# The Allosaurus'

Chris Leitch, Clayton Fielding,
Jared Hardinger, and Jared Lyman

Texas International Terminals (TIT) is a seaport in Galveston Texas specializing in the loading and offloading of granular product. TIT is expanding and as such needs a more sophisticated method of keeping track of product.

Starting in January 2011, The Allosaurus' designed a database and front end for TIT. This database is designed to record, monitor, and make reports for inbound and outbound shipments. This project was for an advanced database management class at Utah State University in the Jon M. Huntsman School of Business and was finished April 2011.

MIS 4330
Dr. David Olsen
April 28, 2011

## Contents

## Overview and Scope of Project

At the beginning of the semester, we were faced with a fairly daunting challenge. We were organized into realistically-assigned groups with the challenge to create a database that would actually help a real-life shareholder. Fortunately, one of our group members presented an excellent idea early on. Clayton suggested that we make a database for the company that his father currently works for: Texas International Terminals. Clayton explained to the group that TIT was in dire need of an efficient database; they were currently using Microsoft Excel to store and organize all of their important data! We decided as a group that this would be the perfect shareholder to build a database for.

Clayton briefly described what Texas International Terminals did as a company, but we needed a little more information regarding the business processes and relationships. To get this vital information, we connected with Matt Haidinyak, a Texas International Terminals (T.I.T) employee who is in charge of recording much of the shipment and inventory data. Over a conference call, Matt explained the intricate details of how T.I.T operated. Over the course of a few phone conversations and many questions from us, we finally got a pretty good hold on what T.I.T is about. To summarize, T.I.T  is basically a third party logistics company; they unload large amounts of cargo from large ships, store the cargo in their warehouse, and eventually load them on trucks, railcars, and/or river barges to be shipped to other companies. Matt was very helpful throughout the entire process. He was very willing to take the time to help us and clearly answer our questions.

Once we had decided we learned T.I.T‘s business rules and processes, we began creating the design for the database. We started by determining which tables and relationships were needed. We took those tables and the relationships between them and made an ER diagram.

After spending three to four hours on the design alone, we started to confuse ourselves and we decided to ask Dr. Olsen for a little help. He gave us some simple advice and we got rid of a few tables and relationships. In about 10-15 minutes we got the help that we really needed and it saved us a lot of time and effort. In hindsight, we learned that it is important to keep your design simple and not to overanalyze relationships and tables. We also learned how helpful it can be to ask for some guidance and advice from someone with experience.

Once we had finalized our ER diagram and database design, we determined that we needed to start actually creating the tables and relationships in SQL Server. First we created a user account for our group, which we called Texas Shipping. We all had the password to the user account and we were ready to go, so we divided up responsibilities. Each person in the group was given a few tables to create in our actual database. Once the tables were created, we decided which stored procedures and triggers would be needed to create an effective database and end-user program that would be of use to Matt at Texas International Terminals.

We created about six stored procedures when all was said and done. Each stored procedure would be used in the GUI that we would create. Many of the stored procedures we made were used to insert rows into tables. For example, the stored procedure sp_InsertCargo allows the user to input data into a new cargo tuple. Upon executing the stored procedure, the user enters the data for each column in the tuple. Continuing the example, a user would enter a unique CargoID and a CargoName.

The triggers we used were also very important to the database. Many of our triggers fired when a row was inserted or updated. One trigger calculates an important measurement (ShrinkAmount) needed by Texas International Terminals when the row was updated or inserted. This trigger takes the value of one column (ShortTonsShipped) in the inserted or updated row, multiplies it by another user-entered value (ShrinkPercentage) and then inserts that value in another column (ShrinkAmount). Previously this trigger was done manually by Matt in Excel, so this trigger and all of our other triggers make the process of calculations a lot more efficient.

Throughout the process of creating tables, relationships, stored procedures, and triggers, we ran in to many problems and bugs in our SQL. We did a lot of research (using Google) and group problem solving to figure out the solutions. We also hit some road blocks when we redesigned a few of the tables and added some necessary rows. These added rows threw off both our stored procedures and our triggers, so we had to go in to each stored procedure and trigger and debug and add code where it was needed.

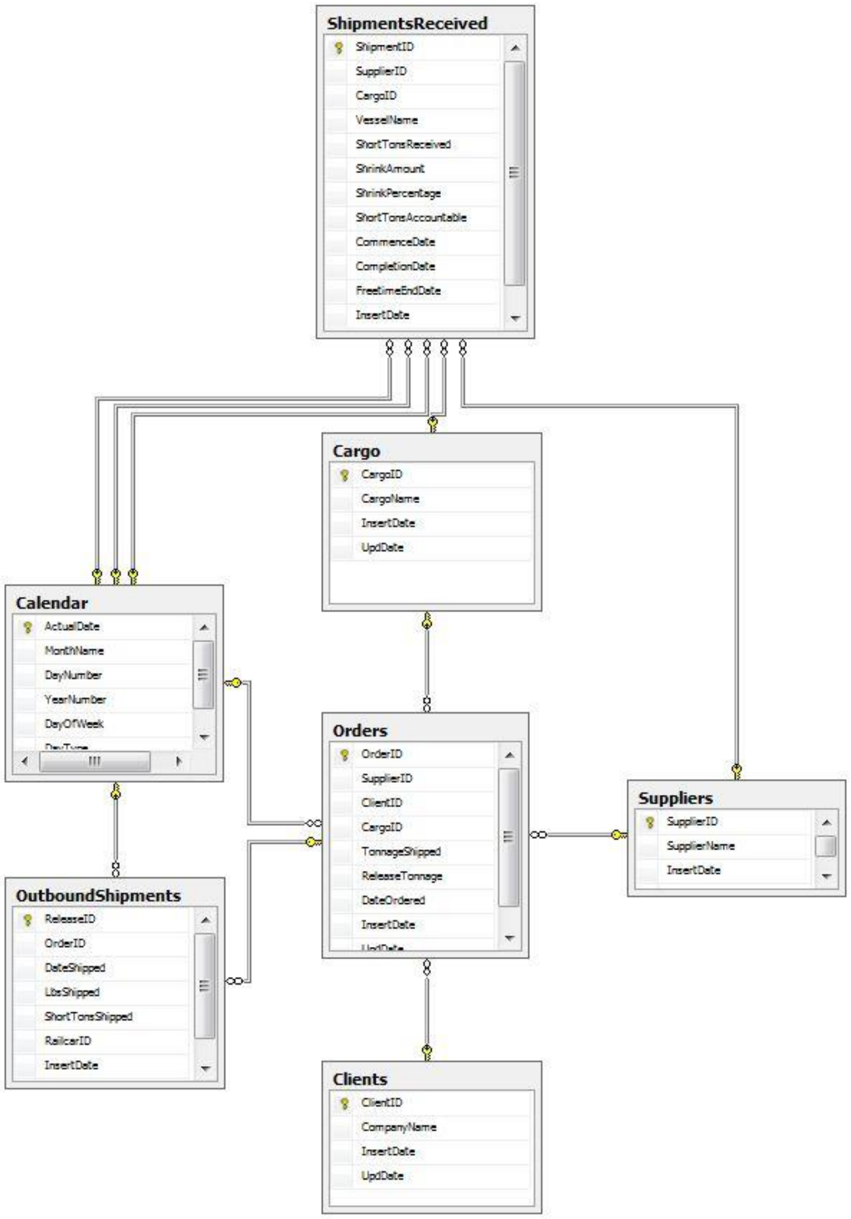
During the process of creating the database, Chris began working on a GUI that Matt and others at Texas International Terminals could easily use. Chris created the program using C# in Visual Studio. He spent many hours on the program, which eventually had somewhere from 2000 to 3000 lines of C# code. Chris connected the program to the database and utilized our stored procedures and triggers to create an excellent, user-friendly program. The program allowed users to edit (UPDATE in SQL) and create (INSERT) new rows. This functionality of the program is directly tied to the stored procedures that we created for that very purpose. Along with these options, the program is able to create useful reports specified by the user, such as total cargo shipped for a specific supplier or client. We determined which reports were most helpful by communicating with Matt; he told us what he would like to see in a report, so Chris created the flexibility for a user to choose the specifications for each report. There is also an option to export the data from the reports to Excel, where users can format the data, create graphs, and print.

