# Giumarra & Associates Co.
# Shipping Warehouse

# ‘Database Project’

Aris Turner
Computer Science 342: Database Systems
Prof. H. Wang
9.25.2010

# Table of Contents

# Phase I: Information Gathering and E-R Modeling

# Fact Finding Techniques

The purpose of fact finding is to identify the necessary components required to create a streamlined and efficient database for my company. This process will identify daily processes, environment, potential users, and needs of the company before a conceptual model is created. By going through this process, potential errors, design flaws, and implementation problems can be eliminated before a database is constructed. The following fact finding techniques were used in preparation for constructing this database.

- **On-Site Inspection**. Being a former employee of GABCO (Giumarra & Associates Bottliing Co.), I was allowed to directly inspect daily routines of the warehouse. I was able to note processes and techniques the warehouse manager used in tracking incoming and outgoing product movement, along with potential errors that the database may alleviate.

- **Interviews**. I talked to the warehouse manager along with the billing supervisor to identify personal routines and needs the database should provide. Through this process I was able to identify who would be using the database along with the different need each user would need.

### Techniques Used

By simply following both potential users during the day I was given a first hand look into the needs of the database. Then by interviews I was able to narrow down and specify entities, environment needs, and specialized processes for each user.

### Introduction to Enterprise/Organization

Giumarra & Associates (GABCO) was created as a division of Giumarra Wineries in 2002. The company produces and bottles various energy drinks and teas for national drink companies such as Monster, Rockstar, and Arizona Tea. Due to a rapid expansion in sales and client needs, GABCO decided it was better to split the company into subdivisions consisting of production and warehousing.

## Structure of Enterprise

Due the the fact that the production division of the company was required to use preproduction databases created by its clients, I chose to focus on the warehousing division which had no formal database system other than simple cell sheets. The company was fully capable of running its various processes through an cell sheet, but due to the manual nature of an excel sheet and the difficulty of controlling user input, many errors arise because human error, i.e. accidental deletion of data, potential harmful access to edit data by unqualified or unauthorized users.

The requirements of the warehouse was simply in many ways. First, there is only one type of product style to control (pallets) meaning entity types and key identifiers would be easy to choose. Second, only two people are needed to run and control the database, meaning little need for numerous environments and a complicated access infrastructure. The warehouse manger needs to keep track of incoming product from the production facility along with shipments to various distribution warehouses for the clients. The manager needs a streamlined interface that will minimize time on a comp due to the speed of product arriving and leaving the warehouse. The billing supervisor need a much more simplified view of the database. He has no need for editing product data other than to mark shipping invoices appropriately throughout the billing process. Restricted access is both a more efficient process, but also eliminates potential number manipulation by unauthorized users.

## Itemized Description of Major Objects

The Pallet will be the major entity involved with the processes used by GABCO. This entity will identify each product in such a way as to meet both internal as well as legal (shipment and product problems) reasons. Connected to this is the Incoming/Outgoing Invoices entities which will be used to track the pallets entering and leaving the warehouse.

## Data Views and Operations for User Groups

The warehouse manager is capable of entering both incoming and outgoing invoices, filling them with Pallet information such as Product Name, Size, and Dates. The manager can create, edit, and delete invoices as necessary. The billing supervisor can view incoming invoices for scheduling purposes, he cannot edit incoming data in any way. The supervisor is allowed to view and add data pertaining to billing to outgoing invoices such as Date Billed, Date Paid. He is, however, not allowed to edit data made during the creation of the invoice. To edit data, he must consult the warehouse manager. Other warehouse employees have no access to the database.

# Conceptual Database Design

## Entity Set Description

### Pallet

- This entity type tracks the pallets that enter and leave the warehouse.
- Candidate Keys: palletID
- Primary Key: palletID
- Strong/Weak Entity: strong
- Fields to be indexed: palletID, prodName, prodSize, expDate
- Attributes:

| Name | palletID | createDate | expDate |
|---|---|---|---|
| **Description** | Pallet ID number | Creation Date of product | Expiration Date of product |
| **Domain/ Type** | Integer | Date | Date |
| **Range** | 0...2^31 | Any | Any |
| **Default** | none | none | none |
| **Null** | no | no | no |
| **Unique** | yes | no | no |
| **Single/ Multivalue** | single | single | single |
| **Simple/ Composite** | simple | composite | composite |

### Drink Type

- This entity tracks the various types of drinks contained in the pallets.
- Candidate Keys: prodName
- Primary Key: prodName
- Strong/Weak Entity: strong
- Fields to be indexed: prodName, prodSize, expDate
- Attributes:

| Name | prodName | prodSize |
|---|---|---|
| **Description** | Drink Name | Size of cans in pallet |
| **Domain/ Type** | String | Integer |
| **Range** | Any | 16, 24 |
| **Default** | none | none |
| **Null** | no | no |
| **Unique** | yes | no |
| **Single/ Multivalue** | single | single |
| **Simple/ Composite** | composite | simple |

### Incoming Invoice

- This entity type tracks incoming shipments from the production facility. This will provide crucial data for proper and timely shipping of product.
- Candidate Keys: inID
- Primary Key: inID
- Strong/Weak Entity: strong
- Fields to be indexed: inID, inDate
- Attributes:

| Name | inID | inDate | checkedinBy |
|---|---|---|---|
| **Description** | Invoice ID | invoice Date | person who checked shipment |
| **Domain/ Type** | Integer | Date | String |
| **Range** | 0...2^31 | Any | 30 char |
| **Default** | none | none | none |
| **Null** | no | no | no |

| Name | inID | inDate | checkedinBy |
|---|---|---|---|
| **Unique** | yes | no | no |
| **Single/ Multivalue** | single | single | single |
| **Simple/ Composite** | simple | composite | composite |

### Outgoing Invoice

- This entity tracks shipments from the warehouse. Along with dates shipped, it must also keep track of company shipped to.
- Candidate Keys: outID
- Primary Key: outID
- Strong/Weak Entity: strong
- Fields to be indexed: outID, outDate, shippedTo
- Attributes:

| Name | outID | outDate | shippedTo | checkedoutBy | billDate | paidDate |
|---|---|---|---|---|---|---|
| **Description** | Invoice ID | invoice Date | company shipped to | person who checked shipment | billing Date | date shipment is paid for |
| **Domain/ Type** | Integer | Date | String | String | Date | Date |
| **Range** | 0...2^32 | Any | 40 char | 30 char | Any | Any |
| **Default** | none | none | none | none | none | none |
| **Null** | no | no | no | no | yes | yes |
| **Unique** | yes | no | no | no | no | no |
| **Single/ Multivalue** | single | single | single | single | single | single |
| **Simple/ Composite** | simple | composite | single | composite | composite | composite |

### Location

- This entity tracks the location of pallets for accurate and timely distibution
- Candidate Keys: rowNum
- Primary Key: rowNum
- Strong/Weak Entity: Strong
- Fields to be indexed: none
- Attributes:

| Name | rNum |
|---|---|
| Descriptio n | row number |
| Domain/ Type | Integer |
| Range | 0...450 |
| Default | none |
| Null | no |
| Unique | yes |
| Single/ Multivalue | single |
| Simple/ Composite | simple |

### User

- This entity tracks employees who have access to database
- Candidate Keys: userName, Dept
- Primary Key: userName
- Strong/Weak Entity: Strong
- Fields to be indexed: Name
- Attributes:

| Name | userName | deptName |
|---|---|---|
| Descriptio n | employee name | employee Dept |

| Name | userName | deptName |
|---|---|---|
| **Domain/ Type** | String | String |
| **Range** | 40 char | 25 char |
| **Default** | none | none |
| **Null** | no | no |
| **Unique** | no | yes |
| **Single/ Multivalue** | single | single |
| **Simple/ Composite** | composite | simple |

## Relationship Set Description

### Shipped In

Each incoming shipment must contain product to be recorded. This is the relationship between the pallets shipped to the warehouse and the invoice used to track the shipment. The relationship has a 'status' attribute to keep track of pallets that may have been damaged and are no longer in the warehouse.

Mapping Cardinality: 1...M

Descriptive Field: none

Participation Constraint: mandatory for all incoming shipments to the warehouse

### Shipped Out

Each outgoing shipment must contain product to be recorded. This is the relationship between the pallets shipped from the warehouse and the invoice used to track the shipment. The relationship has a 'status' attribute to keep track of pallets that may have been damaged and are no longer in the warehouse.

