Web-based Geek Store Database

By Gabe Pike and Jon Hardin

Dr. Huaqing Wang

CMPS 342

Fall 2011

Table of Contents

Phase I

1.1a FACT-FINDING TECHNIQUES

With the proliferation of web-based market systems on the internet, it was not hard to gather relevant facts. An existing website that has a database similar to what we have in mind is www.thinkgeek.com. We analyzed the hierarchical structure of its store and identified important entities, attributes, and relationships. We looked for examples of market database models and used them as guidance to develop our own model.

1.2b INTRODUCTION TO ENTERPRISE

This enterprise is an online store designed to cater to people with geeky interests. A web app will be developed for the end user to navigate the website and purchase products. Products include clothing/apparel, gadgets, computer components, toys, electronics, and decorations and tools for the home or office. It will be a small business (at least to start), but has the potential to grow.

1.3c SCOPE OF CONCEPTUAL DATABASE

Our conceptual database model is only going to cover the parts that deal directly with the online store. It will not cover employees or payroll, but this may be something to consider in a more realistic model, especially if the company grows beyond just a few employees. The entities covered will be User, Item, Item Category, Order, Order Items, and Credit Card.

1.1d ITEMIZED DESCRIPTIONS OF ENTITY SETS AND RELATIONSHIP SETS

A user is a customer, moderator, or administrator who has an account on the store's website. A credit card is a method of payment for a user, and a user can have multiple credit cards on file. An item is a product that can be sold. An item category is used to categorize items. There can be sub-categories. Each category must store all its parent categories to make it easier to search for items. An order is a purchase placed by a user. It also has a list of order items and a payment method (credit card). An order item is used to keep track of each item in an order and the quantity ordered.

1.1e USER GROUPS, DATA VIEWS, AND OPERATIONS

There are three types of users: super administrators, moderators and customers. Administrators can see all data. Customers can only see their own account, credit cards, and order history. Customers can add, remove, and change their account information. Once an order has been put through, they can cancel or edit it before it is shipped. Moderators can see customer data, except for their password hashes and the first 12 characters of their credit card number. They can add or remove customers, but not other moderators or admins. Super administrators can see all data except the first 12 characters of credit card numbers. They can add, remove, or change any user in the database. Moderators and admins can view various reports of usage statistics and financial data.

1.2a ENTITY SET DESCRIPTION

User

This entity will store the info of each user. The entity will hold necessary data such as contact info. A new entry will be added to the database when they are sign up, each entry will be stored and never deleted for our records. Entries may be updated if any data other than user name changes about the person in question.

Candidate keys: email, name

Primary keys: email

Strong / weak entity: Strong

Fields to be indexed: email, name

Name	email	name	Address	password	Acct Type	subscriber
Description	Identifies user	Full name	Physical address	hashed password	Determine privileges	Does user want emails
Domain/Type	String	String	String	String	Integer	Integer
Value Range	Any valid email	Alphabetic	Any	MD5 hash	0=admin, 1=mod., 2=cust.	0=false, 1=true
Default Value	None	None	None	None	2	0
Nullable?	No	No	Yes	No	No	No
Unique?	Yes	No	No	No	No	No
Single or Multiple	Single	Single	Multiple	Single	Single	Single
Simple or Composite	Simple	Comp.	Comp.	Simple	Simple	Simple

Order

This entity will take the order of a person and their payment info.

Candidate keys: Order ID

Primary keys: Order ID

Strong / weak entity: Strong

Fields to be indexed: OrderID, Date

Name	OrderID	Shipping Address	Billing Address	Quantity	Price	Date
Description	Identifies Order	Address to ship to	Address to bill to	Qty of item	Price of item	Date of the order
Domain/Type	Integer	String	String	integer	Float	Date
Value Range	Positive int	Any	Any	0-*	Positive float	Any valid date
Default Value	Auto Increment	None	None	None	None	Current date
Nullable?	No	No	No	No	No	No
Unique?	Yes	No	No	No	No	No
Single or Mult	Single	Single	Single	Single	Single	Single
Simple/Comp	Simple	Comp	Comp	Simple	Simple	Simple

Credit Card

This entity will hold the users credit card information.

Candidate keys: Card number

Primary keys: Card number

Strong / weak entity: Weak

Fields to be indexed:

Name	Card number	Card holder	Provider	Exp date
Description	Identifies credit card	User's full Name	Credit card company	Expiration date of card
Domain/Type	Int	String	String	String
Value Range	16	0-40	0-30	4
Default Value	None	None	None	None
Nullable?	No	No	No	No
Unique?	Yes	No	Yes	No
Single or Mult	Single	Single	Single	Single
Simple or Comp.	Simple	Composite	Simple	Simple

Item

This entity will hold the information of the items on the page.

Candidate keys: ItemID, Name

Primary keys: ItemID

Strong / weak entity: Strong

Fields to be indexed: ItemID, Name

Name	ItemID	Name	Price	Description
Description	Identifies item	User's full Name	User's address	User's email
Domain/Type	Integer	String	Float	String
Value Range	*	String	0 – max float	*
Default Value	None	None	None	None
Nullable?	No	No	Yes	Yes
Unique?	Yes	Yes	No	No
Single or Mult.	Single	Single	Single	Single
Simple or Comp.	Simple	Composite	Simple	Simple

Name	Photo	Video	Stock	manufacturer
Description	# of item photos	# of item videos	# in stock	item manufacturer
Domain/Type	Integer	Integer	Integer	String
Value Range	0-*	0-*	0-max int	*
Default Value	None	None	None	None
Nullable?	Yes	Yes	No	Yes
Unique?	No	No	No	No
Single or Multiple	Single	Single	Single	Single
Simple or Comp.	Simple	Simple	Simple	Simple

Category

This entity will hold which category the item belongs in. It is a recursive entity because every category holds a reference to its parent category.

Candidate keys: Category name

Primary keys: Category name

Strong / weak entity: Strong

Fields to be indexed: Category ID, Category name, Parent

Name	Category ID	Category name	Description	Parent
Description	Identifies category	The category of the item	Description of the category	Parent of the category
Domain/Type	Integer	String	String	Integer
Value Range	Any valid int	0-40	0-1000	Any existing category ID
Default Value	None	None	None	None
Nullable?	No	No	Yes	Yes
Unique?	Yes	Yes	No	No
Single/Mult	Single	Single	Single	Multiple
Simple/Comp	Simple	Simple	Simple	Simple

1.2b RELATION SET DESCRIPTION

User Places an Order

- A use can select a group of items in their shopping cart and place an order.
- Mapping cardinality: M…M
- Descriptive field: none
- Participation constraint: Mandatory for user and order

User owns a Credit Card

- A user must have credit cards associated with her account if she is going to make an order
- Mapping cardinality: 1…M
- Descriptive field: none
- Participation constraint: Optional for User and mandatory for Credit Card

Order charges a Credit Card

- An order must have a credit card associated with it to charge for the purchase
- Mapping cardinality: M…1
- Descriptive field: none
- Participation constraint: Mandatory for Order and optional for Credit Card

Item has a Category

- An item can fall under categories or sub-categories

- Mapping cardinality: M…M
- Descriptive field: none
- Participation constraint: Optional for both

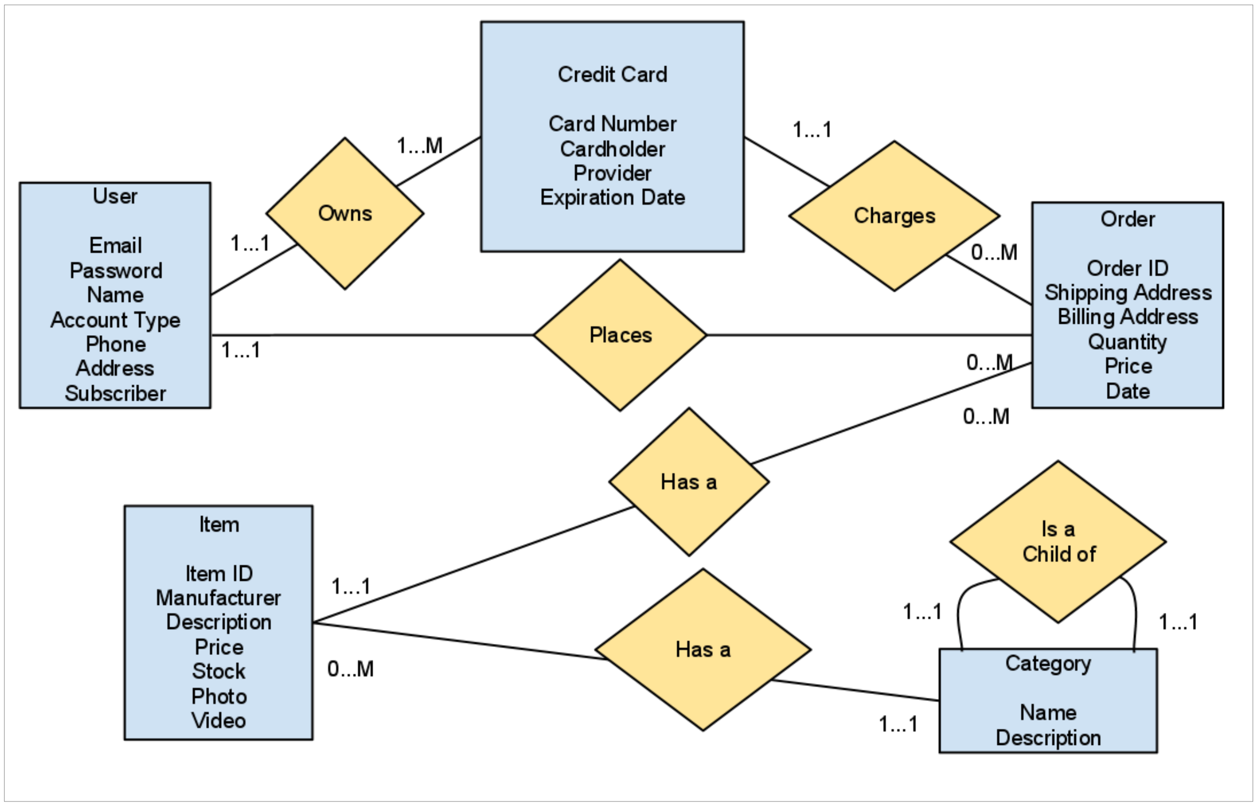
Category is a child of a Category

- A category can be divided into sub-categories. Sub-categories list each parent in order. This can be used to help search for items by category. A category without a parent lists itself as a parent.
- Mapping cardinality: M…1
- Descriptive field: None
- Participation constraint: Mandatory for both

1.2c RELATED ENTITY SET

The only specialization/sub-type relationship is for category. Category has a recursive “child of” relationship to its parent category. There are several has-a relationships. A user has at least one credit card, and an order has a credit card and an item. In addition, an order item has one item, and an order can have many order items.

1.2d E-R DIAGRAM



Phase II

2.1 E-R MODEL AND RELATIONAL MODEL

The entity-relationship model is a popular conceptual data model designed to be easy to read by management and other non-technical people. It is meant to convey the abstract design of a schema in the form of an E-R diagram. The diagram consists of entities which own certain attributes. In addition, the model shows relationships between entities. The relationships are represented as links between two entities with action words such as “orders” or “works for”.

Edgar F. Codd proposed the idea of the relational model in 1969. The relational model was a major advancement in database modeling and is still used to this day in major database management systems. The purpose of this model is to provide the framework for making specific queries on a database. With this model came relational algebra and relational calculus, which for the most part have been implemented in modern DBMS's. The relational model is more detailed than the E-R model, but it is more useful for people who need to design a database or write complex queries for it.

