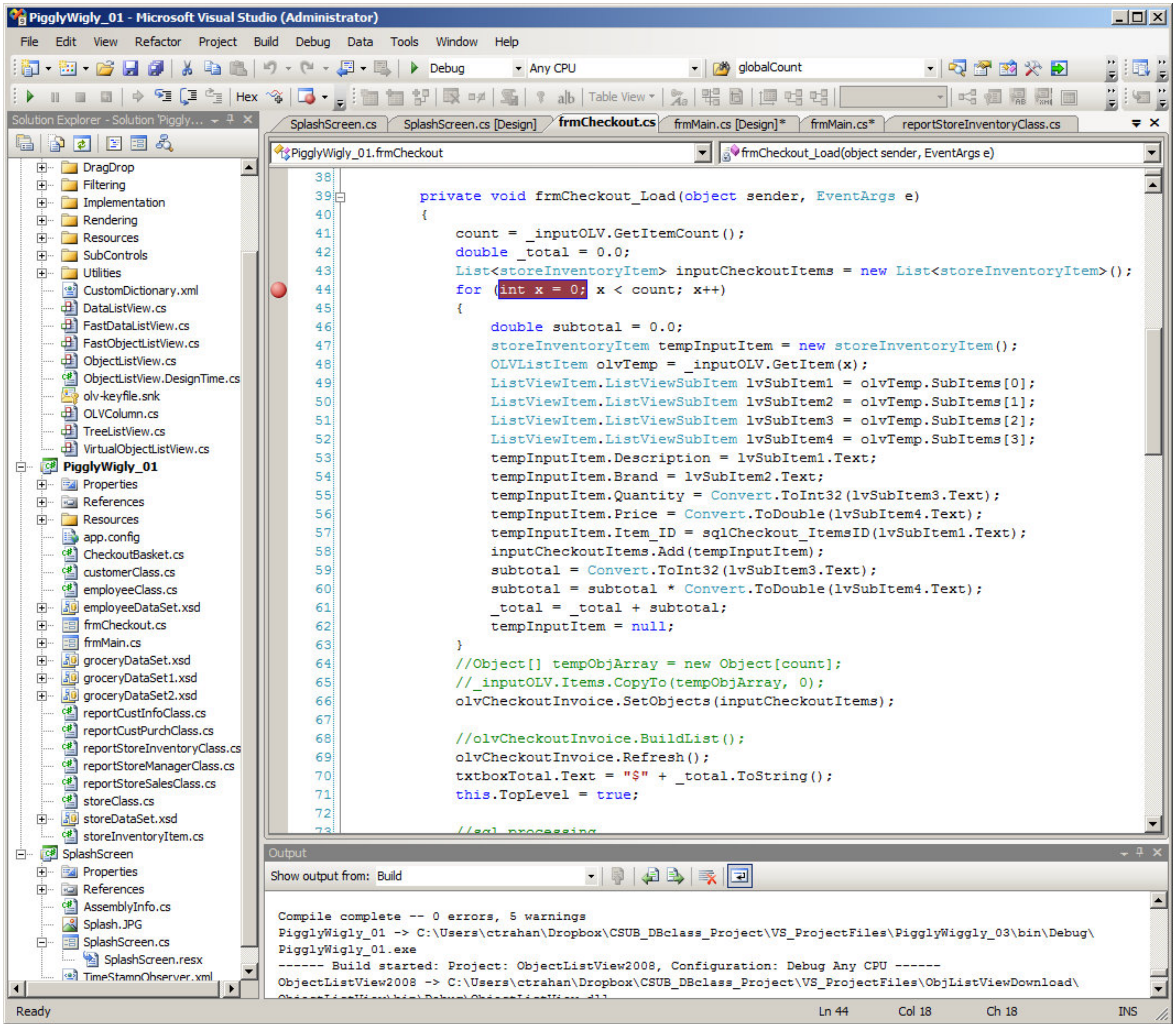
The application uses standard .NET try/catch blocks to catch errors and will exit on fatal errors after displaying a message. It also uses the ‘using’ function when it creates the SQL connection objects. This makes sure the SQL connection is properly closed when the scope leaves the function.