Mapping Cardinality: 1...M

Descriptive Field: none

Participation Constraint: mandatory for all outgoing shipment from the warehouse

### Checks In/Out

Each shipment must be verified (checked) by an employee. This is the relationship between the invoices and the users checking them.

Mapping Cardinality: 1...1

Descriptive Field: none

Participation Constraint: mandatory for all invoices both in and out of the warehouse

### LocatedIN

Tracks the location of each pallet in the warehouse

Mapping Cardinality: M...1

Descriptive Field: none
Participation Constraint: Mandatory for each pallet in warehouse

### Contains

Tracks the type of drink contained in the pallets

Mapping Cardinality: 1...M

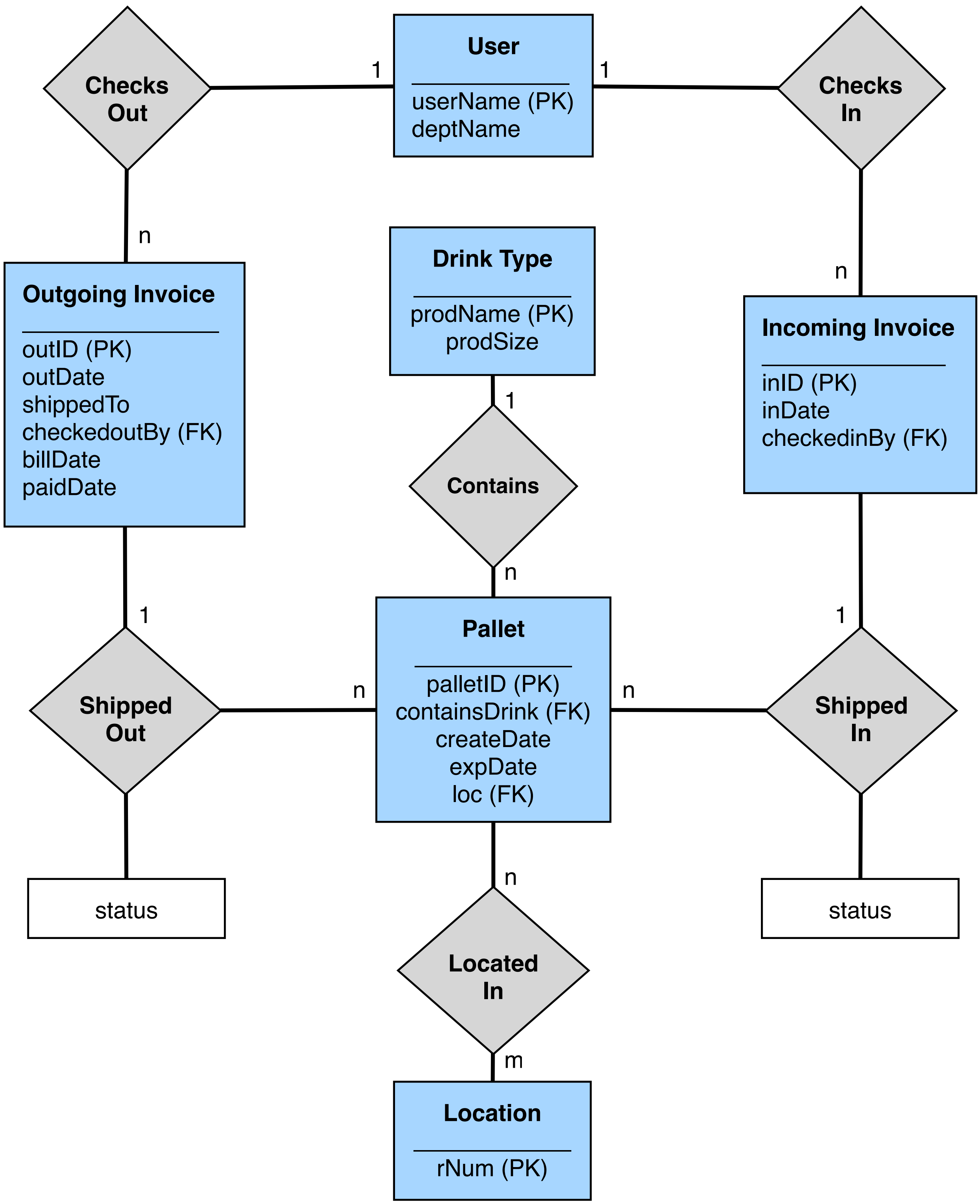
Descriptive Field: none

Participation Constraint: Mandatory for each palled in warehouse

## Related Entity Set

Due to the simplicity of the shipments involved in the warehouse, there is no need for this. The warehouse manager and billing supervisor are fully capable of maintaining these processes. This may change given a need to expand staff as business grows.

# E–R Diagram



Checks Out
1
User
userName (PK)
deptName
1
Checks In
n
Drink Type
prodName (PK)
prodSize
n
Outgoing Invoice
outID (PK)
outDate
shippedTo
checkedoutBy (FK)
billDate
paidDate
1
Contains
Incoming Invoice
inID (PK)
inDate
checkedinBy (FK)
n
1
1
Pallet
palletID (PK)
containsDrink (FK)
createDate
expDate
loc (FK)
Shipped Out
n
n
Shipped In
status
n
status
Located In
m
Location
rNum (PK)

# Phase II: Relational Model

## ER-Model vs. Relational Model

### Description

The Entity-relationship model shown in Phase I is a valuable tool for visualizing the data's organization for the planned database. However, this conceptual form of the database must be converted into a relational form before it can be implemented as an actual, functioning database. The relational model was first described by IMB Researcher Ted Codd in his 1970 paper, "A Relational Model of Data for Large Shared Data Banks." This model allows for all data to be described as a set of relations with constraints set on given domains. Under these conditions, we can produce a valid, finite description of the theoretical database, which allows for faster, easier conversion into the actual database.

### Comparison

The entity-relationship model represents a visual description of the proposed database format, with implied attributes, relations, and cardinality. This serves as the conceptual design for the database. This design omits any sort of implementation details, so no focus is taken away from the design aspects needed for this phase. Because of its visual layout, the entity-relationship model is best used with non-technical potential users of the planned database.

The relational model lays out the previously-described entities and relationships between entities as a table with its attributes as columns. Each row in the table is a valid record, or tuple, in the database, and each table is referred to as a relation. This representation allows the database designers to better understand the size and domain constraints for the final product. The relational model lacks the visual advantages of the entity-relationship model, but allows for better, more accurate descriptions of domain constraints and tuple entries. It also more closely resembles the actual structure of the tables in the implemented database.

### Conversion from E-R model to relational model

Creation of a relational model is facilitated by first producing a conceptual model. A conceptual provides basic structural integrity, from which the relational model's relations and column attributes can be mapped. An algorithm exists to expedite this process. This algorithm takes into account the existence of an extended entity-relationship model (one with specialization, generalization, and union types represented), but the structure of this project's E-R model does not call for these extra steps.

First, a relation is created for each strong entity type in the E-R model. Each relation has the same simple attributes as it did in the E-R model. An attribute is selected as the primary key for the relation, using a combination of attributes if a composite key is used. Secondary keys may also be allowed. Next, weak entities and their attributes are mapped to relations. Their primary key can be denoted as a combination of any partial keys the weak entity has with the primary key of its owner(s). Next, the two participating entities in all 1:1 binary relationship types R from the E-R model are mapped to separate relations S and T. The representation in the relational model can be created using several methods:

- Include the primary key of one relation as a foreign key in the other. Best for one total participation.
- Merge the two entity types into a single relation. Best for total participation in both entities.
- Create another relation to hold the primary keys of each relation. This is necessary for M:N relationships, but can be implemented for any cardinality.

These steps are considered and used for mapping 1:M and M:N relationships to the relational model. The next step is to create a relation to represent multi-valued attributes. This relation will have an attribute for each portion of the E-R model's multi-valued attribute. This relation can be assigned a primary key that can be referenced from any other relation that uses the multi-valued attribute. Finally, these steps are combined to create relations to represent N-ary relationship types from the E-R model.