Conversion from the ER Model to the Relational Model

The ER model only describes the data requirements of each entity, therefore a conversion from the ER model to the Relational model is needed in order to implement the entities into a database. For each type there are special requirements for each conversion, which are listed below.

Strong Entities:

One or more simple attributes are used to create a primary key. A primary key is a selected attribute that is unique or should be composed of multiple attributes that they themselves are unique. Other possible candidate keys,which are keys that can be primary keys, but are not selected may be used for other purposes such as indexes.

Weak Entities:

A weak entity cannot exist alone, and therefore needs a primary key from the owner and a foreign key from the weak entity to represent a primary key for the weak entity.

One to one relationships(1:1):

There are two methods one for total participation or without total participation.

With total participation of one entity:

Include all attributes of one entity, primary key of the other entity, and descriptive attributes of the relationship.

Without total participation:

Include only primary keys from the entities and descriptive attributes of the relation.

One to many relationships(1:M) without total participation:

Same procedure as one to one relationship without total participation.

One to many relationships(1:M) with total participation:

Include all attributes from the many sides entity(M), descriptive attributes of the relationship, and the primary key of the one sides entity(1).

Many to Many relationships(M:M):

Same procedure as one to one relationship without total participation. primary key will be the union of the foreign keys of the two entities.

Ternary and N-ary relationships:

Include all primary keys from all entities. May add additional fields for descriptive attributes of the relationship. The primary key will be the union of all the foreign keys that are on the many side.

Subclasses and super classes(IsA):

Create a relation for both the super and subclass, the primary key for both will be the superclasses'.

Sub class relation:

create a relation for each subclass and append the superclass to each subclass. Primary key will be the superclasses'.

Has a relationship:

This is a recursive relation formed if an entity has multiple items. create a new foreign key in the relation that will reference the primary key.

Categories or unions:

Link the child to the shared parent primary key. If there is no common shared primary key additional keys are required.

Constraints

NOT NULL-	Attribute will not accept null values
UNIQUE-	Attribute cannot be duplicated in the table
Primary key-	Attribute will be a primary key Must be unique Only one primary key attribute per table May be simple or composite attribute
Foreign key-	Attribute is referencing a primary key of another table.
Check-	Attribute will be checked against condition If true the value may be inserted, otherwise rejected
Default-	Attribute will be set to a default value if no value given.

2.2 CONVERSATION TO RELATIONAL DATABASE

User (Converted from User entity)

Email (**Primary key**)– Domain: Varchar2(40)

NOT NULL

FirstName – Domain: Varchar2(25)

NOT NULL

MiddleInit – Domain: char(1)

LastName – Domain: Varchar2(30)

NOT NULL

Password – Domain: char(129)

NOT NULL

Phone – Domain: char(10)

AccountType – Domain: Integer

Constraint: Must be 0 (admin), 1 (moderator), or 2 (customer)

IsSubscriber – Domain: Integer

Constraint: Must be 1 or 0 (boolean)

Candidate keys: Email (PK), Phone

Address (Converted from address composite attribute in User entity)

AddressID (**Primary Key**) – Domain: Integer

Auto incrementing

AddressUser – Domain: Varchar2(40)

Foreign key to User.Username

NOT NULL

AddressLine1 – Domain: Varchar2(50)

NOT NULL

AddressLine2 – Domain: Varchar(25)

AddressCity – Domain: Varchar(30)

NOT NULL

AddressZip – Domain: char(5)

AddressState – Domain: Varchar(30)

AddressCountry – Domain: Varchar(30)

NOT NULL

Candidate keys: AddressID (PK)

Credit Card (Converted from Credit Card entity)

CardNumber (**Primary Key**) – Domain: char(16)

CardUser – Domain: varchar(30)

Foreign key to User.Email

NOT NULL

CardHolder – Domain: Varchar2(60)

NOT NULL

CardProvider – Domain: Varchar2(20)

NOT NULL

Must be from a list of credit card providers that we accept

CardExpirationDate – Domain: Date

NOT NULL

Candidate keys: CardNumber(PK)

Item (Converted from Item Entity)

ItemID (**Primary key**) – Domain: Integer

Auto Incrementing

ItemName – Domain: Varchar2(70)

NOT NULL

Category – Domain: Varchar(40)

Foreign key to Category.CategoryName

NOT NULL

ItemManufacturer – Domain: Varchar2(50)

NOT NULL

ItemPrice – Domain: Number

ItemStock – Domain: Integer

NOT NULL

ItemPhoto – Domain: Integer

ItemVideo – Domain: Integer

Candidate keys: ItemID (PK), ItemName

Category (Converted from Category entity)

CategoryID (**Primary key**) – Domain: Integer

CategoryName – Domain: Varchar2(40)

Description – Domain: Varchar2(1000)

Parent – Domain: Varchar2(40)

Foreign key to Category.CategoryName (recursive)

Order (Converted from Order Entity)

OrderID (**Primary key**) – Domain: Integer

OrderUser – Domain: Varchar2(25)

Foreign key to User.Email

NOT NULL

OrderItem – Domain: Integer

Foreign key to Item.ItemID

OrderShippingAddress – Domain: Integer

Foreign key to Address.AddressID

NOT NULL

OrderBillingAddress – Domain: Integer

Foreign key to Address.AddressID

NOT NULL

OrderCreditCard – Domain: char(16)

Foreign key to CreditCard.CardNumber

NOT NULL

OrderQuantity – Domain: Integer

NOT NULL

OrderPrice – Domain: Integer

NOT NULL

OrderDate – Domain: Date

NOT NULL

Candidate keys: OrderID (PK)

2.3 RELATIONAL INSTANCES

ITEM

ID	NAME	CATEGORY	MANUFACTURER	PRICE	STOCK	Photo	Video
259092743	Red LANYARD	32	INTEL	1.01	2	1	1
459727727	Green LANYARD	32	SWISS	5.50	43	1	1
659912201	8 GB FLASHDRIVE	13	AMD	20.00	31	0	0
130446979	Black COFFEE MUG	12	ASUS	10.00	22	1	0
675198870	Light Saber	7	SWISS	10.00	16	2	0
482691015	USB Lamp	8	ASUS	5.00	0	3	2
935601757	BACONAISE	22	SWISS	2.00	1	4	2
834413066	USB Toaster	15	ASUS	5.00	10	2	0
307585175	Nyan Cat Costume	10	VANS	10.90	5	1	0

ORDER

ID	EMAIL	ITEM_ID	SHIPADR	BILLADR	CREDIT_CARD	QTY	PRICE	DATE
48803864	dui.ae@rhonollis.com	51231232	51238	25123	5214950934212358	2	53.21	9/19/2011
44415799	Nu.ac.sem@enim.net	61231232	12346	13234	5819082387571928	3	86.32	5/18/2011
45866406	soes.purus@scelue.ca	61293873	12676	12167	1237589128732183	1	12.50	8/16/2012
84778124	nisi.a@orci.ca	90869311	51322	51727	6987234978523912	5	66.72	6/23/2011
93113729	Etiam.m@Quire.com	86891723	39051	32124	6128918392012395	4	13.55	9/30/2011
32881190	nisl.senquat@tincit.ca	67192837	55612	55325	6898123901918372	7	43.25	1/3/2012
12742237	elem@Pellentnttus.ca	79817231	12317	68128	8873829182643812	3	3.79	2/3/2011
62419702	mSe@Morumsan.com	69817231	44432	32129	0923891002137284	2	41.12	5/4/2011
65147128	Proin.ues@priisin.org	68723948	88657	88651	4413123890175892	1	8.52	6/3/2011

USER

EMAIL (username)	F_NAME	M_INIT	L_NAME	PASSWORD	PHONE	ACCNT_TYPE	IS_SUBSCRIBER
vel.quam@risusIn.ca	Akeem	T	Marny	VXW69F...	1-691-795-2366	0	0
neque.In.@endiget.ca	Anthony	H	Amery	RGJ53UQ...	1-953-142-3285	1	1
mattis@Aesed.com	Jack	D	Deirdre	OHX9UQR...	1-486-104-4923	2	1
montes@loblass.com	Lillian	G	Armando	UAI06IZM...	1-447-536-2087	2	1
feugiat@nn.ca	Adara	W	Ezekiel	ZJG93HLL...	1-905-862-9525	1	1
sagittis.sde@vul.com	Kylan	T	Raymond	GPO71XA...	1-525-211-8965	2	1
arer@lucttultrices.ca	Priscilla	M	Gabriel	LUO45OC...	1-711-304-5019	2	0
Praesent@euerat.edu	Neil	K	Mariam	EJR40ENC...	1-588-438-6050	1	0
maleda.fringilla@et.ca	Medge	B	Quentin	DQV9LUP...	1-426-269-3763	2	0

CREDIT CARD

CCN	EMAIL (user)	HOLDER	PROVIDER	EXP_DATE
5214950934212358	itor.tellus@hendrerit.ca	Carissa O Burke	DISCOVER	Dec-11
5819082387571928	turpis@nollisvitae.ca	Dillon F Moses	AMERICAN EXPRESS	Jul-12
1237589128732183	coad@Vestibum.edu	Shaeleigh C Michael	AMERICAN EXPRESS	Mar-14
6987234978523912	lestie@SetumProin.ca	Xyla B Daugherty	DISCOVER	Aug-13
6128918392012395	phareed@ssnec.com	Melissa K Reyes	AMERICAN EXPRESS	Apr-14
6898123901918372	hymuris@arcuet.ca	Lucian X Diaz	VISA	Feb-14
8873829182643812	suscsce@lictum.com	Bell W Lawson	MASTER CARD	Aug-14
0923891002137284	ante@senNullam.ca	Jordan P Steele	VISA	Apr-12
4413123890175892	tincidunt@Inat.ca	Marsden V Lee	MASTER CARD	Oct-11

ADDRESS

ID	USER	LINE1	LINE2	CITY	ZIP	STATE	COUNTRY
51232	nequmnec.org	P.O. Box 353, 3508 Purus Rd.	NULL	La Habra Heights	92706	PA	Tanzania
73122	urns@mlit.edu	907-1501 Magna. Ave	P.O. Box 374	Olympia	56318	NULL	Belarus
67123	In@ieSed.edu	9230 At Street	NULL	Rye	92564	CO	Turkey
41236	lobtis@egsa.edu	P.O. Box 119, 7311 Urna Av.	NULL	Miami	53525	NULL	France
83412	Nullm@ma.org	P.O. Box 621, 5798 Sit Street	NULL	Alamogordo	35908	RI	Georgia
31273	In.at@idblit.edu	Ap #333-3746 Amet, Av.	NULL	Arcadia	79534	PA	Antigua
85562	nulla@Quas.edu	3808 FacilisisRd.	P.O. Box 432	Pocatello	85537	NULL	Kiribati
37854	iacis.nec@are.com	Ap #929-5730 Dui. Av.	NULL	Batavia	85525	WA	United States
00765	faus.ut@bidum.ca	Ap #773-9898 Penatibus Av.	NULL	Chicopee	56471	NULL	Armenia

CATEGORY

ID	NAME	PARENT	DESCRIPTION
0	Apparel		Because you have to put on some clothes to go outside
1	Computer Stuff		The man who dies with the most toys wins
2	Gadgets		Inspect this
4	Snacks		The narwhal bacons at midnight
5	T-Shirts		That thing you wear, on your chest
6	Linux	5	Stallmanian fashion
7	USB Devices	1	Put that 24-port USB hub to use
8	Science	5	Nothing to see here
9	Bacon	4	Because everyone loves bacon
10	Watches	2	For those who dont have a cell phone