The first time we showed Matt the program, we decided to use a free website that would allow him to remote in and use Chris's computer to access the program. When he saw the program and Chris began to explain what it could do, all he could say was WOW for about five solid minutes. He was very impressed with all that the program could do and the time that it could save him in the future. Hearing the excitement in Matt’s voice was very rewarding; we could tell that all of our work had made a difference for him. Unfortunately, Matt won’t be able to use the program immediately because Texas International Terminals does not have any server racks or DBMS installed on their network. We decided that doing that would have been a little outside the scope and scale of this project. On the bright side, the database and an excellent program to utilize it are ready for Texas International Terminals to use once they get everything else in place.

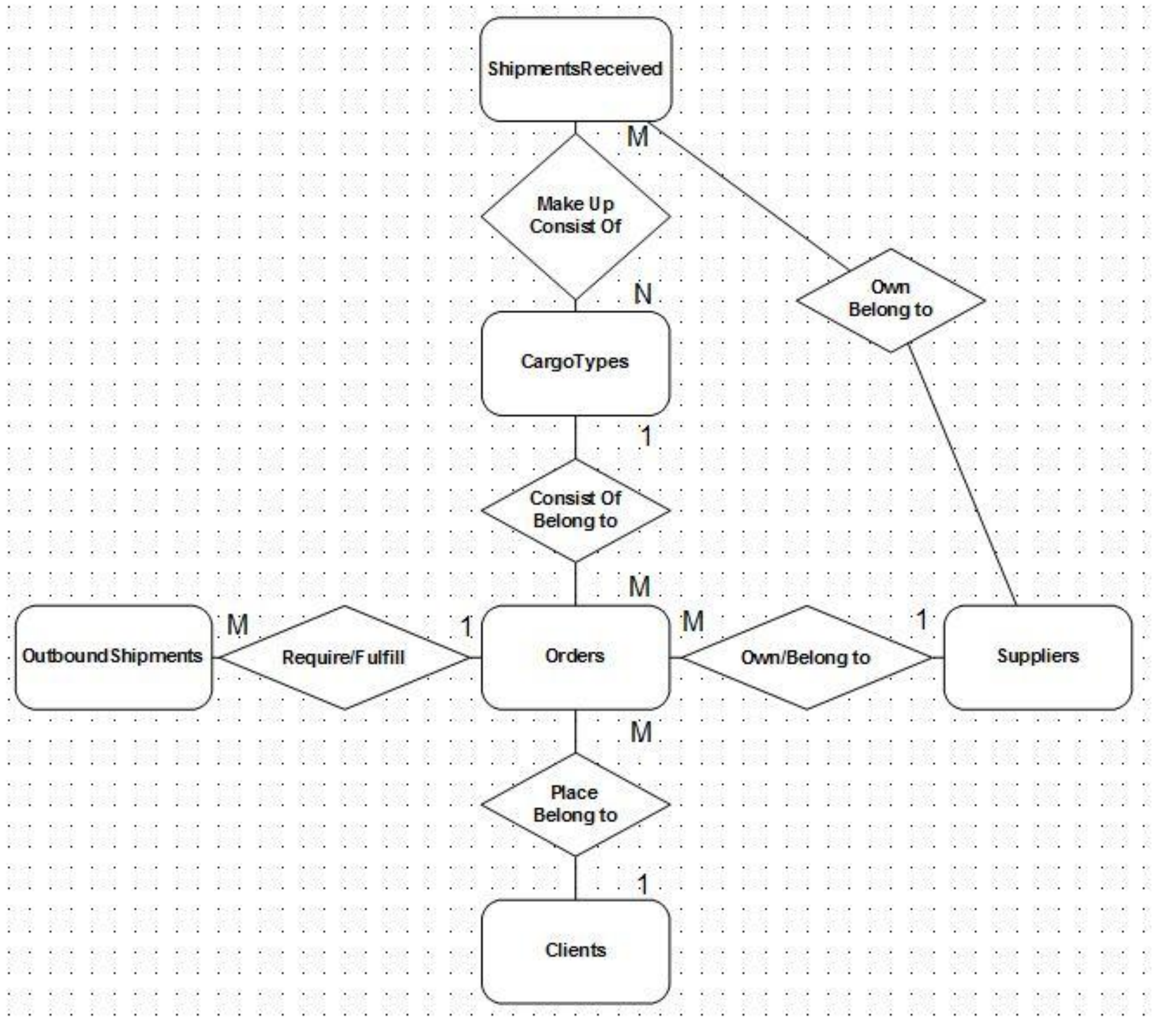
Throughout this process we have learned a great deal of very useful information that we would not have learned without this experience. We feel that the project was well worth our time and effort. Each member of the group contributed to the final product, and in the end we accomplished more *collectively* than we could have done *individually*. We experienced synergy.