For conceptual models that involve specialization or generalization, additional steps are taken for proper representation in the relational model. There are several approaches for this step:

- Create one relation for the superclass with {k, a1, a2, …, an} attributes, and create m relations for each subclass with its own attributes unioned with the superclass' attributes, and specify k as the subclass' primary key. This works for any arrangement of specialization: total, partial, disjoint, or overlapping
- Create a relation for each subclass, and union the subclass' attributes with the superclass' attributes. This works for when every superclass object belongs to at least one subclass.
- Create a single relation with all subclass and superclass attributes unioned. This can create many NULL entries in the resulting tuples.

Following these steps will convert the conceptual model into a valid, complete, relational model.

## Constraints

A relation consists of an ordered set of unique tuples, with each tuple having the same amount and type of attributes. Entities are represented as relations in the relational model, and each row is a valid instance, record, or tuple for the entity. Entity constraints ensure that for a relation, no primary key can be NULL. This ensures that there is a unique element of each tuple in the relation, which is necessary for comparisons and representations in queries and data integrity. Constraints for foreign keys exist to enforce referential integrity: any references to other existing tuples in other relations must be valid. A foreign key constraint states that for a given attribute value in a relation r1, that attribute acts as a primary key for another relation r2, and the value exists for some tuple in r2. Thus, any references made in one tuple actually exist within the database. Check constraints and business rules exist to serve a specific business need for the data. These constraints add further limitations to entries in the database: the data must not only be of the right type, but must meet certain requirements, such as falling within a given range, or making sure a telephone number is from a specific area code.

# ER database to relational database

## Pallet Relation

**Attributes:**

- palletID
  - Domain: integer from 1 to (2^32 -1). Not NULL
- createDate
  - Domain: date. Not NULL
- expDate
  - Domain: date. Not NULL
- containsDrink
  - Domain: varchar2(25). Value corresponds to the Drink Name in the Drink Type entity. Each pallet must contain a drink name so the relationship is represented as an attribute in the Pallet relation.
- loc
  - Domain: integer. Value corresponds to rowNum in the Location entity. Not NULL.

**Constraints:**

- Primary Key: palletID. Must be unique and not NULL.
- Foreign Key: containsDrink. Must have a value that exist in Drink Type relation.
- Foreign Key: loc. Must have value that exist in Drink Type relation.

## Drink Type Relation

**Attributes:**

- prodName
    - Domain: string. Not NULL
- prodSize
    - Domain: integer. 16, 24. Not NULL

**Constraints:**

- Primary Key: prodName. Must be unique. Not NULL.

## Incoming Invoice Relation

**Attributes:**

- inID
    - Domain: integer from 1 to (2^32 -1). Not NULL.
- inDate
    - Domain: Date. Not NULL.
- checkedinBy
    - Domain: integer. Corresponds to user ID in User entity.

**Constraints:**

- Primary Key: inID. Must be unique and not NULL.
- Foreign Key: checkedinBy. Related to userID in User entity. Must contain values in the userName attribute.

## Outgoing Invoice Relation

**Attributes:**

- outID
    - Domain: unsigned integer from 1 to (2^32 -1). Not NULL.
- outDate
    - Domain: date. not NULL.
- shippedTo
    - Domain: string. Limit 40 char. Not NULL.
- billDate:
    - Domain: date not NULL.
- paidDate:
    - Domain: date. not NULL.
- checkedoutBy
    - Domain: integer. Corresponds to userID in User entity.

**Constraints:**

- Primary Key: outID. Must be unique and not NULL.

- Foreign Key: checkedoutBy. Related to userID in User entity. Must contain values in the userID attribute.

## Location Relation

**Attributes:**

- rNum:
    - Domain: integer. not NULL.

**Constraints:**

- Primary Key: rowNum. Must be unique and not NULL.

## User Relation

**Attributes:**

- userName:
    - Domain: varchar2(45). Limit 40 char. not NULL.
- deptName:
    - Domain: varchar2(15). Limit 30 char. not NULL
- userID:
    - Domain: integer. unique. not NULL

**Constraints:**

- Primary Key: userID. Must be not NULL.

## Shipped In Relation

**Attributes:**

- palletID
    - Domain: integer from 1 to (2^31 -1). Not NULL.
- inID
    - Domain: integer from 1 to (2^31 -1). Not NULL.
- status
    - Domain: string. Limit 200 char. NULL.

**Constraints:**

- Foreign Keys: palletID and inID must both exist from respective relations.

## Shipped Out Relation

**Attributes:**

- palletID
  - Domain: integer from 1 to (2^31 -1). Not NULL.
- outID
  - Domain: integer from 1 to (2^31 -1). Not NULL.
- status
  - Domain: string. Limit 200 char. NULL.

**Constraints:**

- Foreign Keys: palletID and outID must both exist in respective relations.

## Checks In Relation

**Attributes:**

- inID
  - Domain: integer from 1 to (2^31 -1). Not NULL.
- userID:
  - Domain: integer. unique. not NULL.

**Constraints:**

- Foreign keys: inID and userID must both exist in respective relations.

## Checks Out Relation

**Attributes:**

- outID
  - Domain: integer from 1 to (2^31 -1). Not NULL.
- userID:
  - Domain: integer. unique. not NULL.

**Constraints:**

- Foreign Keys: outID and userID must both exist in respective relations.

## Located In Relation

### Attributes:

- palletID
  - Domain: integer from 1 to (2^31 -1). Not NULL.
- rNum:
  - Domain: integer. not NULL.

### Constraints:

- Foreign Keys: palletID and rowNum must both exist in respective relations.

## Contains Relation

### Attributes:

- palletID
  - Domain: integer from 1 to (2^31 -1). Not NULL
- prodName
  - Domain: string. Not NULL

### Constraints:

- Foreign Keys: palletID and prodName must both exist in respective relations.

# Relational Instances

## Pallet(palletID, createDate, expDate, containsDrink, loc)

| palletID | createDate | expDate | containsDrink | loc |
|---|---|---|---|---|
| 1 | 10/10/2010 | 10/10/2011 | MonKhaos16 | 1 |
| 2 | 10/10/2010 | 10/10/2011 | MonKhaos16 | 1 |
| 3 | 10/12/2010 | 10/12/2011 | RSMango24 | 2 |
| 4 | 10/13/2010 | 10/13/2011 | RSMango24 | 3 |
| 5 | 10/13/2010 | 10/13/2011 | RSGuava16 | 4 |
| 6 | 10/13/2010 | 10/13/2011 | MonReg24 | 5 |
| 7 | 10/13/2010 | 10/13/2011 | MonReg16 | 6 |
| 8 | 10/14/2010 | 10/14/2011 | MonAss16 | 7 |
| 9 | 10/15/2010 | 10/15/2011 | MonAss16 | 7 |
| 10 | 10/16/2010 | 10/16/2011 | RSGuava24 | 8 |

## Drink_Type(prodName, prodSize)

| prodName | prodSize |
|---|---|
| MonKhaos16 | 16 |
| MonKhaos24 | 24 |
| RSMango16 | 16 |
| RSMango24 | 24 |
| RSGuava16 | 16 |
| RSGuava24 | 24 |
| MonReg16 | 16 |
| MonReg24 | 24 |
| MonAss16 | 16 |
| MonAss24 | 24 |

### Incoming_Invoice(inID, inDate, checkedinBy)

| inID | inDate | checkedinBy |
|---|---|---|
| 12345 | 10/10/2010 | 1 |
| 12346 | 10/10/2010 | 1 |
| 12347 | 10/12/2010 | 3 |
| 12358 | 10/13/2010 | 1 |
| 12567 | 10/13/2010 | 3 |
| 12568 | 10/13/2010 | 2 |
| 12600 | 10/13/2010 | 2 |
| 12601 | 10/14/2010 | 3 |
| 12603 | 10/15/2010 | 3 |
| 12605 | 10/16/2010 | 1 |