2.4 QUERIES IN PLAIN ENGLISH

i. Find the most expensive orders

ii. List orders placed by jdoe@csub.edu between 9/1/2011 and 9/30/2011

iii. Find customers emails that have placed at least two orders

iv. List user emails that purchased a shirt in the first three months of 2011

v. Find orders placed by jdoe@csub.edu where the shipping and billing address are not the same and that were paid with a Visa card

vi. Find the second cheapest items in the Clothing category

vii. List users who have ordered every item

viii. List user emails who own at least two credit cards

ix. Find customer names that have never ordered item with ID 3124

x. Find items that have never been purchased

2.5 QUERIES IN RELATIONAL ALGEBRA AND CALCULUS

i. Find the most expensive orders

cheap ← σ((Order(o1) X Order(o2)) ^ o1.price < o2.price)
mostExpensive ← π(o.oid, o.price)(σ(Order – cheap))

{o1 | Order(o1) ^ (∀o2)(Order(o2) → o1.oid != o2.oid ^ o1.price >= o2.price}

{<i> | Order(i1, _, _, _, _, _, _, p1, _) ^ (∀o2)(Order(i2, _, _, _, _, _, _, p2, _) →
i1 != i2 ^ p1 >= p2}

ii. List orders placed by jdoe@csub.edu between 9/1/2011 and 9/30/2011

π(o.id) (σ(Order(o) ^ o.user='jdoe@csub.edu' ^
o.date >= 9/1/2011 ^ o.date <= 9/30/2011)))

{o | Order(o) ^ (∃u)(User(u) ^ u.email='jdoe@csub.edu' ^ u.email=o.user ^
o.date >= 9/1/2011 ^ o.date <= 9/30/2011}

{<i> | Order(i, 'jdoe@csub.edu', _, _, _, _, _, _, d) ^ (∃u)(User('jdoe@csub.edu',
_, _, _, _, _, _, _) ^ e1='jdoe@csub.edu' ^
u=e ^ d>= 9/1/2011 ^ d <= 9/30/2011) }

iii. Find customers emails who have placed at least two orders

π(u.email)(σ((User X Orders) ^ o1.user=o2.user ^ o1.id != o2.id ^
u.type='customer'))

{u | User(u) ^ (∃o1)(∃o2)(u.type='customer' ^ u.email=o1.user ^ o1.user=o2.user
^ o1.id!=o2.id) }

{<e> | User(e, _, _, _, _, _, 'customer', _) ^ (∃o1)(∃o2)(Order(i, e,_,_,_,_,_,_,_) ^
Order(i, e, _, _, _, _, _, _, _)) }

iv. List user emails who purchased a shirt in the first three months of 2011

π(o.user)(σ((Item(i) X Order(o)) ^ i.category='shirt' ^ o.item=i.itemid ^
o.date>=1/1/2011 ^ o.date<=3/30/2011) }

{o.user | User(u) ^ (∃i)(Item(i)(∃o)(Order(o)(i.category='shirt' ^

o.item=i.id ^ o.date>=1/1/2011 ^ o.date<=3/30/2011)) }

{<u> | (∃i)Item(i, _, 'shirt', _, _, _, _, _) ^ (∃o)Order(o, u, i,_,_,_,_,_, d) ^
d>=1/1/2011 ^ d<=3/30/2011) }

v. Find orders placed by jdoe@csub.edu that was paid with a visa card and the shipping and billing address are not the same

π(o.id) (σ((Order(o) X CreditCard(c))^ o.user='jdoe@csub.edu'
^ o.shipAddr!=o.billAddr ^ c.number=o.cardnumber ^ c.provider='Visa') }

{<o> | Order(o) ^ (∃c)(CreditCard(c)(c.number=o.cardnumber ^ c.provider='Visa'
^ o.user='jdoe@csub.edu' ^ o.shipAddr != o.billAddr)) }

{<o> | Order(o, 'jdoe@csub.edu', _, ba, sa, c, _, _, _) ^
(∃c)(CreditCard(c, 'jdoe@csub.edu', _, 'Visa', _) ^ ba != sa) }

vi. List the second cheapest items in the Snacks category
expensive ← σ((i2.price > i1.price)(item i2 X item i1) ^ i1.category='clothing ^
i2.category = 'clothing'))
expensive2 ← σ((expensive e1 X expensive e2) ^ e2.price > e1.price)
2ndCheapest ← π(i.name)((Item ^ i.category='clothing') *
π(i.id)(σ(Item – expensive2) – expensive))

{<i> | Item(i) ^ (∃i)(Item(i) ^ i.category='clothing' ^ (∃i2)(Item(i2) ^ i2.price<i.price ^
~(∃i3)(Item(i3) ^i3.price < i.price ^ i2.price != i3.price)) }

{<i, n> | Item(i, n, 'clothing',_,p1,_,_,_) ^ (∃p2)(Item(_,_,'clothing',_,p2,_,_,_) ^
p2 < p1 ^ ~(∃p3)(Item(_,_,'clothing',_,p3,_,_,_) ^ p3 < p1 ^ p2 != p3)) }

vii. List users who have ordered every item

π(ItemID, user)(Order) / π(ItemID)(Item)

{u | user(u) ^((∀i)Item(i) → (∃o)order (o)) ^ o.user=u.email ^ o.item = i.item}

{<u>|User(u,_,_,_,_,_,_,_,_)^(∀i)Item(i,_,_,_,_,_,_,_) → (∃o)Order(o,u,_,_,_,i,_,_)}

vii. List user emails who own at least two credit cards

π(u.email)(σ(CreditCard(c1) X CreditCard(c2) ^ c1.ccn != c2.ccn ^ c1.user=c2.user))

{c1.user| (∃c1) credit_card(c1) ^ (∃c2) credit_card(c2) ^
c1.ccn !=c2.ccn ^ c1.user = c2.user}

{<u> | (∃c1) credit_card(c1,u,_,_,_) ^ (∃c2)credit_card(c2,u,_,_,_) ^ c1 != c2 }

ix. Find customer names that have never ordered item with ID 3124

π(u1.fname, u1.lname)(User u1 - (σ((User u X Orders o) ^
u.accnt_type='customer' ^ u.email = o.orderuser') ^ o.orderitem = 3124))

{u.fname, u.lname | User(u) ^ u.type = 'customer' ^ ~(∃o)(order(o) ^
u.email=o.orderuser ^ o.orderItem=3124)}

{<f, l>| (∃u1)User(u1,f, _, l, _, 'customer', _) ^ ~(∃o)Order(o,u1,_,_,_,3124,_,_))}

x. Find items that have never been purchased

π(i.ItemID)(σ(Item(i).ItemID – Order(o).OrderItem))

{i.ItemID | ItemID(i) ^ ~(∃o)(OrderID(o) ^ o.item == I.ItemID)}

{<i,n>|ItemID(i,n,_,_,_,_,_,_) ^ ~(∃o)(Order(o,_,i,_,_,_,_,_,_))}

Phase III

3.1 NORMALIZATION OF RELATIONS

First Normal Form (1NF)

The first normal form only allows values that are simple and single. This means that there cannot be relations within relations or relations as attribute values within tuples.

Second Normal Form (2NF)

For a relation to be 2NF, it must be 1NF as well. Additionally, every non-prime attribute must be fully functionally dependent on the primary key. You only need to test for it if the primary key is more than one attribute. A relation schema can be second normalized into a number of 2NF relations by associating nonprime attributes with only part of the primary key that they are fully functionally dependent on.

Third Normal Form (3NF)

The third normal form is present when there is no transitive dependency of a non-key attribute on the primary key. To normalize, break up the relation and set up relations that include the non-key attributes that functionally determine other non-key attributes.

Boyce-Codd Normal Form (BCNF)

A relation R is in Boyce-Codd Normal Form when a nontrivial functional dependency X → A holds in R and X is a super-key of R. All BCNF relations are also 3NF, but not all 3NF relations are BCNF.

Problems with Normalization

Modification anomalies can occur when updating an attribute that is functionally dependent on the primary key. If you change that value, you will have to change all values if it is not normalized.

Normal Forms of Relations

All of the relations within our database are in 2nd normal form.

Modification Anomalies

Second normal form relations are susceptible to update anomalies. this implies that a column has data in multiple tuples that are the same. When one goes to update the data in one tuple the other one should be updated also but remains as its original value.

Normalization

Since all relations are only in first normal form, there is no way to convert them to the other forms.

3.2 SQL*PLUS

SQL*PLUS is a command-line utility created by Oracle that can run SQL and PL/SQL commands interactively or from a script file. The first incarnation of the program was called UFI (“User Friendly Interface”). It then became Advanced UFI after some more features were added. Then its name was changed to SQL*PLUS.

3.3 SCHEMA OBJECTS

Table

Tables are basic database objects that store data in rows and columns. Each column stores data for a single attribute and each row stores data for a single record.

Syntax:

```
CREATE TABLE table_name (
column1         datatype        null/not null,
column2         datatype        null/not null,
...
CONSTRAINT constraint_name PRIMARY KEY (column1, column2, . column_n)
);
```

Tables in our database:

GPJH_USER

GPJH_ITEM

GPJH_CC

GPJH_ADDR

GPJH_CATEG

GPJH_ORDER

View

Views are virtual tables that do not actually store any data. Views are often stored procedures that display certain attributes from one or more tables, or even use aggregate functions and PL/SQL to generate sets of interest.

Syntax:

```
CREATE VIEW view_name AS
SELECT columns
FROM table
WHERE predicates;
```

Views in our database:

GPJH_ORDERS_2011Q1

GPJH_SNACKS

Index

An index is a stored copy of one or more columns of a table. They are used to improve the speed of retrieving data, but with the downside of increased storage space and slower writes.

Syntax:

```
CREATE [UNIQUE] INDEX index_name
ON table_name (column1, column2, . column_n)
```

Indexes in our database:

GPJH_CATEG_NAME

Sequence

Sequences are used to create a sequence of numbers, which are often used to auto-increment primary key attributes. However, this may not always be the case. In addition, sequences do not have to increment by 1; they can be whatever number specified.

Syntax:

```
CREATE SEQUENCE sequence_name
   MINVALUE value
   MAXVALUE value
   START WITH value
   INCREMENT BY value
   CACHE value;
```

Clusters

A cluster is a group of one or more tables that are physically stored together because they share common columns that are often used together. The goal of clustering is to reduce disk access time, which is the number one bottleneck for DBMS software.