# Diagrams

## Database Diagram



## ER Diagram


ShipmentsReceived
M
Make Up
Consist Of
N
Own
Belong to
CargoTypes
1
Consist Of
Belong to
M
OutboundShipments
M
Require/Fulfill
1
Orders
M
Own/Belong to
1
Suppliers
M
Place
Belong to
1
Clients

## Timeline

| | |
|---|---|
| December 21, 2010 | Clayton contacted Texas International Terminals about creating a database |
| December 29, 2010 | Received Excel spreadsheet data from Matt Haidinyak, Director of Sales and Traffic at Texas International Terminals |
| January 18, 2011 | Began organizing the data from Excel and creating an ER diagram |
| January 20, 2011 | Emailed Matt questions about the business process recorded in Excel |
| January 21, 2011 | Received Matt's response to questions |
| February 3, 2011 | Emailed Matt to schedule a time for him to walk us through the business process |
| February 7, 2011 | Arranged to get pictures of the shipping process |
| February 8, 2011 | Scheduled a phone meeting with Matt for February 9 |
| February 9, 2011 | Talked with Matt on the phone and discussed the business shipping process |
| February 17, 2011 | Modified ER Diagram in response to Matt's explanation |
| February 22, 2011 | Met with Dr. Olsen and got advice on diagram. Simplified and finalized the diagram |
| February 24, 2011 | Set up our database account with Dr. Olsen |
| March 1, 2011 | Created the necessary tables, attributes, relationships and constraints in SQL Server |
| March 10, 2011 | Created stored procedures and triggers |
| March 17, 2011 | Asked Matt questions about the database to make sure we were on track |
| March 22, 2011 | Chris Leitch began programming a user interface |
| March 24, 2011 | Jared Lyman, Clayton Fielding, and Jared Hardinger created stored procedures to insert data into tables |
| March 29, 2011 | Chris Leitch altered the shrink percent trigger. Jared Hardinger researched user defined functions. Clayton Fielding began working on our class presentation. Jared Lyman began documentation. |
| April 6, 2011 | Scheduled a meeting with Matt for him to test the database and user interface |

|  |  |
|---|---|
| April 8, 2011 | Met with Matt using the phone and a remote desktop connection. He was impressed by the interface. |
| April 12, 2011 | Added functions and reports to the interface including a summary of all shipments for each client (total tonnage shipped), a function to look up a shipment for a specific release number, the ability to change the shrinkage amount, fixed the function to add cargo to the database, added the ability to export reports to Excel, and a function to determine how much cargo a specific supplier still has in inventory. |
| April 14, 2011 | Fixed and updated stored procedures. Created nested triggers so the triggers would not call themselves |
| April 19, 2011 | Finished preparing presentation |
| April 20, 2011 | Practiced presentation in classroom |
| April 21, 2011 | Gave presentation in class |

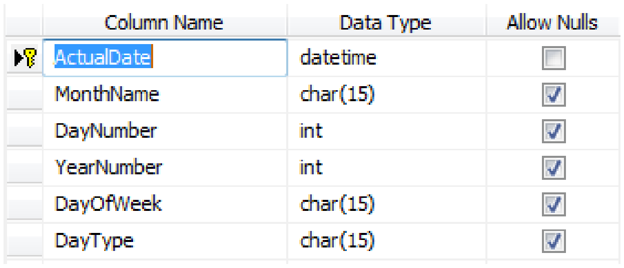
# User Manual

## The Database

The database is made up of 8 tables. Each table has two common columns: an insert date and an update date. These are simply used in order to tell us when a row was initially put into the database and then to tell us when they were updated. The insert date is added when we run a stored procedure to insert data, and the update date is changed by a trigger every time we change a row. That is all the explanation necessary for these common columns and I will not discuss them in the description of each table. I will cover each of these tables briefly:

### Calendar

The Calendar table contains the following columns:

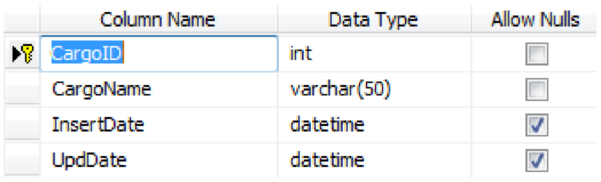


| | Column Name | Data Type | Allow Nulls |
|---|---|---|---|
| 🔑 | ActualDate | datetime | ☐ |
| | MonthName | char(15) | ✓ |
| | DayNumber | int | ✓ |
| | YearNumber | int | ✓ |
| | DayOfWeek | char(15) | ✓ |
| | DayType | char(15) | ✓ |

This is a simple calendar table. It contains dates, month names, day and year numbers, days of the week, and day types. We use this to validate dates in the tables. This may also be used in the future in case we ever need to know when weekends or things of that sort are.

### Cargo

The Cargo table contains the following columns:

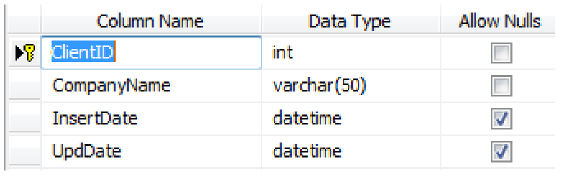


| | Column Name | Data Type | Allow Nulls |
|---|---|---|---|
| 🔑 | CargoID | int | ☐ |
| | CargoName | varchar(50) | ☐ |
| | InsertDate | datetime | ✓ |
| | UpdDate | datetime | ✓ |

The cargo table simply holds the Cargo ID and Cargo Name. This is useful to know what cargo has been received for a specific supplier and helps us know what is in our inventory of each type of cargo. The name is there solely for clarity and convenience. This table would interact in the same way even if it were just ID numbers.