### Outgoing_Invoice(outID, outDate, checkedoutBy)

| outID | outDate | shippedTo | billDate | paidDate | checkedinBy |
|---|---|---|---|---|---|
| 652 | 12/14/2010 | AMD | 12/15/2010 | 01/04/2011 | 2 |
| 672 | 12/17/2010 | AMD | 12/19/2010 | 01/04/2011 | 1 |
| 690 | 12/17/2010 | TA Dist | 12/19/2010 | | 3 |
| 700 | 12/17/2010 | TA Dist | 12/19/2010 | | 1 |
| 701 | 12/19/2010 | GA Wholesale | 12/23/2010 | | 3 |
| 703 | 12/20/2010 | Master Storage | 12/26/2010 | | 2 |
| 704 | 12/21/2010 | Master Storage | | | 2 |
| 752 | 12/21/2010 | GA Wholesale | | | 3 |

| outID | outDate | shippedTo | billDate | paidDate | checkedinBy |
| --- | --- | --- | --- | --- | --- |
| 800 | 12/23/2010 | TA Dist | | | 3 |
| 801 | 12/29/2010 | TA Dist | | | 1 |

## Location(rNum)

| rNum |
| --- |
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
| 6 |
| 7 |
| 8 |
| 9 |
| 10 |

## User(userName, userDept)

| userID | userName | userDept |
| --- | --- | --- |
| 1 | Aris Turner | Inventory |
| 2 | M. Crazy | Inventory |
| 3 | J. Doe | Inventory |
| 4 | A. Man | Finance |
| 5 | I. Johg | Finance |
| 6 | O. Snapp | Finance |

| userID | userName | userDept |
|---|---|---|
| 7 | R. Brown | Inventory |
| 8 | M. Jackson | Inventory |
| 9 | S. Rice | Finance |
| 10 | H. Fukufugi | Finance |

### Shipped_In(palletID, inID, status)

| palletID | inID | status |
|---|---|---|
| 1 | 12345 | NULL |
| 2 | 12345 | NULL |
| 3 | 12345 | NULL |
| 4 | 12358 | NULL |
| 5 | 12567 | Bad wrap. Rewrapped. |
| 6 | 12567 | broken cans. Destroyed |
| 7 | 12600 | NULL |
| 8 | 12600 | NULL |
| 9 | 12600 | NULL |
| 10 | 12605 | wrong date on packaging. Sent back to production |

### Shipped_Out(palletID, outID, status)

| palletID | outID | status |
|---|---|---|
| 3 | 652 | NULL |
| 4 | 652 | NULL |
| 7 | 652 | NULL |
| 8 | 700 | NULL |
| 10 | 700 | NULL |

| palletID | outID | status |
|---|---|---|
| 1 | 703 | NULL |
| 2 | 704 | NULL |
| 5 | 752 | torn wrap. rewrapped |
| 9 | 800 | NULL |
| 6 | 801 | NULL |

### Checks_In(inID, userName)

| inID | userID |
|---|---|
| 12345 | 1 |
| 12346 | 1 |
| 12347 | 3 |
| 12358 | 1 |
| 12567 | 3 |
| 12568 | 2 |
| 12600 | 2 |
| 12601 | 3 |
| 12603 | 3 |
| 12605 | 1 |

### Checked_Out(outID, userName)

| outID | userName |
|---|---|
| 652 | 2 |
| 672 | 1 |
| 690 | 3 |
| 700 | 1 |

| outID | userName |
|---|---|
| 701 | 3 |
| 703 | 2 |
| 704 | 2 |
| 752 | 3 |
| 800 | 3 |
| 801 | 1 |

## Located_In(palletID, rowNum)

| palletID | rowNum |
|---|---|
| 1 | 1 |
| 2 | 1 |
| 3 | 2 |
| 4 | 3 |
| 5 | 4 |
| 6 | 5 |
| 7 | 6 |
| 8 | 7 |
| 9 | 7 |
| 10 | 8 |

## Contains(palletID, prodName)

| palletID | prodName |
|---|---|
| 1 | MonKhaos16 |
| 2 | MonKhaos16 |
| 3 | RSMango24 |

| palletID | prodName |
|---|---|
| 4 | RSMango24 |
| 5 | RSGuava16 |
| 6 | MonReg24 |
| 7 | MonReg16 |
| 8 | MonAss16 |
| 9 | MonAss16 |
| 10 | RSGuava24 |

# Queries

1. Select all Outgoing Invoices checked by a ‘A. Turner’
2. Select all locations of ‘Mon Khaos 16’
3. Select all Incoming invoices created on ‘10/10/2010’
4. Select all pallets shipped out on ‘10/18/2010’
5. Select all pallets with expiration before ‘12/30/2010’
6. Select all invoices with ‘RS Mango 24’ in them
7. Select user name who checked Incoming Invoice ‘12345’
8. Select users who checked incoming invoices on ‘10/16/2010’

## 1. Select all invoices checked by a ‘A. Turner’

**Relational Algebra**

π (outID) (σ(checkedinBy = ‘A. Turner’)(Incoming_Invoice))

**Tuple Calculus**

{o | Incoming_Invoice(o) AND o.checkedinBy = ‘A. Turner’}

**Domain Relational Calculus**

{a | Incoming_Invoice(a, b, ‘A. Turner’) }

## 2. Select all locations of ‘Mon Khaos 16’

**Relational Algebra**

Khaos <– π (palletID) (σ(prodName = ‘Mon Khaos 16’)(Drink_Type))

π(rowNum) (σ(palletID = Khaos)(Located_In))

## 3. Select all Incoming invoices created on ’10/10/2010’

**Relational Algebra**

π(inID)(σ(inDate = ’10/10/2010’)(Incoming_Invoice)

**Tuple Calculus**

{i | Incoming_Invoice(i) AND i.inDate = 10/10/2010 }

**Domain Relational Calculus**

{a | Incoming_Invoice(a, 10/10/2010, c) }

## 4. Select all pallets shipped out on ’12/17/2010’

**Relational Algebra**

π(palletID) σ(Shipped_Out.outid = σ(shipDate = ’12/17/2010’)
(Outgoing_Invoice)

## 5. Select all pallets with expiration date before ’10/14/2011’

**Relational Algebra**

π(palletID) (σ(expDate < 10/14/2011)(Pallet)

**Tuple Calculus**

{ p | Pallet(p) AND p.expDate < 10/14/2011 }

**Domain Relational Calculus**

{a | (∃a)(∃b)(∃c) (Pallet(abc) AND b < 10/14/2011) }

## 6. Select all invoices with ‘RS Mango 24’ in them

## 7. Select the user name who checked Incoming Invoices ‘12345’

**Relational Algebra**

π(userName)

**Tuple Calculus**

{ u | User(u) AND (∃o)(Outgoing_Invoice(o)) AND o.outID=12345 AND u.userName=o.checkedoutBy }

**Domain Relational Calculus**

{ a | (∃m)(∃o)(∃a) (User(ab) AND Outgoing_Invoice(mnopqr) AND m=12345 AND a=o) }

## 8. Select all users who checked incoming invoices on '10/13/2010'

**Relational Algebra**

Invoice <- π (inID) (σ(inDate = 10/16/2010)(Incoming_Invoice))

Result <- π(userName)(σ(Invoice)(Checks_In))

**Relational Calculus**

{u | Checks_In(u) AND (∃i)(Incoming_Invoice(i)) AND i.inDate = 10/16/2010 AND i.inID = u.inID }

# Phase III: Implementation of the relational database

## SQL*PLUS

Now that the relational model has been completed, the description for each relation can be used to actually create a database that meets its requirements regarding attributes, constraints, and relationships. To do this, I will use the implementation of SQL from the Oracle Relational Database Management System (hereafter referred to as Oracle). Structured Query Language, or SQL, was first developed at IBM in the 1970s. Since then, it has undergone rigorous optimization and standardization, and several popular implementations are used for most databases. These include Microsoft's Transact-SQL, or T-SQL, MySQL, and Oracle. Oracle provides a tool called SQL*PLUS that allows users to interactively run any SQL commands. It's a command-line tool that supports both user interaction and automated scripts. By using a hierarchy of scripts to call appropriate commands, a database can be destroyed and re-created very quickly using SQL*PLUS.