3.4 RELATION SCHEMA AND CONTENT

User

```
CS342 SQL> DESC GPJH_USER;

Name                                              Null?   Type
------------------------------------------------- -------- ---------------------------------
EMAIL                                             NOT NULL          VARCHAR2(50)
FNAME                                             NOT NULL          VARCHAR2(40)
MINIT                                             NOT NULL                CHAR(1)
LNAME                                             NOT NULL          VARCHAR2(40)
PASSWORD                                          NOT NULL               CHAR(40)
PHONE                                             NOT NULL               CHAR(12)
ACCOUNTTYPE                                       NOT NULL               NUMBER
ISSUBSCRIBER                                      NOT NULL               NUMBER


CS342 SQL> select * from gpjh_user;

EMAIL          FNAME      M LNAME      PASSWORD       PHONE       ACCOUNTTYPE ISSUBSCRIBER
--------------- ---------- - ---------- --------------- ------------ ----------- ------------ ----------------------- ----------------------
alus@actas.edu  Grady      H Fulton     XEA84MSG6XC     1406990445          1          1
tpis@equet.edu  Nissim     N Kerr       AKR02ENY4EV     1558147292          2          0
ante@rimis.com  Shea       L Sellers    JLF04LFA3QE     1217032851          0          0
Don@puereat.ca  Whitney    C Dudley     ULS04FLR2VI     1304107393          1          0
```

```
rlis@rtor.com   Velma     L Joyner     EFY82KCR0IN     1375886625           0          1
luaam@lla.com   Dillon     Q Rutledge   TVA67GCV9EE     1458947609           0          0
erus@Mis.org    Hermione   V Chaney     FPZ22TMJ5KN     1780674335           0          1
sera@nec.org    Tatum      D Graham     BIY05FUA5JK     1012170604           0          0
cot@vite.org    Rhona      H Quinn      EJB04QHF6WP     1096304364           2          1
tm.eu@nt.edu    Hollee     V Joseph     DZW20HWJ5NG     1051781128           1          1
var@blras.com   Uriah      Z Douglas    KCG23INF7AS     1912091213           0          1
Cras@mris.com   Kessie     Z Huff       UJO06XJE1BF     1893002233           1          1
meus@estie.ca   Xena       T Torres     LOH95XXY0AZ     1289680804           0          1
vunt@adio.com   Ivana      L Delacruz   JWU68JOT2QT     1295254335           2          1
14 rows selected.
```

Category

```
CS342 SQL> DESC GPJH_CATEG;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 CATEG_ID                                  NOT NULL NUMBER
 NAME                                      NOT NULL VARCHAR2(50)
 PARENT                                             NUMBER
 DESCRIPTION                                        VARCHAR2(255)

CS342 SQL> select * from gpjh_categ;

ID         NAME            PARENT     DESCRIPTION
---------- --------------- ---------- ------------------------------------------------------------
         0 Apparel                    Because you have to put on some clothes to go outside
         1 Computer Stuff             The man who dies with the most toys wins
         2 Gadgets                    Inspect this
         3 Snacks                     The narwhal bacons at midnight
         4 Books                      For those who... read
         5 Kids                      Your kid can be a geek too!
         6 T-Shirts              0       That thing you wear, on your chest
         7 Linux                 6       Stallmanian fashion
         8 USB Devices           1       Put that 24-port USB hub to use
         9 Science               6       Nothing to see here
        10 Gaming                6       Show off your skillz
        11 Bacon products        3       Because everyone loves bacon
        12 Watches               2       For those who dont have a cell phone

13 rows selected.
```

Credit Card

```
CS342 SQL> DESC GPJH_CC;
 Name                                                      Null?    Type
 ----------------------------------------------------------------- -------- --------------------------------------------
 CCN                                               NOT NULL CHAR(16)
 CARDUSER                                          NOT NULL VARCHAR2(50)
 HOLDER                                            NOT NULL VARCHAR2(65)
 PROVIDER                                          NOT NULL VARCHAR2(10)
 EXPDATE                                           NOT NULL CHAR(5)
CS342 SQL> select * from GPJH_CC;

CCN              CARDUSER        HOLDER               PROVIDER   EXPDA
---------------- --------------- -------------------- ---------- -----
9907938645811588 vunt@adio.com   Ora R. Ryan          Visa       01/12
9806706776162620 meus@estie.ca   Florence O. Arnold   Visa       09/12
2721329487444206 Cras@mris.com   Zephr Y. Harrell     Visa       12/14
2512140453757532 var@blras.com   Emery T. Hampton     Visa       10/13
```

```
7762398410815452 tm.eu@nt.edu    Ruth Z. Collins       Visa       08/13
2336013647521730 cot@vite.org    Linus X. Potts        Mastercard 09/12
7126859229366731 sera@nec.org    Raymond D. English    Mastercard 11/11
7927299567059352 erus@Mis.org    Anastasia F. Saunders Discover   12/13
0752031303399979 luaam@lla.com   Dai Y. Colon          Visa       04/16
9374771392605749 Don@puereat.ca  Hilary R. Bonner      Discover   09/15
9694197835558575 ante@rimis.com  Aladdin C. Lane       Visa       10/13
5559556497466722 alus@actas.edu  Haley X. Gamble       Visa       12/15

12 rows selected.
```

Address

```
CS342 SQL> DESC GPJH_ADDR;
 Name                                                            Null?     Type
 ----------------------------------------------------------------- --------- --------------------------------------------
 ADDR_ID                                                         NOT NULL  NUMBER
 ADDR_USER                                                       NOT NULL  VARCHAR2(50)
 LINE1                                                           NOT NULL  VARCHAR2(50)
 LINE2                                                                     VARCHAR2(30)
 CITY                                                            NOT NULL  VARCHAR2(50)
 STATE                                                           NOT NULL  VARCHAR2(20)
 ZIP                                                             NOT NULL  VARCHAR2(10)
 COUNTRY                                                         NOT NULL VARCHAR2(40)

CS342 SQL> select * from gpjh_addr;
  ID     USER               LINE1              LINE2     CITY            STATE     ZIP          COUNTRY
---------- --------------------- --------------------- ------------- ------------------ ------- ------------- -------------------
     232 tpis@equet.edu  Ap #783-2037 …               Bakersfield      YT     36581       Hungary
     451 tpis@equet.edu  Ap #934-1151 …               Stockton         MB     59102       Swaziland
```

```
    954 rlis@rtor.com      Ap #626-3972 …              Lock Haven      NC     59102       Poland
    151 erus@Mis.org       862-4600 Nun…               Hanahan         TX     32190       Belize
    792 cot@vite.org        2229 Velit. St.            Claremore       NT     63109       Guam
     21 var@blras.com     1179 Nulla. Av.              Harrisburg      PE     30950       Japan
    908 Cras@mris.com    445-3761 ….                   New York        NL     86012331 Kiribati
     18 meus@estie.ca     138 Tristique Rd.            Ogden           KY     601923      Palau
      562 vunt@adio.com   555-9099 ...   PO Box 12   Bowie           AL     96312       Morocco
      550 cot@vite.org      8325 Conval…               Avalon          NB     003214      Nigeria
       75 vunt@adio.com   2908 Arcu…                   Texas City      NM     95810       USA
      178 sera@nec.org     Ap #522-4…                  Olean           PE     673910      Tunisia

12 rows selected.
```

Item

```
CS342 SQL> DESC GPJH_ITEM;
Name                                                            Null?          Type
 --------------------------------------------------------------- -------------- -----------------------------------------
 ITEM_ID                                                        NOT NULL NUMBER
 NAME                                                           NOT NULL VARCHAR2(55)
 CATEGORY                                                                NUMBER
 MANUFACTURER                                                   NOT NULL VARCHAR2(255)
 PRICE                                                                   VARCHAR2(50)
 STOCK                                                                   VARCHAR2(50)
 PHOTO                                                          NOT NULL VARCHAR2(50)
 VIDEO                                                          NOT NULL VARCHAR2(50)

CS342 SQL> select * from gpjh_item;
 ID        NAME            CATEG MANUF.     PRICE    STOCK PHOTO VIDEO
---------- ----------------- --------- ---------------- ------------ ---------  ---------- -----
```

```
1048 Portal 2 Poster   6        Valve          355.58   292        2     2
582 USB Toaster        8        IntelliTech    448.81   143        2     0
9840 Pip-boy           2        Bethesda       322.42   489        3     0
9212 Nuke-Cola         3        Bethesda       209      609        2     0
9251 Portal boots      0        Valve          1        183        1     0
856 Gravity Gun Toy    5        Valve          363      602        1     2
9570 Wizard hat        0        Wizards San. 494        494        0     2
6358 Spy camera        2        ACME           295      160        3     2
831 Wifi hat           0        IntelliTech    426      82         4     1
7741 Balance           2        Blizzard       130      13         3     1
8302 xkcd joke #392    7        xkcd           12       717        2     0
2709 Headcrab lotion   3        Valve          134      751        3     2

12 rows selected.
```

Order

```
CS342 SQL> DESC GPJH_ORDER;
Name                                                   Null?  Type
------------------------------------------------------ -------- ------------------------------------
O_ID                                           NOT NULL NUMBER
O_USER                                         NOT NULL VARCHAR2(50)
O_ITEM                                         NOT NULL NUMBER
S_ADDR                                         NOT NULL NUMBER
B_ADDR                                         NOT NULL NUMBER
CC                                             NOT NULL CHAR(16)
QTY                                            NOT NULL NUMBER
PRICE                                          NOT NULL NUMBER
O_DATE                                                  DATE
```

```
CS342 SQL> SELECT * FROM GPJH_ORDER;

O_ID O_USER              O_ITEM S_ADD B_ADD   CREDIT CARD        QTY    PRICE   O_DATE

------- ----------------------- -----------  ---------- -----------  --------------------------  ---------- ---------- ------------

3512 jdoe@csub.edu    1048       954       954       2721329487444206   4        48.32    19-JUN-11

3712 ante@rimis.com    582        232       232       2336013647521730   7        218.46   19-JUN-11

3112 Don@puereat.ca   9251        451     451       9806706776162620   1        482.32   19-JUN-11

7312 erus@Mis.org      9570        151     151      7927299567059352   2        90.45     04-DEC-10

8318 sera@nec.org       9212        21       21       9374771392605749   1        32.88     29-FEB-12

1337 sera@nec.org       856          21       21       9694197835558575   1        9.59       29-FEB-12

9001 sera@nec.org       7741        21       21       9694197835558575   3        390.59   29-FEB-12

6523 var@blras.com     8302        908     562      7927299567059352    9       159.14    12-JUL-11


8 rows selected.
```

3.5 QUERIES

-- #1 Most expensive order

```
SELECT  o1.O_ID,o1.O_Item,o1.Price,o1.O_Date
FROM    GPJH_ORDER o1
WHERE   not exists (
   select *
   from GPJH_ORDER o2
   WHERE (o2.Price > o1.price and o1.O_ID <> o2.O_ID)
);
```

-- #2 lists orders placed by jdoe@csub.edu between 9/1/2011 and 9/30/2011

```
SELECT O.O_ID, O.O_USER, O.O_ITEM
FROM GPJH_USER U, GPJH_ORDER O
WHERE U.EMAIL = 'jdoe@csub.edu' AND U.EMAIL = O.O_USER
```

```
AND O.O_DATE >= to_date('2011-09-01','yyyy-mm-dd')
AND O.O_DATE <= to_date('2011-09-30','yyyy-mm-dd');
```

-- #3 Find customers emails who have placed at least two orders

```
select distinct u.email
from    gpjh_user u, gpjh_order o, gpjh_order o2
where   u.accounttype = 2 and u.email = o.o_user and
        o.o_user = o2.o_user and o.o_id != o2.o_id;
```

-- #4 List user emails who purchased a T-Shirt in the first three months of 2011

```
SELECT  distinct O.O_USER FROM GPJH_ORDER O,GPJH_ITEM I,GPJH_CATEG C
WHERE   C.NAME = 'T-Shirts' AND I.CATEGORY = C.CATEG_ID AND
        O.O_ITEM = I.ITEM_ID AND
        O.O_DATE >= to_date('2011-01-01','yyyy-mm-dd') AND
        O.O_DATE <= to_date('2011-03-30','yyyy-mm-dd');
```