## Clients

The Clients table contains the following columns:

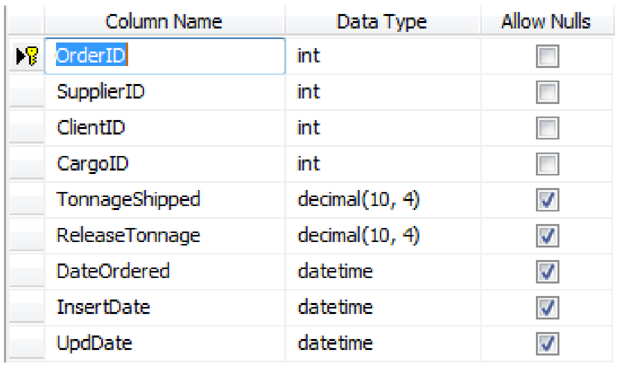


| Column Name | Data Type | Allow Nulls |
|---|---|---|
| ClientID | int | ☐ |
| CompanyName | varchar(50) | ☐ |
| InsertDate | datetime | ✓ |
| UpdDate | datetime | ✓ |

This table, much like the cargo table, only has two useful fields, a Client ID and a Client Name. This table is utilized when determining what client a shipment is going to and what client has ordered product.

## Orders

The Orders table contains the following columns:

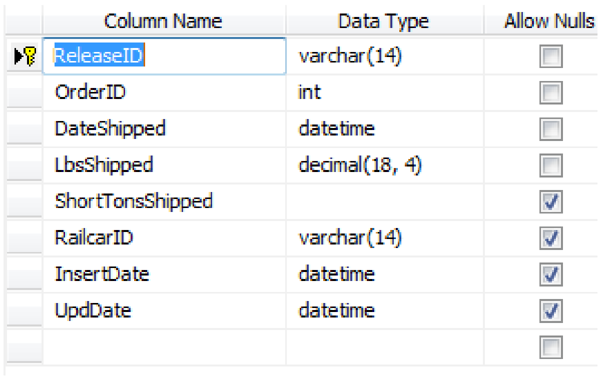


| Column Name | Data Type | Allow Nulls |
|---|---|---|
| OrderID | int | ☐ |
| SupplierID | int | ☐ |
| ClientID | int | ☐ |
| CargoID | int | ☐ |
| TonnageShipped | decimal(10, 4) | ✓ |
| ReleaseTonnage | decimal(10, 4) | ✓ |
| DateOrdered | datetime | ✓ |
| InsertDate | datetime | ✓ |
| UpdDate | datetime | ✓ |

This table holds orders for product. It tells us what supplier the product is coming from, what client ordered the product, and what cargo they ordered. It also contains the total tonnage shipped for this order (updated by a trigger in the Outbound Shipments table) and the release tonnage. These can be compared to see if an order has been completely filled or not. It also tells us the date it was ordered.

## Outbound Shipments

The Outbound Shipments table contains the following columns:

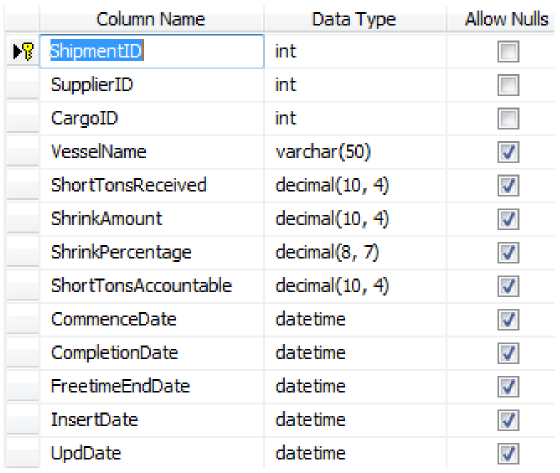


| Column Name | Data Type | Allow Nulls |
|---|---|---|
| ReleaseID | varchar(14) | ☐ |
| OrderID | int | ☐ |
| DateShipped | datetime | ☐ |
| LbsShipped | decimal(18, 4) | ☐ |
| ShortTonsShipped | | ✓ |
| RailcarID | varchar(14) | ✓ |
| InsertDate | datetime | ✓ |
| UpdDate | datetime | ✓ |
| | | ☐ |

This table holds all the shipments that we have sent to clients. It takes a Release ID primary key and an Order ID foreign key. This relates the table to the orders table so we know what order the shipment is for. We then hold the date the product was shipped and thou pounds shipped. Short Tons Shipped is a derived column that takes the pounds shipped and divides it by 2000. Railcar ID is only populated if a shipment has a railcar release number; otherwise, it is null.

## Shipments Received

The Shipments Received table contains the following columns:

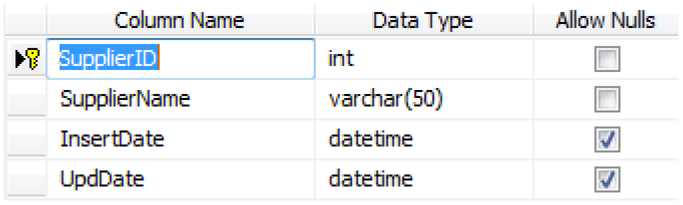


| Column Name | Data Type | Allow Nulls |
|---|---|---|
| ShipmentID | int | ☐ |
| SupplierID | int | ☐ |
| CargoID | int | ☐ |
| VesselName | varchar(50) | ✓ |
| ShortTonsReceived | decimal(10, 4) | ✓ |
| ShrinkAmount | decimal(10, 4) | ✓ |
| ShrinkPercentage | decimal(8, 7) | ✓ |
| ShortTonsAccountable | decimal(10, 4) | ✓ |
| CommenceDate | datetime | ✓ |
| CompletionDate | datetime | ✓ |
| FreetimeEndDate | datetime | ✓ |
| InsertDate | datetime | ✓ |
| UpdDate | datetime | ✓ |

This table holds all the shipments that we have received from suppliers. It contains a Shipment ID which is the primary key and also two foreign keys: Supplier ID and Cargo ID. In this way we can tell what supplier the shipment came from and the cargo we received. We also hold a Vessel Name that tells us the ship that the shipment came on. We enter in the Short Tons Received and the Shrink Percentage and triggers in the table then calculate the Shrink Amount and Short Tons Accountable. The user also enters a Commence Date (when the ship started being unloaded), a Completion Date (when we finished unloading the ship), and a Free Time End Date that tells us when the suppliers free time for this shipment ends.