## Schema Objects in Oracle

In oracle, a collection of schema objects forms a schema. A tablespace logically organizes the structure of the database with respect to various schemas. It also contains the locations used to physically store data on the database's media. Schema objects are logical data structures that are stored logically in a given tablespace within the database. The physical data for each schema object is stored in the tablespace's data files. This structure allows any tablespace to logically contain many different types of schema objects, but remain optimized for storage and access. Oracle supports several schema objects. They are as follows:

### Tables

Tables are used to represent relations from the relational model. They serve as the basic storage unit for an Oracle database. A table's columns represent the relation's attributes, and rows in the table represent existing records or tuples in the relation. Each column has a unique name and data type. Tables store information about the relation's primary key, any foreign keys, and any other constraints it may possess (including referential or null). After the table is created, rows can be inserted to represent the existence of tuples.

### Views

Views are essentially read-only query commands that will always return tuples from tables that meet a certain requirement. These are used when the same 34commands will be used repeatedly. This allows for clearer organization, and database optimization. The results of a view can be thought of as tables and, as such, can be accessed and modified like a table. Views lose referential or integrity constraints, but these can be implied by the underlying tables that the view accesses in its execution. Views do not use storage space like a table - only the commands that represent the view are saved. Its results are not, since they are implied from other tables. Views can be used to obfuscate data, prevent direct access for certain users, and simplify representation for users. Materialized views are special views that perform a specific function on the data it retrieves, including aggregate functions, sorting, summations, data transfer, and reorganization.

### Dimensions

Dimensions are used to create hierarchical relationships between columns in a table. This can be used between columns of the same table (denormalized) or of separate tables (normalized).

### Sequences

Sequence generators create a sequential set of numbers for use in a multi-user environment. These sequence numbers can then be used to determine order for queued operations or requests. They are not dependent on any table, but they can be used to generate primary keys for a specific table. Sequence numbers can also be used to keep track of roll-backs in transactions, ensuring that the right commands are reversed without confliction between separate users.

### Synonyms

Synonyms are alternate aliases for certain types of schema objects, such as tables, procedures, functions, or views. They do not require any additional storage space other than their entries in the database's data dictionary. Synonyms can be used to directly hide internal data for outside users or to simplify complex SQL commands.

### Indexes

Databases attempt to optimize traversal of each table by caching the values of unique attributes, such as primary keys. Additional attributes can be specified such that the database more quickly accesses their values during comparisons for overall faster results. Indexes can also be created for combinations of certain attributes. Furthermore, an existing index can be used to create another dependent index. An Oracle system will automatically maintain indexes once specified by a user.

### Database Links

Put simply, database links are hard-coded, read-only links to another database. This allows one database to perform queries and retrieve results using another database, while simultaneously preventing both databases from risking the integrity of one another.

### Stored procedures and functions

These can be seen as scripts that are stored on the database. When executed, a stored procedure or function always performs the same task as instructed upon
35
its creation. Functions in Oracle always return a single value to the user, while stored procedures do not.

### Packages

Packages are a specific collection of stored procedures, functions, and cursors. Combined, they act as a single unit of instructions. This is critical for large- scale operations performed by stored procedures. Packages organize and simplify design requirements for databases that require persistent, complex tasks.

### Schema objects in this project

In this project, the two most frequently used schema objects are the table
Most of the tables are created using syntax similar to this:
CREATE Table [TableName] (
attributes attribute types nullable?

Constraints:
pk_tablename PRIMARY KEY (AttributeName)
fk_ParentName_ChildName FOREIGN KEY (AttributeName) REFERENCES ParentName (ParentAttributeName)
);

The scheme objects created using this syntax are as follows:

- at_checksin
- at_checksout
- at_contains
- at_drinktype
- at_ininvoice
- at_locatedin
- at_location
- at_outinvoice
- at_pallet
- at_shippedin
- at_shippedout
- at_user

### at_checksin

```
CS342 SQL> desc at_checksin;
 Name                 Null?            Type
 ---------------      -------------   --------------
 INID                 NOT NULL        NUMBER(38)
 USERID               NOT NULL        NUMBER(38)

CS342 SQL> select * from at_checksin;

      INID     USERID
---------- ----------
     12345          1
     12346          1
     12347          3
     12358          1
     12567          3
     12568          2
     12600          2
     12601          3
     12603          3
     12605          1

10 rows selected.

CS342 SQL> spool off
```

### at_checksout

```
CS342 SQL> desc at_checksout;
 Name                            Null?     Type
 ------------------------        --------  -------------------
 OUTID                           NOT NULL  NUMBER(38)
 USERID                          NOT NULL  NUMBER(38)

CS342 SQL> select * from at_checksout;

     OUTID     USERID
---------- ----------
       652          2
       672          1
       690          3
       700          1
       701          3
       703          2
       704          2
       752          3
       800          3
       801          1
```

```
10 rows selected.

CS342 SQL> spool off
```

**at_contains**

```
CS342 SQL> desc at_contains;
 Name                          Null?     Type
 ------------------------      --------  ------------------
 PALLETID                      NOT NULL  NUMBER(38)
 PRODNAME                      NOT NULL  VARCHAR2(25)

CS342 SQL> select * from at_contains;

  PALLETID     PRODNAME
----------     -------------------------
         1     MonKhaos16
         2     MonKhaos16
         3     RSMango24
         4     RSMango24
         5     RSGuava16
         6     MonReg24
         7     MonReg16
         8     MonAss16
         9     MonAss16
        10     RSGuava24

10 rows selected.

CS342 SQL> spool off
```

**at_drinktype**

```
CS342 SQL> desc at_drinktype;
 Name                Null?    Type
 ------------------ -------- -----------------------
 PRODNAME            NOT NULL VARCHAR2(25)
 PRODSIZE            NOT NULL NUMBER(4)

CS342 SQL> select * from at_drinktype;

PRODNAME                        PRODSIZE
------------------------- ----------
MonKhaos16                            16
```

```
MonKhaos24                              24
RSMango16                               16
RSMango24                               24
RSGuava16                               16
RSGuava24                               24
MonReg16                                16
MonReg24                                24
MonAss16                                16
MonAss24                                24

10 rows selected.

CS342 SQL> spool off
```

**at_ininvoice**

```
CS342 SQL> desc at_ininvoice;
 Name                      Null?    Type
 ------------------------- -------- ----------------------
 INID                      NOT NULL NUMBER(38)
 INDATE                    NOT NULL DATE
 CHECKEDINBY               NOT NULL NUMBER(38)

CS342 SQL> select * from at_ininvoice;

      INID INDATE    CHECKEDINBY
---------- --------- -----------
     12345 10-OCT-10           1
     12346 10-OCT-10           1
     12347 12-OCT-10           3
     12358 13-OCT-10           1
     12567 13-OCT-10           3
     12568 13-OCT-10           2
     12600 13-OCT-10           2
     12601 14-OCT-10           3
     12603 15-OCT-10           3
     12605 15-OCT-10           1

10 rows selected.

CS342 SQL> spool off
```

**at_locatedin**

```
CS342 SQL> desc at_locatedin;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------
 PALLETID                                  NOT NULL NUMBER(38)
 RNUM                                      NOT NULL NUMBER(5)

CS342 SQL> select * from at_locatedin;

  PALLETID       RNUM
---------- ----------
         1          1
         2          1
         3          2
         4          3
         5          4
         6          5
         7          6
         8          7
         9          7
        10          8

10 rows selected.

CS342 SQL> spool off
```

**at_location**

```
CS342 SQL> desc at_location;
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------------------
 RNUM                                      NOT NULL NUMBER(5)

CS342 SQL> select * from at_location;

      RNUM
----------
         1
         2
         3
         4
         5
         6
         7
         8
```

```
         9
        10

10 rows selected.

CS342 SQL> spool off
```

**at_outinvoice**

```
CS342 SQL> desc at_outinvoice;
 Name                     Null?    Type
 ------------------------ -------- ---------------------
 OUTID                    NOT NULL NUMBER(38)
 OUTDATE                  NOT NULL DATE
 SHIPPEDTO                NOT NULL VARCHAR2(40)
 BILLDATE                          DATE
 PAIDDATE                          DATE
 CHECKEDOUTBY             NOT NULL NUMBER(38)

CS342 SQL> select * from at_outinvoice;

     OUTID OUTDATE   SHIPPEDTO BILLDATE  PAIDDATE  CHECKEDOUTBY
---------- --------- --------- --------- --------- ------------
       652 14-DEC-10 AMD       15-DEC-10 04-JAN-11            2
       672 17-DEC-10 AMD       19-DEC-10 04-JAN-11            1
       690 17-DEC-10 TA Dist   19-DEC-10                      3
       700 17-DEC-10 TA Dist   19-DEC-10                      1
       701 19-DEC-10 GA Wholesale 23-DEC-10                   3
       703 20-DEC-10 Master Storage 26-DEC-10                 2
       704 21-DEC-10 Master Storage                           2
       752 21-DEC-10 GA Wholesale                             3
       800 23-DEC-10 TA Dist                                  3
       801 29-DEC-10 TA Dist                                  1

10 rows selected.

CS342 SQL> spool off
```