-- #5 Find orders placed by jdoe@csub.edu that was paid with a visa card and the shipping and billing address are not the same

```
SELECT  O.*
FROM    GPJH_CC C, GPJH_ORDER O
WHERE   C.PROVIDER like 'Visa' AND C.CCN = O.CC AND
        O.O_USER = 'jdoe@csub.edu' AND O.S_ADDR <> O.B_ADDR;
```

-- #6 List the second cheapest items in the Snacks category

```
select  i.item_id,i.name,i.price
from    gpjh_item i, gpjh_item i2, gpjh_categ c
where   c.name='Snacks' and i.category=c.categ_id and
        i2.category=c.categ_id and i2.price < i.price and
        not exists (
            select     *
```

```
          from        gpjh_item i3
          where       i3.category=c.categ_id and
                      i3.price < i.price and i2.price != i3.price);
```

-- #7 List users who have ordered every item

```
select  unique u.email
from    gpjh_user u
where   not exists (
             select *
             from gpjh_order o
             where not exists (
                select      *
                from        gpjh_item i
                where       o.o_item = i.item_id and
                            u.email = o.o_user
                         )
             );
```

-- #8 List user emails that own at least two credit cards

```
select  distinct c1.carduser
from    gpjh_cc c1, gpjh_cc c2
where   c1.carduser = c2.carduser and c1.ccn != c2.ccn;
```

-- #9 Find customer names that have never ordered an item with ID 3124

```
select  u.fname,u.minit,u.lname
from    gpjh_user u
where   not exists (
   select *
   from       gpjh_order o
```

```
    where       u.email = o.o_user and
                o.o_item = 3124
            );
```

-- #10 selects items that have never been purchased

```
select i.item_id,i.name
from GPJH_ITEM i
where not exists (
    select * from GPJH_ORDER o
    where i.item_id = o.o_item);
```

-- #11 List cities where there are in at least 2 addresses, group by city

```
select  city, count(*) as "# of cities"
from    gpjh_addr
group by city
having  count(city) >= 2;
```

-- #12 Find the cheapest price of an item manufactured by Valve

```
select  min(price)
from gpjh_item
where manufacturer = 'Valve';
```

-- #13 Create table from orders in 2011 and only projecting item, quantity, and price

```
create table gpjh_2011orders as
        select  o.o_item,i.name,o.qty,o.o_date
        from    gpjh_order o,gpjh_item i
        where   o.o_item = i.item_id and
                o.o_date >= to_date('2011-01-01','yyyy-mm-dd') and
                o.o_date <= to_date('2011-12-31','yyyy-mm-dd');
```

3.6 DATA LOADER

Data Loading Methods

The most rudimentary way to load data into a database in Oracle is with INSERT INTO … VALUES (…). In Oracle, only one row can be inserted at a time with this method. However, one can also use INSERT INTO … SELECT (…) FROM (…) to use the result set of a select statement as input for another table. Oracle also provides some utilities for loading data.

Oracle Data Pump is a feature of Oracle Database 11g/Release 2 that provides high speed import and export utilities and a web-based interface. Oracle Data Pump also boasts several other features that improve data loading. SQL*Loader is also a fast data loading utility from Oracle that loads from external files into tables. It can accept many formats of input, perform filtering, and load into multiple tables at once. External Tables is another Oracle utility that provides a preprocessor to increase flexibility of input formats.

Java Dataloader

The Java Dataloader program was created by Dr. Huaqing Wang, Professor of Computer Science at California State University of Bakersfield. Its purpose is to read formatted data from a text file into Oracle database tables. Dr. Wang lets students in his Database Systems class modify the program to make it more user friendly. We have not done this (yet). Data must be inputted into the program in the following format:

TABLENAME | *tableName* | *numberOfColumns*
row1col1value | *row1col2value* | … | *row1colNvalue*
row2col1value | *row2col2value* | … | *row2colNvalue*
…

Phase IV

4.1 FEATURES OF PL/SQL AND TRANSACT-SQL

PL/SQL (Procedural Language/Structured Query Language) is an extension for SQL and Oracle's relational database that provides many procedural language features. Its syntax is similar to Ada or Pascal, and it supports variable declaration, arrays, loops, conditional statements, exception handling, and object-oriented features.

Transact-SQL (T-SQL) is a proprietary extension for SQL created by Microsoft and Sybase that is provides a similar functionality for Microsoft SQL Server as PL/SQL does for Oracle relational databases. Unlike PL/SQL, it has made changes to the DELETE and UPDATE statements. All applications must communicate with a Microsoft SQL Server by sending T-SQL statements, no matter what interface it uses.

The purpose of both of these extensions is to provide benefits like better design structure, security, and performance. They both allow for the creation of stored subprograms, which provide both of these benefits. There is a performance increase because queries do not have to be compiled when calling stored subprograms. Parameters can simply be passed to the existing procedure, which reduces network traffic and improves CPU performance. These programs also promote better design practices because business rules can simply be stored in stored procedures or triggers so that frontend application programmers can focus on other issues.

Although PL/SQL and T-SQL both aim to achieve similar goals and share some common features, there are many differences as well. Besides the obvious syntax differences, the two extensions differ in many of the features they provide. PL/SQL allows for the creation of packages, but T-SQL has no equivalent to this. PL/SQL also uses %TYPE, which allows flexibility and portability because the datatype of an attribute can substitute as the datatype of a variable or another attribute. In some cases, T-SQL is much simpler than PL/SQL. For instance, sequences are not needed in T-SQL because you can simply add an auto-increment clause for an attribute in a create statement. Also, T-SQL SELECT statements can be put just about anywhere without the need for temporary variables as placeholders.

4.2 ORACLE PL/SQL

Program Structure

Pl/SQL program structure is based on code blocks. There are three basic parts to a code block: declaration, execution, and exception handling, though not all parts are required for every code block. Variables are declared after DECLARE statement. Commands are executed after the BEGIN statement. Additional statements such as control statements and loops are optional. Exceptions are handled for re-thrown after the EXCEPTION statement. A code block is terminated with END.

Code block syntax:

```
DECLARE [label]
    <variable name>      <datatype>
    […]
BEGIN
    Statements
[EXCEPTION]
    EXCEPTION handlers
END [label];
```

Control statement syntax:

```
IF <condition> THEN <statements>
ELSEIF <condition> THEN <statements>
ELSE <Statements>
END IF;
```

Loop syntax:

```
FOR <variable> in <lower bound> .. <upper bound> LOOP
    <statements>
END LOOP;
```

Exception syntax:

```
[EXCEPTION]
    WHEN <exception name> THEN <statements>
END;
```

Stored Procedures

A stored procedure is a pre-compiled procedure that performs actions on the database or makes a query, and they can be called by application software. They increase performance because less data needs to be sent when calling a stored procedure and it is already compiled.

Creation Syntax:

```
CREATE [OR REPLACE] PROCEDURE <procedure name>
    [(<variable> IN|OUT <datatype>, ...)] -- list of arguments
AS|IS
    [variable declarations]
BEGIN
    <statements>
[EXCEPTION]
    WHEN <exception name> THEN <statements>
```

```
        END;
```

Execution syntax:

```
        EXEC <procedure name>([arguments]);
```

Stored Functions

A stored function is very similar to a stored procedure, except that it must return a single data type.

Syntax:

```
        CREATE OR REPLACE FUNCTION <functionName>
                [(<variable> IN|OUT <datatype>,…)] -- list of arguments
        IS|AS
                [variable declarations]
        BEGIN
                <statements>
        [EXCEPTION]
                WHEN <exception name> THEN <statements>
```

Package

A package is a collection of schema objects (procedures, functions, etc.) similar to classes in popular object-oriented languages. A package requires a prototype that declares the schema objects used and a body that defines what the schema objects do.

Syntax:

```
CREATE PACKAGE <package name> AS

    <OBJECT TYPE> <name>(arguments);

    …

END <package name>

CREATE PACKAGE BODY <package name> AS

    <object definition (see create procedure or create function syntax)>

    …

END <package name> ;
```

Triggers

Triggers are created to automate business rules on a database. They are automatically called when a specified record or attribute is inserted, updated, or deleted. Triggers can check whether data meets certain conditions, save changes to a log, and make changes to other tables, among many other things.

Syntax:

```
CREATE [OR REPLACE] TRIGGER <trigger name>

BEFORE|AFTER      INSERT|DELETE|UPDATE [OR INSERT|DELETE|UPDATE]

OF COL <column_name>

ON <table name>

[DECLARE

    <variables>]

BEGIN
```

```
            FOR EACH ROW
                    [WHEN <condition>]
                    <statements>;
    END
```

4.3 ORACLE PL/SQL SUBPROGRAMS

Stored Procedures

GPJH_PROC_insertUser

This stored procedure is called to insert a new user. When calling it, you must pass every attribute of the user table.

```
-- insert user record

CREATE OR REPLACE PROCEDURE gpjh_proc_insertUser (
   user_email          in      gpjh_user.email%TYPE,
   user_fname          in      gpjh_user.fname%TYPE,
   user_minit          in      gpjh_user.minit%TYPE,
   user_lastName       in      gpjh_user.lname%TYPE,
   user_passwd                 in      gpjh_user.password%TYPE,
   user_phone          in      gpjh_user.phone%TYPE,
   user_accttype       in      gpjh_user.accounttype%TYPE,
   user_subscribe      in      gpjh_user.issubscriber%TYPE
)
```

```
IS
BEGIN
    INSERT INTO gpjh_user (
            email,
            fname,
            minit,
            lname,
            password,
            phone,
            accounttype,
            issubscriber
    )
    VALUES (
            user_email,
            user_fname,
            user_minit,
            user_lastname,
            user_passwd,
            user_phone,
            user_accttype,
            user_subscribe
```

```
   );


   COMMIT;

   EXCEPTION

   WHEN others THEN

         ROLLBACK;

         raise_application_error(-20998, sqlcode || ' : ' || sqlerrm);

         COMMIT;


END;

/
```

GPJH_PROC_deleteUser

This stored procedure deletes a single user record based off of the email (primary key) passed as an argument to the procedure. However, all records that depend on that user (orders, credit cards, addresses) are deleted first.

```
create or replace procedure gpjh_proc_deleteUser (

   user_email          in        gpjh_user.email%TYPE

)

IS

BEGIN
```

```
    -- remove dependents first
    delete
    from gpjh_order
    where o_user = user_email;
    delete
    from gpjh_cc
    where carduser = user_email;
    delete
    from gpjh_addr
    where addr_user = user_email;
    delete
    from gpjh_user
    where email = user_email;

    commit;
    exception
    when others then
        rollback;
        raise_application_error(-20998, sqlcode || ' : ' || sqlerrm);
        commit;
END;
```

```
/
```

Functions

GPJH_FUN_avgHighPrice

This calculates the average of the top N highest priced items. When calling this function, pass an integer N (default 10 if not passed) as the number of records you want to select from the most expensive items in the database. The function will return the average of those items.

```
create or replace function gpjh_fun_avgHighPrice (n in number default 10)

return number

is

        avg_price number;

begin

   with orderByHighest as

   (

        select price

        from gpjh_order

        order by price desc

   )


   select avg(price)

   into avg_price

   from orderByHighest
   where rownum <= n;


   return avg_price;


   exception
   when others then
         raise_application_error(-20998, sqlcode || ' : ' || sqlerrm);
end gpjh_fun_avgHighPrice;
/
```

Triggers

GPJH_TRIG_logItemChange

This trigger logs when an item in the database is updated or deleted into the table GPJH_logTable. It saves the old price and ID and the new price and ID into the table.

```
create or replace trigger gpjh_trig_logItemChange
before update or delete
on gpjh_item
for each row
begin
   insert into gpjh_logTable (
         oldVal,
         newVal
```

```
  ) values (
      :old.item_id || ' ' || :old.name,
             :new.item_id || ' ' || :new.name
  );


  exception
  when others then
      rollback;
      raise_application_error(-20998, sqlcode || ' : ' || sqlerrm);
      commit;
end;
/
```

Phase V

5.1 DAILY ACTIVITIES OF THE USER GROUPS

Each user is constrained by their account type. Actions allowed by each user are as follows:

A. Admin

Admins have more control of the site and may add items to the listings along with generate statistical data of usage and finances.