## Suppliers

The Suppliers table contains the following columns:

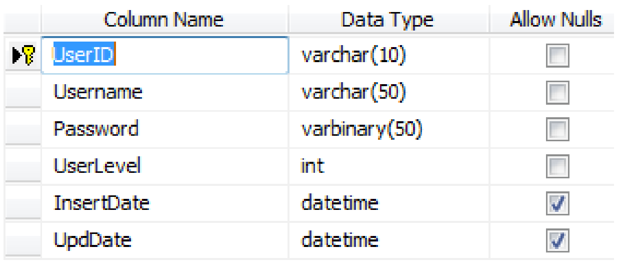


| Column Name | Data Type | Allow Nulls |
|---|---|---|
| SupplierID | int | ☐ |
| SupplierName | varchar(50) | ☐ |
| InsertDate | datetime | ✓ |
| UpdDate | datetime | ✓ |

This table simply holds a Supplier ID and a Supplier Name. We use this to tell us what shipments belong to what suppliers and also what suppliers are supplying product for what orders.

## Users

The Users table contains the following columns:



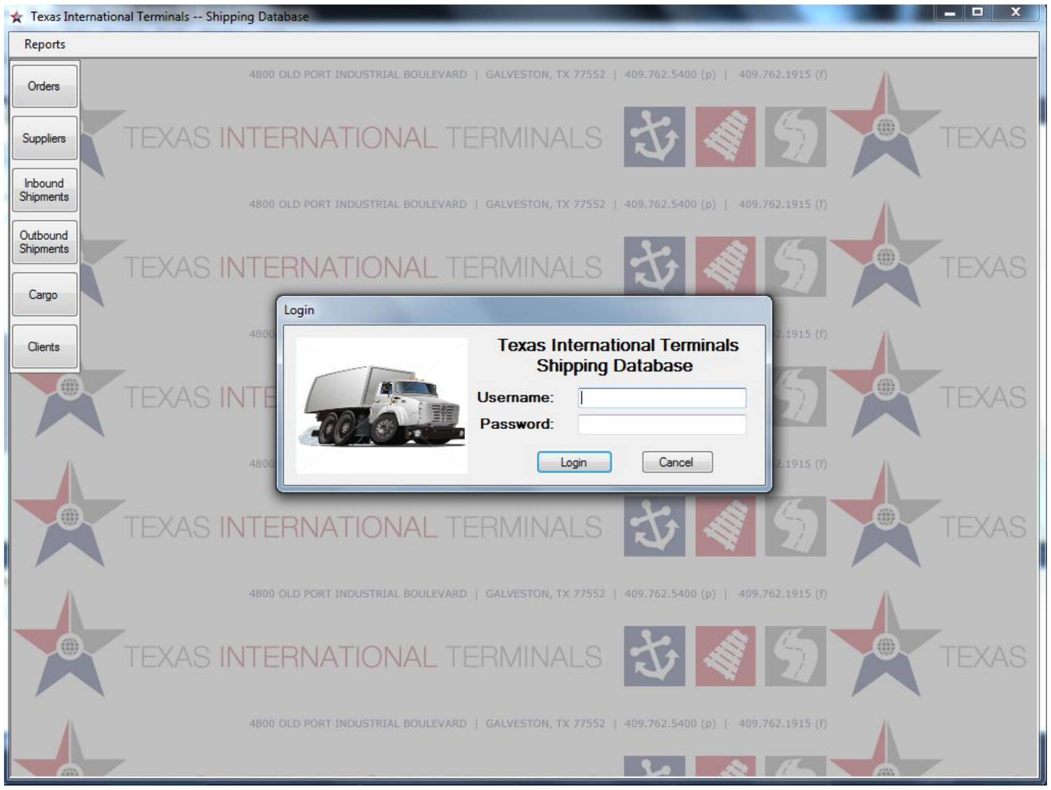
| Column Name | Data Type | Allow Nulls |
|---|---|---|
| UserID | varchar(10) | ☐ |
| Username | varchar(50) | ☐ |
| Password | varbinary(50) | ☐ |
| UserLevel | int | ☐ |
| InsertDate | datetime | ✓ |
| UpdDate | datetime | ✓ |

This table is used solely for the user interface. It holds a User ID which is the primary key. It also holds the username, the password (encrypted using PWDENCRYPT in SQL Server), and a user level which is used to tell us what things the logged in user can do in the user interface. Mainly, this is used to tell us if a user is an administrator or not.

## The User Interface

### Logging In

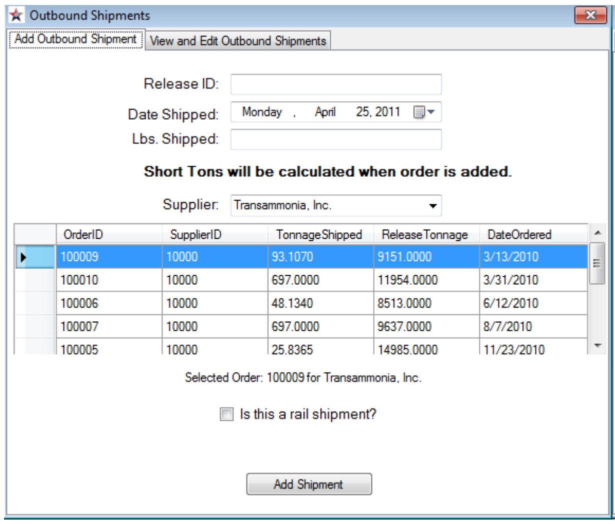
You may interact with the database using the program. When you first open the program you will be required to log in. The accounts are stored in the Users table and hold a user ID, username, password (encrypted using the build in SQL function) and permissions level. The permissions level determines whether or not you can delete data from the program.



If you do not have a username, you will not be able to interact with the program past this screen. You may click cancel to exit. Users that are added to the database while the program is running will not be able to log in until the program has been restarted.