**at_pallet**

```
CS342 SQL> desc at_pallet;
 Name                      Null?    Type
 ------------------------- -------- --------------------------
 PALLETID                  NOT NULL NUMBER(38)
```

```
 CREATEDATE                       NOT NULL DATE
 EXPDATE                          NOT NULL DATE
 CONTAINSDRINK                    NOT NULL VARCHAR2(25)
 LOC                              NOT NULL NUMBER(5)

CS342 SQL> select * from at_pallet;

  PALLETID CREATEDAT EXPDATE   CONTAINSDRINK             LOC
---------- --------- --------- ------------------ ----------
         1 10-OCT-10 10-OCT-11 MonKhaos16                  1
         2 10-OCT-10 10-OCT-11 MonKhaos16                  1
         3 12-OCT-10 12-OCT-11 RSMango24                   2
         4 13-OCT-10 13-OCT-11 RSMango24                   3
         5 13-OCT-10 13-OCT-11 RSGuava16                   4
         6 13-OCT-10 13-OCT-11 MonReg24                    5
         7 13-OCT-10 13-OCT-11 MonReg16                    6
         8 14-OCT-10 14-OCT-11 MonAss16                    7
         9 15-OCT-10 15-OCT-11 MonAss16                    7
        10 16-OCT-10 16-OCT-11 RSGuava24                   8

10 rows selected.

CS342 SQL> spool off
```

**at_shippedin**

```
CS342 SQL> desc at_shippedin;
 Name                              Null?    Type
 ----------------------------- -------- --------------------
 PALLETID                          NOT NULL NUMBER(38)
 INID                              NOT NULL NUMBER(38)
 STATUS                                     VARCHAR2(200)

CS342 SQL> select * from at_shippedin;

  PALLETID       INID STATUS
---------- ---------- ----------------------------------------
         1      12345
         2      12345
         3      12345
         4      12358
         5      12567 Bad wrap. Rewrapped.
         6      12567
         7      12600
         8      12600
```

```
         9      12600 Wrong date on pallet. Retagged.
        10      12605

10 rows selected.

CS342 SQL> spool off
```

**at_shippedout**

```
CS342 SQL> desc at_shippedout;
 Name                           Null?    Type
 ------------------------------ -------- --------------------
 PALLETID                       NOT NULL NUMBER(38)
 OUTID                          NOT NULL NUMBER(38)
 STATUS                                  VARCHAR2(200)

CS342 SQL> select * from at_shippedout;

  PALLETID      OUTID STATUS
---------- ---------- ----------------------------------------
         3        652
         4        652
         7        652
         8        700
        10        700
         1        703
         2        704
         5        752 Torn wrap. Rewrapped.
         9        800
         6        801

10 rows selected.

CS342 SQL> spool off
```

**at_user**

```
CS342 SQL> desc at_user;
 Name                             Null?    Type
 -------------------------------- -------- -------------------------
 USERID                           NOT NULL NUMBER(38)
 USERNAME                         NOT NULL VARCHAR2(45)
 DEPTNAME                         NOT NULL VARCHAR2(30)
```

```
CS342 SQL> select * from at_user;

    USERID USERNAME                    DEPTNAME
---------- --------------------------- -----------------
         1 Aris Turner                 Inventory
         2 Mike Crazy                  Inventory
         3 John Doe                    Inventory
         4 Andrew Man                  Finance
         5 Isaiah Johg                 Finance
         6 Omar Snapp                  Finance
         7 Randy Brown                 Inventory
         8 Michael Jackson             Inventory
         9 Sidney Rice                 Finance
        10 Hideki Fukufugi             Finance

10 rows selected.

CS342 SQL> spool off
```

# Queries

**1. Select all invoices checked by 'A.Turner'**

```
column userName format a15;
select inID, userName
from at_checksin natural join at_user
where at_user.userName = 'Aris Turner'
/
```

```
CS342 SQL> @q1

      INID USERNAME
---------- ---------------
     12345 Aris Turner
     12346 Aris Turner
     12358 Aris Turner
     12605 Aris Turner
```

**2. Select all locations of 'MonKhaos16'**

```
select rNum, prodName
from at_locatedin natural join at_contains
where at_contains.prodName = 'MonKhaos16'
/
```

```
CS342 SQL> @q2

      RNUM PRODNAME
---------- -------------------------
         1 MonKhaos16
         1 MonKhaos16
```

**3. Select all incoming invoices create on ’10/10/2010’**

```
select *
from at_ininvoice
where inDate = to_date('10/10/2010', 'mm/dd/yyyy')
/
```

```
CS342 SQL> @q3

      INID INDATE    CHECKEDINBY
---------- --------- -----------
     12345 10-OCT-10           1
     12346 10-OCT-10           1
```

**4. Select all pallets shipped on ’12/17/2010’**

```
select palletId, outDate
from at_shippedout natural join at_outinvoice
where at_outinvoice.outDate = to_date('12/17/2010', 'mm/dd/yyyy')
/
```

```
CS342 SQL> @q4

  PALLETID OUTDATE
---------- ---------
         4 17-DEC-10
         7 17-DEC-10
         8 17-DEC-10
```

**5. Select all pallets with expiration date before ’10/14/2011’**

```
select *
from at_pallet
where expDate < to_date('10/14/2011', 'mm/dd/yyyy')
```

```
/

CS342 SQL> @q5

  PALLETID CREATEDAT EXPDATE   CONTAINSDRINK
LOC
---------- --------- --------- -------------------------
----------
         1 10-OCT-10 10-OCT-11 MonKhaos16
1
         2 10-OCT-10 10-OCT-11 MonKhaos16
1
         3 12-OCT-10 12-OCT-11 RSMango24
2
         4 13-OCT-10 13-OCT-11 RSMango24
3
         5 13-OCT-10 13-OCT-11 RSGuava16
4
         6 13-OCT-10 13-OCT-11 MonReg24
5
         7 13-OCT-10 13-OCT-11 MonReg16
6

7 rows selected.
```

**6. Select all outgoing invoices with 'RSMango24' in them**

```
column status format a21;
column shippedto format a20;
column prodName format a15;
select outID, outDate, palletID, prodName
from at_outinvoice natural join (at_shippedout natural join at_contains)
where prodName = 'RSMango24'
/
```

```
CS342 SQL> @q6

     OUTID OUTDATE     PALLETID PRODNAME
---------- --------- ---------- ---------------
       652 14-DEC-10          3 RSMango24
       672 17-DEC-10          4 RSMango24
```

**7. Select the User who checked incoming invoice '12345'**

```
column userName format a25;
select inID, userID, userName
from at_ininvoice natural join at_user
```

```
where at_ininvoice.checkedinBy = at_user.userID AND
inID = 12345
/
```

```
CS342 SQL> @q7

      INID     USERID USERNAME
---------- ---------- -------------------------
     12345          1 Aris Turner
```

**8. Select all invoices who checked incoming invoices on '10/13/2010'**

```
column userName format a25;
select inID, inDate, userId, userName
from at_ininvoice natural join at_user
where at_ininvoice.checkedinBy = at_user.userId
and inDate = to_date('10/13/2010', 'mm/dd/yyyy')
/
```

```
CS342 SQL> @q8

      INID INDATE        USERID USERNAME
---------- --------- ---------- -------------------------
     12358 13-OCT-10          1 Aris Turner
     12600 13-OCT-10          2 Mike Crazy
     12568 13-OCT-10          2 Mike Crazy
     12567 13-OCT-10          3 John Doe
```

**9. Select the date with the most number of pallets shipped. (Not Finished)**

```
select outDate, count(palletID) palletnum
from at_outinvoice natural join at_shippedout
group by outDate
/
```

```
CS342 SQL> @q9

OUTDATE    PALLETNUM
--------- ----------
29-DEC-10          1
19-DEC-10          1
14-DEC-10          1
17-DEC-10          3
20-DEC-10          1
23-DEC-10          1
21-DEC-10          2
```

```
7 rows selected.

CS342 SQL> spool off
```

# Phase IV: Stored Procedures

## Common Features in Oracle PL/SQL and Microsoft Transact-SQL

Oracle and Microsoft's implementations of SQL are not completely independent. Based off of a common language, both Procedural Language/Structured Query Language and Transaction-SQL share many common features, despite being developed separately by Oracle and Microsoft, respectively. Both languages support commands to create tables, constraints, functions, cursors, stored procedures, triggers, and packages. Their biggest differences are in the syntax used to create and maintain these objects in the database. Furthermore, both languages have supported functions to translate and compare variables, look up dates and times, and manage user-defined variables.