Administrators (Account type 0) are able to change/remove accounts and items, create accounts for new admins and moderators, and view all data on a given user. An admin may also generate reports on important data such as items with the most sales, items with back orders, profits for a given time period, and geographical data of users.

B. Moderator

Moderators are a small but necessary group in which most provide assistance for the customers. Moderators are allowed to change customer data and remove accounts of customers, and view account history of the customers. They do not have as much privilege or responsibility as an admin.

C. Customer

Customers are the general user base. Customers may only view data relevant to their own account as well as all items for sale in the database. Printable reports on complete account history are available to each customer. The report includes all items purchased, the date purchased, and also the order id. Customers are allowed to add remove and change account information should specific data need to be updated. If a customer places an order and decides they no longer want the item it may be edited or canceled if it has not shipped yet.

5.2 RELATIONS, VIEWS, AND SUBPROGRAMS

Gpjh_orders_2011q1-

This view returns the order item, quantity, price, and the order date for orders in the first quarter of 2011. It is sorted by order date in ascending order.

gpjh_snacks-

This view returns the item name, manufacturer, price, and the item stock from the item and category table where the category name is Snacks. It is ordered by item name in ascending order.

gpjh_proc_insertUser-

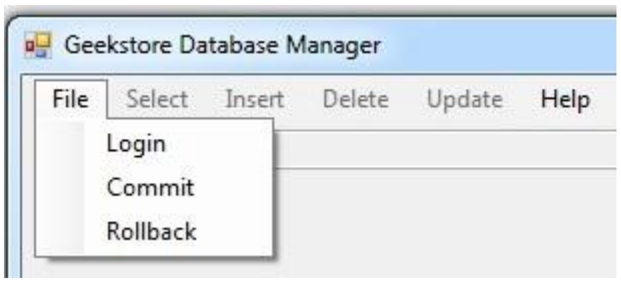
Is a stored procedure in which a new user may be added to the database. It takes parameters for all fields of the User table.

gpjh_ proc_deleteUser-

Is a stored procedure in which an existing user may be removed from the database starting with dependent records (have a foreign key referencing the user) cascading down.

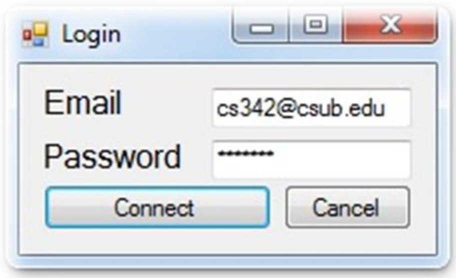
Due to the likelihood that these functions will be called often, we created these stored subprograms in order reduce downtime by having precompiled procedures. Similar deletion procedures were also created for all other tables.

5.3 SCREEN SHOTS OF PROGRAM



Login –

Selecting **Login** will pull up the window as shown below, allowing the user to enter credentials to log into the database.



Email: cs342@csub.edu
Password: ********

Commit-

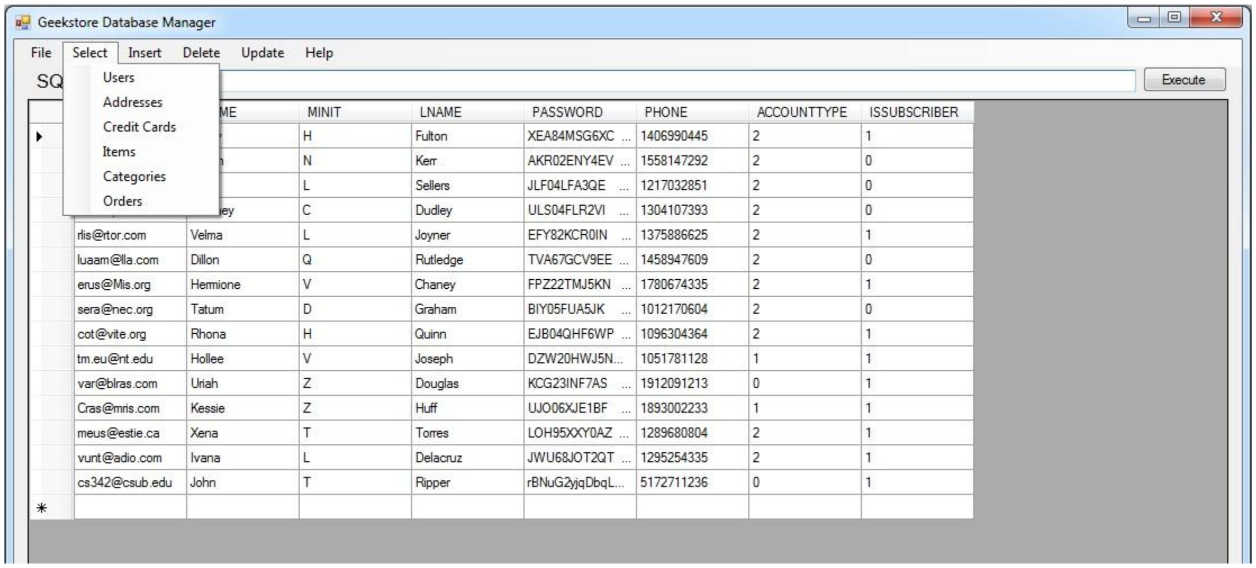
All changes to the database are unreversable after clicking **Commit**.

Rollback-

Click **Rollback** to revert all changes back to the last Commit.

Select-

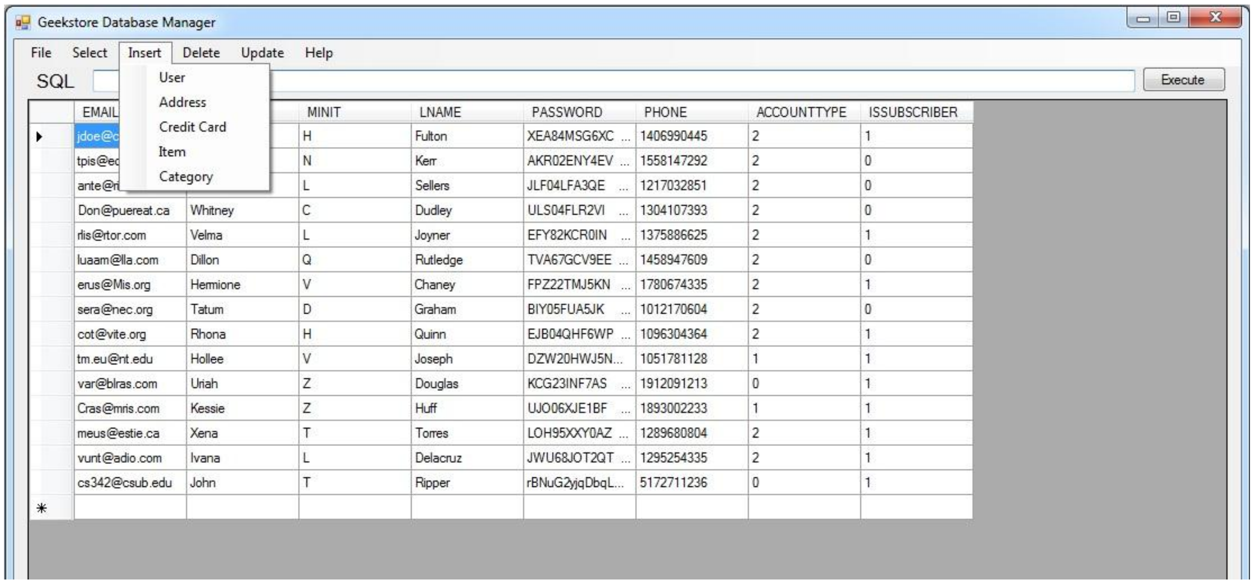
The **Select** option allows the user to view all user data, addresses, credit card data, items, categories, and orders. We have also implemented a SQL execution field in which an admin may type in custom SQL selection statements if they need to select more specific data.



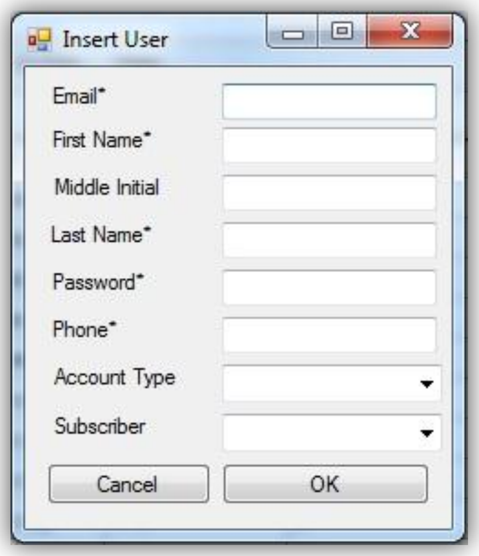
	ME	MINIT	LNAME	PASSWORD	PHONE	ACCOUNTTYPE	ISSUBSCRIBER
	y	H	Fulton	XEA84MSG6XC ...	1406990445	2	1
	n	N	Kerr	AKR02ENY4EV ...	1558147292	2	0
		L	Sellers	JLF04LFA3QE ...	1217032851	2	0
	ey	C	Dudley	ULS04FLR2VI ...	1304107393	2	0
rlis@rtor.com	Velma	L	Joyner	EFY82KCR0IN ...	1375886625	2	1
luaam@lla.com	Dillon	Q	Rutledge	TVA67GCV9EE ...	1458947609	2	0
erus@Mis.org	Hermione	V	Chaney	FPZ22TMJ5KN ...	1780674335	2	1
sera@nec.org	Tatum	D	Graham	BIY05FUA5JK ...	1012170604	2	0
cot@vite.org	Rhona	H	Quinn	EJB04QHF6WP ...	1096304364	2	1
tm.eu@nt.edu	Hollee	V	Joseph	DZW20HWJ5N...	1051781128	1	1
var@blras.com	Uriah	Z	Douglas	KCG23INF7AS ...	1912091213	0	1
Cras@mris.com	Kessie	Z	Huff	UJO06XJE1BF ...	1893002233	1	1
meus@estie.ca	Xena	T	Torres	LOH95XXY0AZ ...	1289680804	2	1
vunt@adio.com	Ivana	L	Delacruz	JWU68JOT2QT ...	1295254335	2	1
cs342@csub.edu	John	T	Ripper	rBNuG2yjqDbqL...	5172711236	0	1

Insert-

Selecting one of the choices from **Insert** will bring up a form in a new window that lets the user fill out all fields in the table. The user insert form is shown below. All fields with an asterik are required.