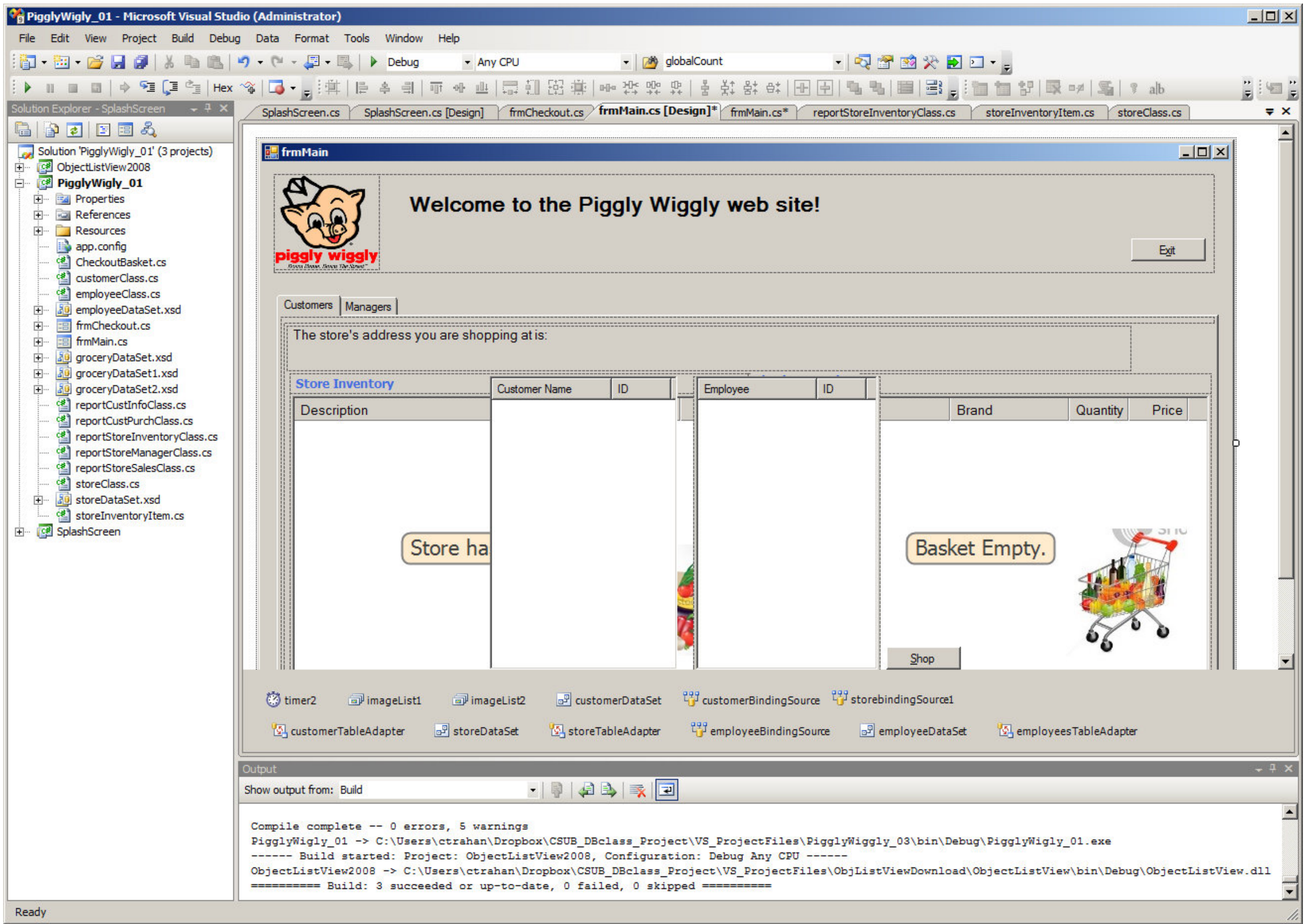
```
//Get connection string
            string cnStr =
ConfigurationManager.ConnectionStrings["PigglyWigly_01.Properties.Settings.groceryConnection
String"].ConnectionString;
            // create and open a connection
            SqlConnection sqlCn = new SqlConnection();
            OpenConnection(sqlCn, cnStr);

            using (SqlCommand cmd = new SqlCommand("updateStoreQuantity", sqlCn))
            {
                cmd.CommandType = CommandType.StoredProcedure;
                SqlParameter param = new SqlParameter();

                param.ParameterName = "@Quantity";
                param.SqlDbType = SqlDbType.Int;
                param.Value = inputQuantity;
                param.Direction = ParameterDirection.Input;
                cmd.Parameters.Add(param);
```

…




PigglyWigly_01 - Microsoft Visual Studio (Administrator)
frmMain
Welcome to the Piggly Wiggly web site!
Exit
Customers
Managers
The store's address you are shopping at is:
Store Inventory
Description
Customer Name
ID
Employee
Brand
Quantity
Price
Basket Empty.
Shop
timer2
imageList1
imageList2
customerDataSet
customerBindingSource
storebindingSource1
customerTableAdapter
storeDataSet
storeTableAdapter
employeeBindingSource
employeeDataSet
employeesTableAdapter
Output
Show output from: Build
Compile complete -- 0 errors, 5 warnings
PigglyWigly_01 -> C:\Users\ctrahan\Dropbox\CSUB_DBclass_Project\VS_ProjectFiles\PigglyWiggly_03\bin\Debug\PigglyWigly_01.exe
------ Build started: Project: ObjectListView2008, Configuration: Debug Any CPU ------
ObjectListView2008 -> C:\Users\ctrahan\Dropbox\CSUB_DBclass_Project\VS_ProjectFiles\ObjListViewDownload\ObjectListView\bin\Debug\ObjectListView.dll
========== Build: 3 succeeded or up-to-date, 0 failed, 0 skipped ==========
Ready

5.5 Major Steps and learning process

This project has been a great learning process for SQL relational technology and object programming using Visual Studio 2008 and .NET technologies. The fundamentals of the database design started with the process of research and conceptual design. The entities and their attributes were then identified. The relationships between those entities were then identified. The relationship model was created to describe the tables, candidate keys, primary keys and foreign keys. This entity relationship model is described on page 21.

From here we used relational algebra to design the queries to insert, update and delete data in the database. The queries were then translated to SQL. The database model was built in MS SQL using TSQL and a model in Oracle using Navicat. Navicat works with Oracle databases.

Building the UI in C# took weeks with a lot of trial and error. The Visual Studio IDE was very nice to work with and I learned a lot about C# objects, including third party objects, debugging, SQL integration, and Oracle integration. I plan on using the skills from this project on some of my current work projects. For example, I will be using the OLV object to replace the ListView object in certain projects. The UI needs more work but we believe it functions quite well. This project has increased our skills in both SQL technology and C# programming.