Differences between the two forms of SQL often stem from the version being used. For example, Oracle 8i does not have very much support for nested SELECT statements in cursors, but allows this in later versions. It is difficult to perform mass updates on records in early T-SQL, but Oracle provides this functionality. In a way, the necessities of database users and designers have pushed both languages to converge to provide similar functionality so that users who choose one implementation do not miss the benefits of the other.

Depending on the structure and usage of a database, it might be advantageous to define tasks that can be repeatedly and quickly run by specific users. Stored subprograms, or stored procedures, are supported in both PL/SQL and T-SQL for this purpose. Subprograms can be written to automate otherwise tedious processes, such as inserting, deleting, or updating records in the database. Furthermore, the database designer can obfuscate important, confidential information from its users by storing these tasks in a subprogram. Since the user can only invoke the subprogram, not view or edit it, any sensitive information is protected. Furthermore, having a stored subprogram saves the programmer from designing an application that has to repeatedly send dynamically-generated SQL strings to the database. This means the programmer does not have to worry about SQL injection exploits, SQL string sanitization, or any other possible caveats that appear when using dynamic SQL to communicate with front-end database management systems.

## Oracle PL/SQL

Most PL/SQL programs follow a similar structure regardless of their purpose. Code statements are organized into blocks. There are three main sections of a block:

- Declaration: Declaration of variables, cursors, and user-defined exceptions are made here.
- Execution: This portion consists of the SQL statements that perform the task's job.

- Exception: This section catches any exceptions, either system or user-defined, raised during execution of the task.

**Layout:**

```
DECLARE
	variable_name			variable_type		:= value | DEFAULT
BEGIN
	SELECT | INSERT | UPDATE | DELETE
END;
```

**Variable types:**

All variable types supported by the Oracle server should be supported in PL/SQL. This includes numbers, floating points, character arrays, dates, unique IDs, and more.

**Cursors:**

Cursors are user-defined SQL statements that allow structured traversal of a table using a loop structure. They are defined using the following syntax:

```
DECLARE
	CURSOR cursor_name [parameters]
	IS select_statement;
```

After creation, a cursor can be used in the following format:

```
BEGIN
	FOR t in cursor_name LOOP
		Perform tasks
	END LOOP;
END;
```

**Control statements**

Control statements manage the logical flow of a PL/SQL subprogram. Since PL/SQL is a procedural language, the location and usage of these statements is extremely important. The following are example control statements:

```
IF condition THEN statement;
ELSEIF condition THEN statement;
END IF;
```

```
LOOP
	EXIT WHEN can be used to quit this loop
END LOOP;
```

```
FOR I IN lowerbound .. upperbound LOOP
	statement
END LOOP;
```

```
FOR cursor_variable IN cursor_name LOOP
	statement

END LOOP;
```

**Exception Handling**

PL/SQL allows users to catch and raise exceptions under certain conditions. The syntax to raise and handle exceptions is simple:

```
DECLARE
	User_defined_exception EXCEPTION;

BEGIN
	IF condition THEN RAISE User_defined_exception;
	END IF;

EXCEPTION
	WHEN Exception_name THEN statement;

END;
```

**Stored procedures**

Stored procedures can greatly facilitate performing complex jobs on the SQL server's data while maintaining abstraction for non-technical users. The structure of a stored procedure depends largely on the type of work it will be performing, but the syntax is the same for all of them:

```
CREATE [OR REPLACE] PROCEDURE procedure_name
[ (variablename		IN|OUT		variabletype)]
AS
(DECLARE variables go here)
BEGIN
	SQL statements
END;
```

Execution of a stored procedure from SQL*PLUS can be accomplished as follows:

```
SQL> exec sp_name(arguments go here);
```

**Stored functions**

Stored functions are created and run in a method very similar to the syntax for stored procedures. However, they differ by explicitly declaring a variable type to return after completion. These can be used to guarantee that a variable will be returned. The syntax is as follows:

```
CREATE [OR REPLACE] FUNCTION function_name
[ (variablename		IN|OUT		variabletype)]
RETURN datatype;
AS
```

```
(DECLARE variables go here)
BEGIN
        SQL statements;
        RETURN variable;
END;
```

**Packages**

Packages are a distinct collection of procedures and functions. Creating a package requires a prototype for each included procedure and function:

```
CREATE PACKAGE package_name AS
        PROCEDURE names..;
        FUNCTION names…;

END package_name;

CREATE PACKAGE BODY package_name AS
        PROCEDURE name IS…
        BEGIN
                Statements
        END;

        FUNCTION name RETURN DATATYPE IS…
        BEGIN

                Statements
                RETURN variable
        END;

END package_name;
```

**Triggers**

Triggers make collecting records, logs, and audits extremely easy. Instead of managing this task and forcing users to use pre-defined stored procedures and packages that manually execute tasks, triggers are executed when a certain condition is met, depending on the action (UPDATE, DELETE, INSERT, etc). Since these tasks are automated after the trigger's creation, the user does not have to worry about maintaining or checking data before or after the operations.

```
CREATE [OR REPLACE] TRIGGER trigger_name
BEFORE|AFTER                INSERT|DELETE|UPDATE OF COL [column_name] [OR
DELETE|UPDATE|INSERT]
ON table_name
DECLARE
          variables
BEGIN
        FOR EACH ROW
        [WHEN CONDITION]
```

```
        Statements;
END;
```

## Oracle PL/SQL Subprograms

### Stored Procedures

### deleteuser

This procedure deletes a user by taking in the desired user ID as an identifier

```
CREATE OR REPLACE PROCEDURE deleteUser( userpk IN number)
        AS
        BEGIN
                delete from at_user
                where userID = userpk;

        END deleteUser;
/
```

### insertuser

This procedure inserts a new user into the user table. I takes as arguments the user ID, user name, and the department the user works for.

```
CREATE OR REPLACE PROCEDURE insertUser(
        userID IN number,
        name IN varchar2,
        dept IN varchar2
        )
        AS
        BEGIN
                insert into at_user values(
                        userID,
                        name,
                        dept);
        END insertUser;
/
```

### NAvgpallets

This procedure find out the average number of pallets shipped each day by the following syntax

```
CREATE OR REPLACE FUNCTION NAvgpallets (n IN NUMBER) RETURN NUMBER
IS
	s number(9,2) := 0.0;
	p number(7,2);
	CURSOR c IS
		SELECT count(palletID)
		FROM at_outinvoice natural join at_shippedout
		GROUP BY outDate;
BEGIN
	open c;
	FOR i IN 1..n LOOP
		fetch c into p;
		s := s + p;
	END LOOP;
	close c;
	RETURN s/n;

END;
/
```

**at_userafterupdate**

This trigger creates a log of users info whenever the user's dept is changed

```
CREATE OR REPLACE TRIGGER at_userafterupdate
after update of deptName on at_user
for each row
begin
	insert into at_logtable
	values(userlogsequence.nextval,
sysdate, :old.userID, :old.userName, :old.deptName, :new.deptName);

END;
/
```

**at_deleteuserupdate**

This trigger creates a log of users info whenever they are deleted from the user table

```
CREATE OR REPLACE TRIGGER at_userdeleteupdate
BEFORE DELETE
	ON at_user
	FOR EACH ROW
BEGIN

	insert into at_deletetable
	values(userdeletesequence.nextval, sysdate,
:old.userID, :old.userName, :old.deptName);
```

```
END;
/
```

# Phase V: GUI Design and Implementation

## Daily Activities

There are several user groups for this database.