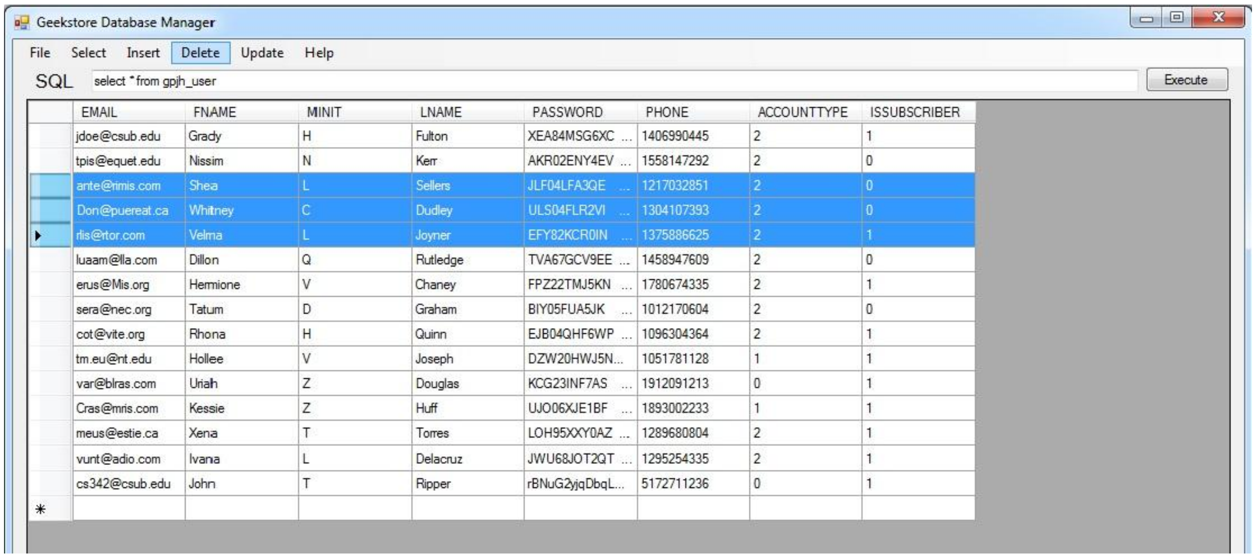
EMAIL		MINIT	LNAME	PASSWORD	PHONE	ACCOUNTTYPE	ISSUBSCRIBER
jdoe@c		H	Fulton	XEA84MSG6XC ...	1406990445	2	1
tpis@ec		N	Kerr	AKR02ENY4EV ...	1558147292	2	0
ante@n		L	Sellers	JLF04LFA3QE ...	1217032851	2	0
Don@puereat.ca	Whitney	C	Dudley	ULS04FLR2VI ...	1304107393	2	0
rlis@rtor.com	Velma	L	Joyner	EFY82KCR0IN ...	1375886625	2	1
luaam@lla.com	Dillon	Q	Rutledge	TVA67GCV9EE ...	1458947609	2	0
erus@Mis.org	Hemione	V	Chaney	FPZ22TMJ5KN ...	1780674335	2	1
sera@nec.org	Tatum	D	Graham	BIY05FUA5JK ...	1012170604	2	0
cot@vite.org	Rhona	H	Quinn	EJB04QHF6WP ...	1096304364	2	1
tm.eu@nt.edu	Hollee	V	Joseph	DZW20HWJ5N...	1051781128	1	1
var@blras.com	Uriah	Z	Douglas	KCG23INF7AS ...	1912091213	0	1
Cras@mris.com	Kessie	Z	Huff	UJO06XJE1BF ...	1893002233	1	1
meus@estie.ca	Xena	T	Torres	LOH95XXY0AZ ...	1289680804	2	1
vunt@adio.com	Ivana	L	Delacruz	JWU68JOT2QT ...	1295254335	2	1
cs342@csub.edu	John	T	Ripper	rBNuG2yjqDbqL...	5172711236	0	1



Email*	
First Name*	
Middle Initial	
Last Name*	
Password*	
Phone*	
Account Type	
Subscriber	

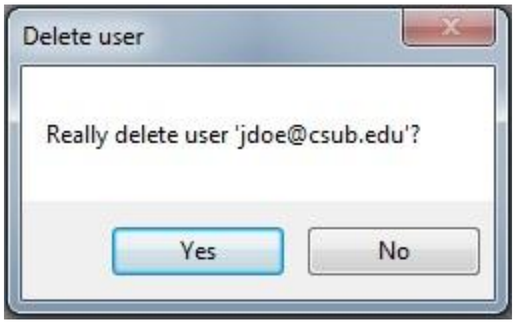
Delete-

Select a whole record by clicking on the small panel on the left of the table. Click it and drag to select multiple records. After the desired entrees are selected click on the delete button on the menu bar. A window will pop up and ask if you want to confirm that you want to delete the selected rows.



```
select * from gpjh_user
```

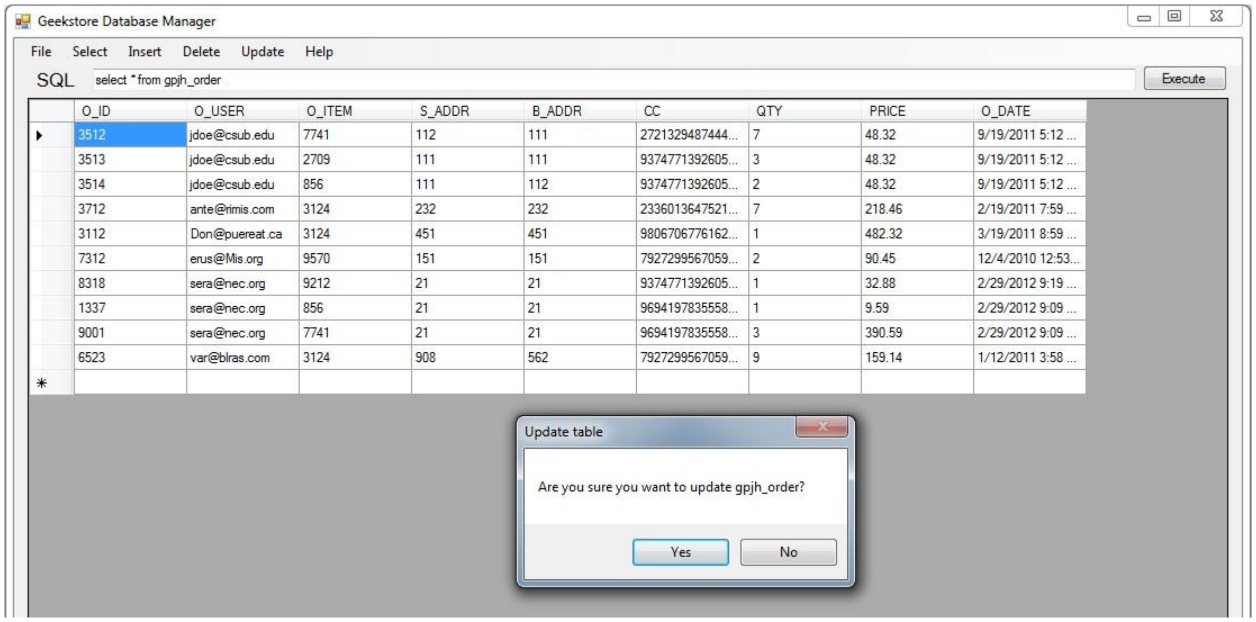
EMAIL	FNAME	MINIT	LNAME	PASSWORD	PHONE	ACCOUNTTYPE	ISSUBSCRIBER
jdoe@csub.edu	Grady	H	Fulton	XEA84MSG6XC ...	1406990445	2	1
tpis@equet.edu	Nissim	N	Kerr	AKR02ENY4EV ...	1558147292	2	0
ante@rimis.com	Shea	L	Sellers	JLF04LFA3QE ...	1217032851	2	0
Don@puereat.ca	Whitney	C	Dudley	ULS04FLR2VI ...	1304107393	2	0
rlis@rtor.com	Velma	L	Joyner	EFY82KCR0IN ...	1375886625	2	1
luaam@lla.com	Dillon	Q	Rutledge	TVA67GCV9EE ...	1458947609	2	0
erus@Mis.org	Hermione	V	Chaney	FPZ22TMJ5KN ...	1780674335	2	1
sera@nec.org	Tatum	D	Graham	BIY05FUA5JK ...	1012170604	2	0
cot@vite.org	Rhona	H	Quinn	EJB04QHF6WP ...	1096304364	2	1
tm.eu@nt.edu	Hollee	V	Joseph	DZW20HWJ5N...	1051781128	1	1
var@blras.com	Uriah	Z	Douglas	KCG23INF7AS ...	1912091213	0	1
Cras@mris.com	Kessie	Z	Huff	UJO06XJE1BF ...	1893002233	1	1
meus@estie.ca	Xena	T	Torres	LOH95XXY0AZ ...	1289680804	2	1
vunt@adio.com	Ivana	L	Delacruz	JWU68JOT2QT ...	1295254335	2	1
cs342@csub.edu	John	T	Ripper	rBNuG2yjqDbqL...	5172711236	0	1



Really delete user 'jdoe@csub.edu'?

Update-

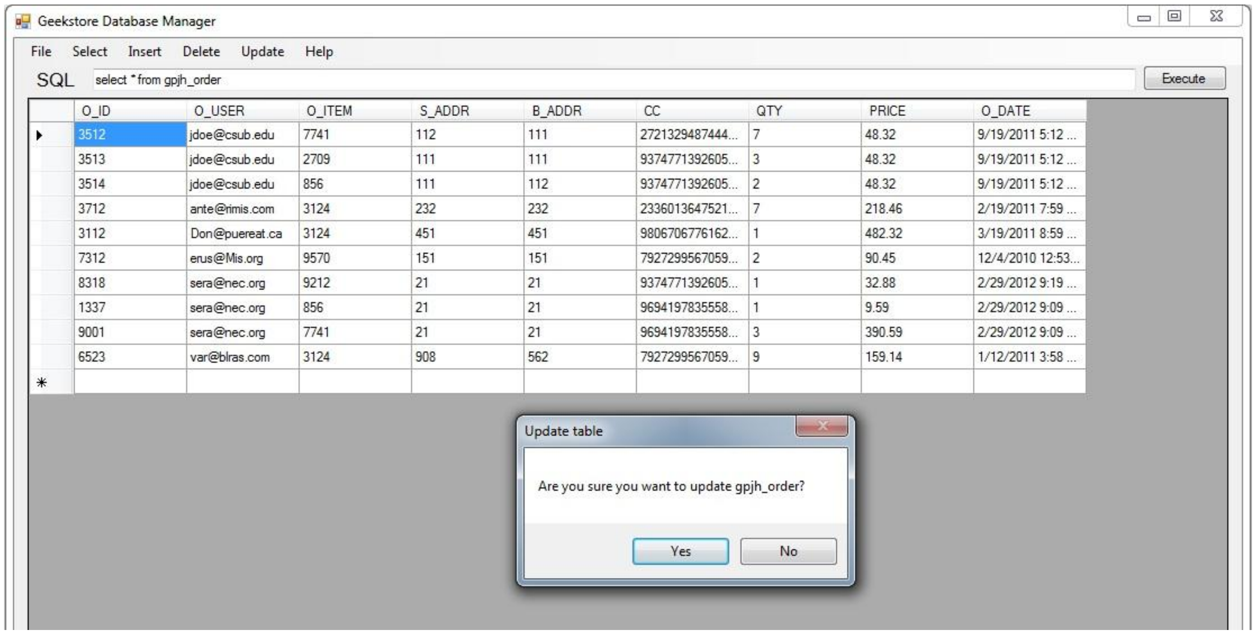
Selecting the update field from the menu bar takes the data entered into the fields and updates them into the database if they have been changed.