## Modifying Data

The main screen of the program allows you to add, edit, and view all the main aspects of the database including: Orders, suppliers, shipments, cargo, and clients. You will find that the layout of each of these options is quite similar. They all have a screen that has two tabs, one for adding data, and one for viewing and modifying data, like so:



Release ID:
Date Shipped: Monday , April 25, 2011
Lbs. Shipped:

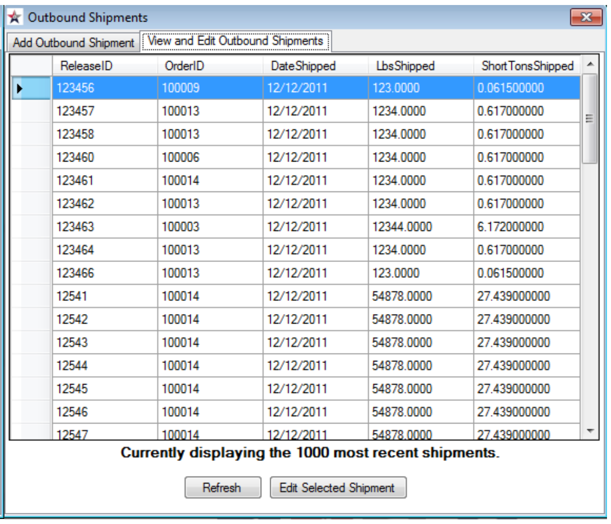
**Short Tons will be calculated when order is added.**

Supplier: Transammonia, Inc.

| | OrderID | SupplierID | TonnageShipped | ReleaseTonnage | DateOrdered |
|---|---|---|---|---|---|
| ▶ | 100009 | 10000 | 93.1070 | 9151.0000 | 3/13/2010 |
| | 100010 | 10000 | 697.0000 | 11954.0000 | 3/31/2010 |
| | 100006 | 10000 | 48.1340 | 8513.0000 | 6/12/2010 |
| | 100007 | 10000 | 697.0000 | 9637.0000 | 8/7/2010 |
| | 100005 | 10000 | 25.8365 | 14985.0000 | 11/23/2010 |

Selected Order: 100009 for Transammonia, Inc.

☐ Is this a rail shipment?



| | ReleaseID | OrderID | DateShipped | LbsShipped | ShortTonsShipped |
|---|---|---|---|---|---|
| ▶ | 123456 | 100009 | 12/12/2011 | 123.0000 | 0.061500000 |
| | 123457 | 100013 | 12/12/2011 | 1234.0000 | 0.617000000 |
| | 123458 | 100013 | 12/12/2011 | 1234.0000 | 0.617000000 |
| | 123460 | 100006 | 12/12/2011 | 1234.0000 | 0.617000000 |
| | 123461 | 100014 | 12/12/2011 | 1234.0000 | 0.617000000 |
| | 123462 | 100013 | 12/12/2011 | 1234.0000 | 0.617000000 |
| | 123463 | 100003 | 12/12/2011 | 12344.0000 | 6.172000000 |
| | 123464 | 100013 | 12/12/2011 | 1234.0000 | 0.617000000 |
| | 123466 | 100013 | 12/12/2011 | 123.0000 | 0.061500000 |
| | 12541 | 100014 | 12/12/2011 | 54878.0000 | 27.439000000 |
| | 12542 | 100014 | 12/12/2011 | 54878.0000 | 27.439000000 |
| | 12543 | 100014 | 12/12/2011 | 54878.0000 | 27.439000000 |
| | 12544 | 100014 | 12/12/2011 | 54878.0000 | 27.439000000 |
| | 12545 | 100014 | 12/12/2011 | 54878.0000 | 27.439000000 |
| | 12546 | 100014 | 12/12/2011 | 54878.0000 | 27.439000000 |
| | 12547 | 100014 | 12/12/2011 | 54878.0000 | 27.439000000 |

**Currently displaying the 1000 most recent shipments.**

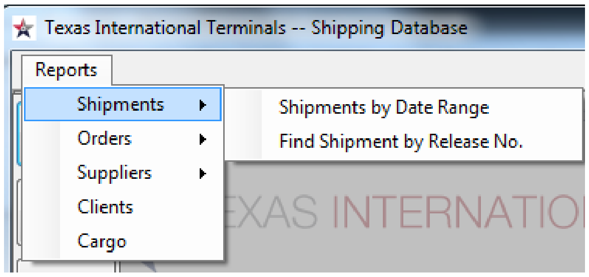
You may also right click on any of the items to delete them from the database. Note: You will only see this menu appear if you have administrator privileges. Also, some items will give you an error when you try to delete them if other pieces of data rely on them. For example, an order that has shipments tied to it cannot be deleted and will give you an error message.)

Also on this screen you will see a refresh button. This button is useful after you have added data and you want to verify it's accuracy without closing and reopening the edit window. It will pull the data from the database and display it in the output panel.

Beyond that, data may also be modified by clicking the edit button. This will give a screen almost identical to the add screen you see when entering data for the first time. The only difference being that the data is populated to reflect the current data in the database.

## Reports

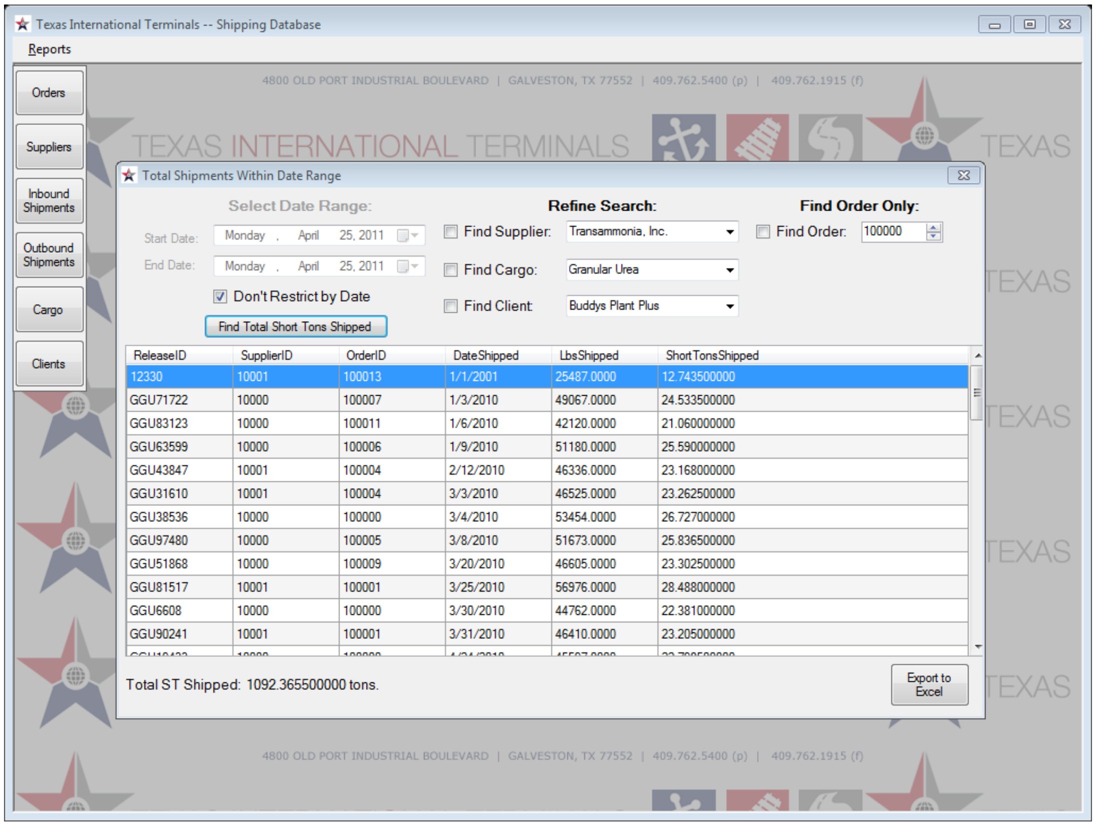
At the top of the main screen you will find a menu bar that allows you to view different reports about the data.