### Warehouse Managers

Warehouse managers will be the people entering invoices for incoming and outgoing shipping, along with creating data for new pallets as shipments come in. They have a lot of access to the database as they are the key data inputters for the company. They will be allowed to create pallets for tracking, add invoices for shipments, and also delete pallets due to damage.

### Finance Officers

Finance officers will take the outgoing shipments and use the reports to bill clients for the pallets. The will not have and edit access to the database other than marking outgoing invoices with bill dates and pay dates.

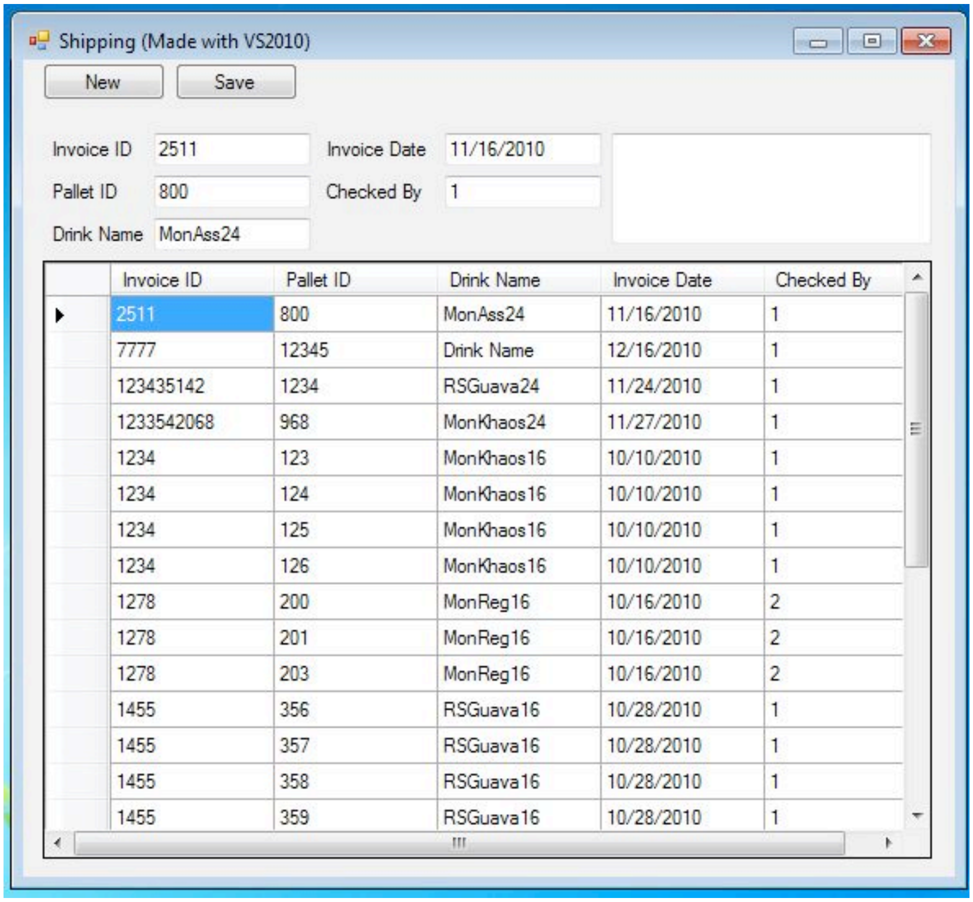
### Company Managers

Lastly, a company manager will have no editing ability on the database. They will be only allowed to view reports on the warehouse activity.

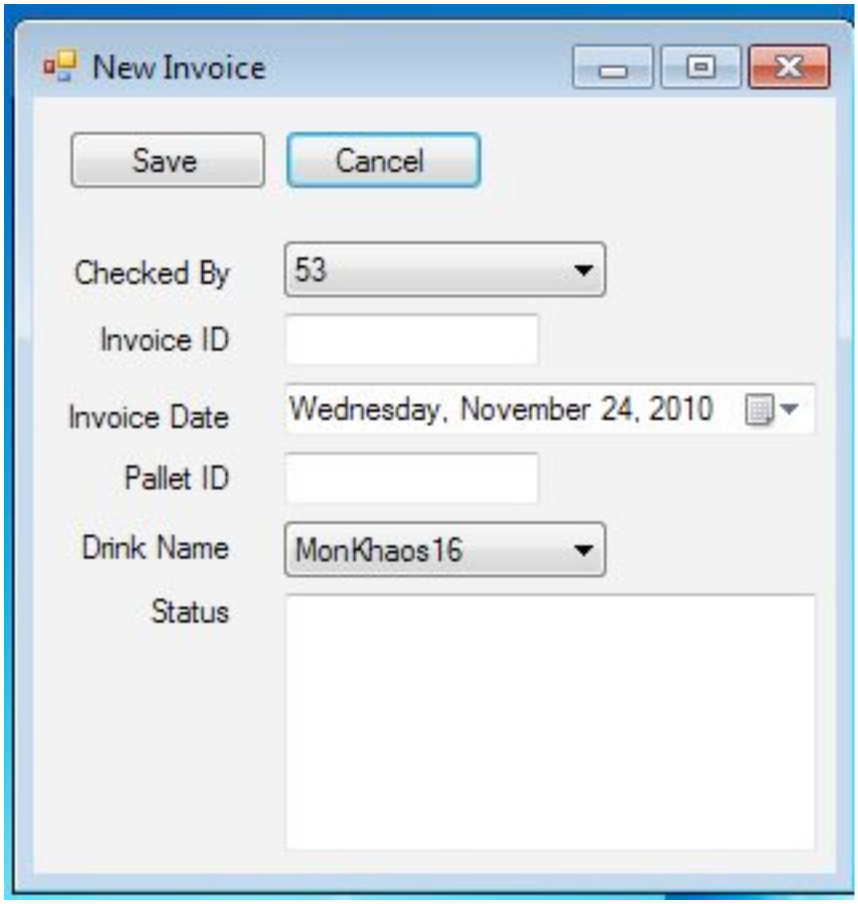
## Relations, Views, Subprograms

For each of the groups to have proper access to the database. Every relation in the database will be used. As I was unable to build a fully functional application I have not created the necessary views or subprograms that might be needed to make the application runs smoothly.

## Application Screenshots




The application I was able to create was very simple in both it's layout and it's functionality. The main screen (shown above) listed all incoming invoices in the warehouse. It utilized the at_shipping table to list the Invoice ID, pallets in the invoice, the drink name, invoice date, and who checked the shipment in. The main screen also contains a [new] button, which launched the 'new invoice' screen, and a [save] button which takes all changes made to the tables and pushes them onto the database.




The New Invoice screen is where a warehouse manager would enter the incoming invoices for input into the database. This is not a full design, it should have include a 'create pallet' command and allow for multiple pallets to be added all at once. Also the 'checked by' option would ideally be handled by a authentication login which would then launch the appropriate views. The 'drink name' option is bound by the at_drinktype table which contains all the drinks manufactured by the production facility.

## Code Description and GUI Design

I had originally had high ambitions for the style of the application, but due to my extreme difficulties in getting my computer to connect to the database, I was limited as to how much I could do. I was hoping to have a more detailed account into which the user would put in the new invoices. I would have liked the program to take user input out of the scenario so as to minimize mistakes.

Once I was able to get my computer connected, I used the very powerful and convenient tools for creating a dataset in visual studio. This allowed me to focus on building a GUI that would be more functional without having to spend a lot of time making sure my connections were correct. What I have learned through the process of building my simple application is that connecting to a database and manipulating the data can be very simple once the proper knowledge of the coding is acquired. Where the success of an application is built is in the laying out of tasks and functions that a user might need in order to get things done quickly and efficiently. Had I spent more time on this last phase I should have begun by describing the application in a practical sense before any coding had begun.

## Conclusion

This project was my first real experience at a full application and all that is entailed in creating even a simple one. This experience has given my great skills, maybe not particularly coding in c# or building a database, but as to the steps to attack a project and make leaps towards building my own skill set for the real world. I plan on spending the next few months continuing the program so that i know that my knowledge is set in my head for future use and so i can build in the foundations I have learned in the class.