Geekstore Database Manager
File Select Insert Delete Update Help
SQL
Execute
Update table
Yes
No

select * from gpjh_order

O_ID	O_USER	O_ITEM	S_ADDR	B_ADDR	CC	QTY	PRICE	O_DATE
3512	jdoe@csub.edu	7741	112	111	2721329487444...	7	48.32	9/19/2011 5:12 ...
3513	jdoe@csub.edu	2709	111	111	9374771392605...	3	48.32	9/19/2011 5:12 ...
3514	jdoe@csub.edu	856	111	112	9374771392605...	2	48.32	9/19/2011 5:12 ...
3712	ante@rimis.com	3124	232	232	2336013647521...	7	218.46	2/19/2011 7:59 ...
3112	Don@puereat.ca	3124	451	451	9806706776162...	1	482.32	3/19/2011 8:59 ...
7312	erus@Mis.org	9570	151	151	7927299567059...	2	90.45	12/4/2010 12:53...
8318	sera@nec.org	9212	21	21	9374771392605...	1	32.88	2/29/2012 9:19 ...
1337	sera@nec.org	856	21	21	9694197835558...	1	9.59	2/29/2012 9:09 ...
9001	sera@nec.org	7741	21	21	9694197835558...	3	390.59	2/29/2012 9:09 ...
6523	var@blras.com	3124	908	562	7927299567059...	9	159.14	1/12/2011 3:58 ...

Are you sure you want to update gpjh_order?


Geekstore Database Manager
File Select Insert Delete Update Help
SQL
Execute
Update table
Yes
No

select * from gpjh_order

O_ID	O_USER	O_ITEM	S_ADDR	B_ADDR	CC	QTY	PRICE	O_DATE
3512	jdoe@csub.edu	7741	112	111	2721329487444...	7	48.32	9/19/2011 5:12 ...
3513	jdoe@csub.edu	2709	111	111	9374771392605...	3	48.32	9/19/2011 5:12 ...
3514	jdoe@csub.edu	856	111	112	9374771392605...	2	48.32	9/19/2011 5:12 ...
3712	ante@rimis.com	3124	232	232	2336013647521...	7	218.46	2/19/2011 7:59 ...
3112	Don@puereat.ca	3124	451	451	9806706776162...	1	482.32	3/19/2011 8:59 ...
7312	erus@Mis.org	9570	151	151	7927299567059...	2	90.45	12/4/2010 12:53...
8318	sera@nec.org	9212	21	21	9374771392605...	1	32.88	2/29/2012 9:19 ...
1337	sera@nec.org	856	21	21	9694197835558...	1	9.59	2/29/2012 9:09 ...
9001	sera@nec.org	7741	21	21	9694197835558...	3	390.59	2/29/2012 9:09 ...
6523	var@blras.com	3124	908	562	7927299567059...	9	159.14	1/12/2011 3:58 ...

Are you sure you want to update gpjh_order?

5.4 DESCRIPTION OF PROGRAM AND CODE

A. Major steps of designing a user interface

Designing a GUI application in Visual Studio and C# was a new experience for both of us. We used a menu bar to control the flow of the program and stored the data into a single data grid. All of the basic SQL operations – Select, Insert, Update, and Delete – are accessible from the menu bar. When designing a GUI, you must make sure that menu item placement is fairly intuitive. In addition, selections need to give proper error or confirmation messages. To improve our existing design, we could add tabbed data grid management in the future.

B. C# Database Classes

Oracle database classes can be used by C# in Visual Studio by adding a reference to Oracle.DataAccess.dll provided by the Oracle 11g client. These classes are specially designed to interact with Oracle databases. In addition to these classes, there are some essential ones built into the .NET Framework that are designed handle data from any database. I will cover the important classes in this section.

OracleConnection-

The OracleConnection class is used to manage connections with the Oracle database. To create a connection, you can give it a connection string in the constructor that specifies the data source, user id, and password. This is the connection string used in our database:

```
const string connString = "DATA SOURCE=delphi;USER ID=cs342;Password=c3m4p2s;";
```

OracleDataAdapter-

The OracleDataAdapter class is a special class that is often used for storing the results of a query on an Oracle database and filling the results into a generic dataset. The class can be instantiated with the connection and the SELECT statement you want to execute. Invoking the Fill method on the object with fill up the passed DataSet object with records from the result set.

OracleCommand-

While data adapters are used for SELECT statements, the OracleCommand class is designed for SQL commands that manipulate the data, such as INSERT, UPDATE, DELETE, and calling stored procedures. The OracleCommand object can also be used to parameterize commands, which has the advantage of being more secure and more efficient. Like OracleDataAdapter, an OracleCommand object can be instantiated with a SQL command and an OracleConnection object. Programmers also need to set the CommandType field correctly as well, such as when calling a stored procedure. To execute the command, call ExecuteNonQuery.

```
cmd = new OracleCommand(procedure, connectString);
cmd.CommandType = CommandType.StoredProcedure;
cmd.ExecuteNonQuery();
```

DataSet-

This class can be used for locally storing database queries into easy-to-access tables. It is not Oracle-specific – it could be used to store data from any database. The tables can then be used as a data source for a user-friendly DataGridView.

DataGridView-

DataGridView is used to create user-friendly visual data tables. Users can edit the cells of the table if read-only is not enabled. A DataGridView object can be filled with data by setting the data source to a DataTable:

```
adapter = new OracleDataAdapter(sql, connectString);
dataSet = new DataSet();
adapter.Fill(dataSet);
dataGrid.DataSource = dataSet.Tables[0];
```

C. Major Features of GUI Program

The deleteRow method can delete any record from the table based on the name of the stored procedure and the key that the record is identified by. A stored procedure that is set based on the currently selected table is called when this function is invoked. deleteRows invokes this function in a *for* loop when deleting multiple rows.

```
private bool deleteRow(string proc, object key)
        {
            try
            {
                openConnection();

                cmd = new OracleCommand(proc, cnn);
                cmd.CommandType = CommandType.StoredProcedure;

                if (key.GetType().ToString() == "System.Decimal")
                    cmd.Parameters.Add("arg1", OracleDbType.Int32, 0).Value =
Convert.ToDouble(key);
                else if (key.GetType().ToString() == "System.String")
                    cmd.Parameters.Add("arg1", OracleDbType.Varchar2, 0).Value =
Convert.ToString(key);

                int k = cmd.ExecuteNonQuery();
                cmd = null;
                return true;
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.ToString());
            }
            finally
            {
                closeConnection();
            }
            return false;
        }
```

The insertUser method is invoked from a form that has all the fields to create a new user. The fields are passed to the function, and then a stored procedure is called on the database server to execute the SQL command.

```
public bool insertUser(string email, string fname, string minit, string lname, string
pass, string phone, int acc, int sub)
        {
            try
            {
                openConnection();

                cmd = new OracleCommand("gpjh_proc_insertUser", cnn);
                cmd.Parameters.Add("email", OracleDbType.Varchar2, 0).Value = email;
                cmd.Parameters.Add("fname", OracleDbType.Varchar2, 0).Value = fname;
                cmd.Parameters.Add("minit", OracleDbType.Varchar2, 0).Value = minit;
                cmd.Parameters.Add("lname", OracleDbType.Varchar2, 0).Value = lname;
                cmd.Parameters.Add("pass", OracleDbType.Char, 0).Value = pass;
                cmd.Parameters.Add("phone", OracleDbType.Char, 0).Value = phone;
                cmd.Parameters.Add("acctype", OracleDbType.Int32, 0).Value = acc;
                cmd.Parameters.Add("subscribes", OracleDbType.Int32, 0).Value = sub;

                tableName = "gpjh_user";
                int k = cmd.ExecuteNonQuery();

                selectTable(tableName);
                cmd = null;
                return true;
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex);
            }
            finally
            {
                closeConnection();
            }
            return false;
        }
```

The updateTable method calls the UPDATE command for every attribute of the currently selected data in the DataGridView. It works for every table. In its current state, it takes a few seconds to execute for even a small set of a data. A more efficient version of this method would use a cached DataGridView and compare each value of the current DataGridView to the cached one before executing the SQL command. It would only execute an UPDATE command if two corresponding cells differ. Due to time constraints, we were not able to implement the more efficient method.

```
private void updateTable()
        {
            if (tableName == null || tableName == "")
                return;

             if (MessageBox.Show(
                    String.Format("Are you sure you want to update {0}?", tableName),
                    "Update table", MessageBoxButtons.YesNo) != DialogResult.Yes)
                return;

            object key;
            string keyName;

            keyName = dataGrid.Columns[0].Name;
            for (int i = 0; i < dataGrid.RowCount; i++)
            {
                key = dataGrid[0, i].Value;
                for (int j = 1; j < dataGrid.ColumnCount; j++)
                {
                    updateAttr(dataGrid.Columns[j].Name, keyName, key, dataGrid[j,
i].Value);
                }
            }
        }
```

D. Thoughts on Visual Studio / C#

After writing several projects in Java, the transition to C# was very easy. C# is so similar to Java that it is sometimes hard to tell the difference when coding in it. Visual Studio made the transition even easier, since GUI elements can be created with the click of a mouse. In addition, the IDE assists you by helping you complete class, variable, and method names.

5.5 Steps of Designing and Implementing a Database

An end-to-end database solution is no trivial matter. A project of such magnitude requires a lot of research, analysis, design, and refining.

The first step is to research the business, organization, or other purpose that the database is meant so serve. If it is an existing business/organization, work closely with people from within. It is a new idea, define your goals and all things that should be kept track of in the database. We have learned that it is very important to clearly define your goals from the start, because it is harder to change them later on.

Step two is to take your research and create an E-R (Entity-Relationship) model. This model is a basic model that will give you a general idea of how the database will be implemented. An E-R model can be easily explained to management and other non-technical people. We have learned that it is very important to make a well-designed E-R model because it will directly affect your relational model.

In the next step, one must convert the E-R model into a more practical form – the relational model. Modern DBMS's like Oracle are based on the relational model. In this process, entities are converted to relations, and foreign keys are used to refer to other relations. We've learned that relational models are very important because they can be directly applied to real databases. If your relations are not designed well, your database will require a lot of refinement.

In step 4, one should design queries in relational algebra and calculus that will be used in the database. However, a person who is an expert in the database design should be able to create SQL statements without writing relational algebra and calculus first. If the database designer is ever confused on how to write a SQL statement, they should first try to write it in relational algebra/calculus and convert it. Relational algebra and calculus helped us understand how to design real-world queries.

Step five is where stored subprograms are made, such as procedures, functions, and triggers. These are important because storing database subprograms on a server helps streamline the whole process. It abstracts the database design from the application programmer. In addition, it is more efficient and secure to call a stored subprogram that to write it directly in the interface software.

Lastly, the software interface should be written for the database. For businesses/organizations, this is usually in the form of a windowed GUI application or a website. However, sometimes it is more useful to write a console or CLI (command-line interface) application to interact with the database as well. It all depends on the purpose of the database and program.

In summation, every step of the database process is very important. Mistakes that are made early have a way of making it more difficult down the road. We had to go back and refine our ER model and relational model several times. We have learned the importance of defining business goals and sticking to them, and considering the consequences of every design decision. Though we made many mistakes, we went back and fixed most of them. I feel as though the best way to learn is the hard way, and this project has been a perfect example of that. After completing this project, we feel much better prepared to approach the task of designing and implementing a database.