All reports can export the relevant data to excel to be transformed further (made into graphs, etc.) or be printed.

### *Total Shipments within Date Range*

This is an example report of all shipments in the Outbound Shipments table. You can see there are a number of check boxes by which to modify the results you get from the report. Here you can narrow down the results by supplier, cargo, client, date, or a combination of all four. You may also simply view all shipments ever made or find an order by order number.


Texas International Terminals -- Shipping Database
Reports
Orders
Suppliers
Inbound Shipments
Outbound Shipments
Cargo
Clients
4800 OLD PORT INDUSTRIAL BOULEVARD | GALVESTON, TX 77552 | 409.762.5400 (p) | 409.762.1915 (f)
TEXAS INTERNATIONAL TERMINALS
Total Shipments Within Date Range
Select Date Range:
Refine Search:
Find Order Only:
Find Total Short Tons Shipped
Export to Excel

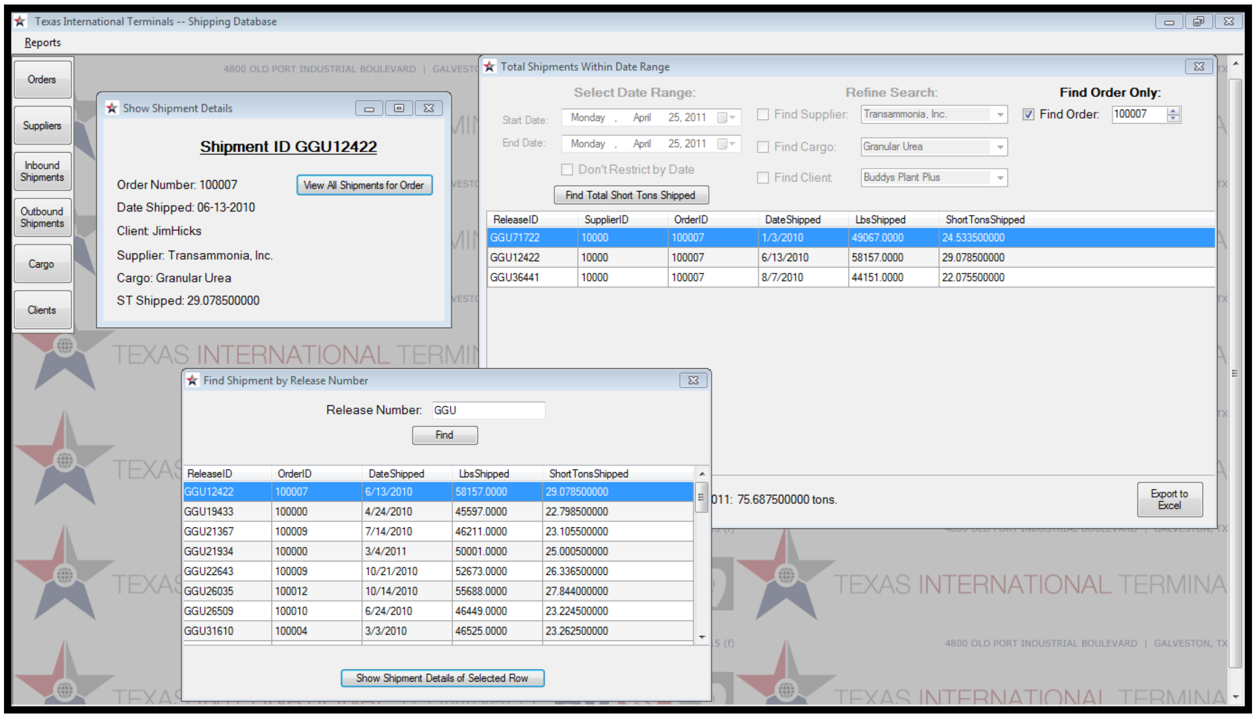
Start Date: Monday , April 25, 2011
End Date: Monday , April 25, 2011
[✓] Don't Restrict by Date
[ ] Find Supplier: Transammonia, Inc.
[ ] Find Cargo: Granular Urea
[ ] Find Client: Buddys Plant Plus
[ ] Find Order: 100000

| ReleaseID | SupplierID | OrderID | DateShipped | LbsShipped | ShortTonsShipped |
|---|---|---|---|---|---|
| 12330 | 10001 | 100013 | 1/1/2001 | 25487.0000 | 12.743500000 |
| GGU71722 | 10000 | 100007 | 1/3/2010 | 49067.0000 | 24.533500000 |
| GGU83123 | 10000 | 100011 | 1/6/2010 | 42120.0000 | 21.060000000 |
| GGU63599 | 10000 | 100006 | 1/9/2010 | 51180.0000 | 25.590000000 |
| GGU43847 | 10001 | 100004 | 2/12/2010 | 46336.0000 | 23.168000000 |
| GGU31610 | 10001 | 100004 | 3/3/2010 | 46525.0000 | 23.262500000 |
| GGU38536 | 10000 | 100000 | 3/4/2010 | 53454.0000 | 26.727000000 |
| GGU97480 | 10000 | 100005 | 3/8/2010 | 51673.0000 | 25.836500000 |
| GGU51868 | 10000 | 100009 | 3/20/2010 | 46605.0000 | 23.302500000 |
| GGU81517 | 10001 | 100001 | 3/25/2010 | 56976.0000 | 28.488000000 |
| GGU6608 | 10000 | 100000 | 3/30/2010 | 44762.0000 | 22.381000000 |
| GGU90241 | 10001 | 100001 | 3/31/2010 | 46410.0000 | 23.205000000 |

Total ST Shipped: 1092.365500000 tons.

## *Find Shipment by Release Number*

You may also find shipments based on their release number. This will find any shipments that have the search value in any portion of the release number. From here, you may see the shipment details of the selected shipment, and from there you may see all shipments for the order that shipment came from.


Texas International Terminals -- Shipping Database
Reports
Orders
Suppliers
Inbound Shipments
Outbound Shipments
Cargo
Clients
TEXAS INTERNATIONAL TERMINALS
4800 OLD PORT INDUSTRIAL BOULEVARD | GALVESTON, TX
Show Shipment Details
View All Shipments for Order
Total Shipments Within Date Range
Find Total Short Tons Shipped
Export to Excel
Find Shipment by Release Number
Find
Show Shipment Details of Selected Row

**Shipment ID GGU12422**

Order Number: 100007
Date Shipped: 06-13-2010
Client: JimHicks
Supplier: Transammonia, Inc.
Cargo: Granular Urea
ST Shipped: 29.078500000

Select Date Range:
Start Date: Monday , April 25, 2011
End Date: Monday , April 25, 2011
☐ Don't Restrict by Date

Refine Search:
☐ Find Supplier: Transammonia, Inc.
☐ Find Cargo: Granular Urea
☐ Find Client: Buddys Plant Plus

Find Order Only:
☑ Find Order: 100007

| ReleaseID | SupplierID | OrderID | DateShipped | LbsShipped | ShortTonsShipped |
|---|---|---|---|---|---|
| GGU71722 | 10000 | 100007 | 1/3/2010 | 49067.0000 | 24.533500000 |
| GGU12422 | 10000 | 100007 | 6/13/2010 | 58157.0000 | 29.078500000 |
| GGU36441 | 10000 | 100007 | 8/7/2010 | 44151.0000 | 22.075500000 |

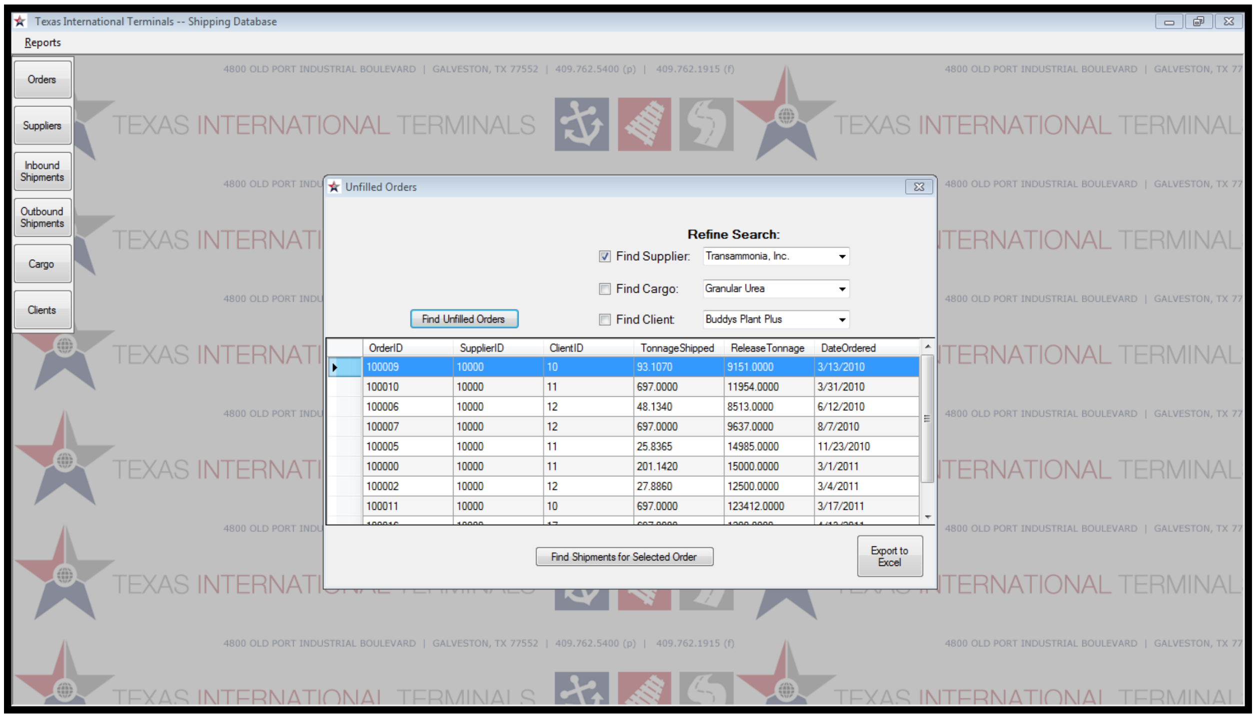
011: 75.687500000 tons.

Release Number: GGU

| ReleaseID | OrderID | DateShipped | LbsShipped | ShortTonsShipped |
|---|---|---|---|---|
| GGU12422 | 100007 | 6/13/2010 | 58157.0000 | 29.078500000 |
| GGU19433 | 100000 | 4/24/2010 | 45597.0000 | 22.798500000 |
| GGU21367 | 100009 | 7/14/2010 | 46211.0000 | 23.105500000 |
| GGU21934 | 100000 | 3/4/2011 | 50001.0000 | 25.000500000 |
| GGU22643 | 100009 | 10/21/2010 | 52673.0000 | 26.336500000 |
| GGU26035 | 100012 | 10/14/2010 | 55688.0000 | 27.844000000 |
| GGU26509 | 100010 | 6/24/2010 | 46449.0000 | 23.224500000 |
| GGU31610 | 100004 | 3/3/2010 | 46525.0000 | 23.262500000 |

## *Unfilled Orders*

This report displays all orders that do not have a shipped tonnage equal to or greater than the release tonnage. Within this report you may narrow it down by supplier, cargo, client, or some combination of the three. You may also find the shipments for the selected order based on what row you have selected in the output panel.



Refine Search:

[✓] Find Supplier: Transammonia, Inc.

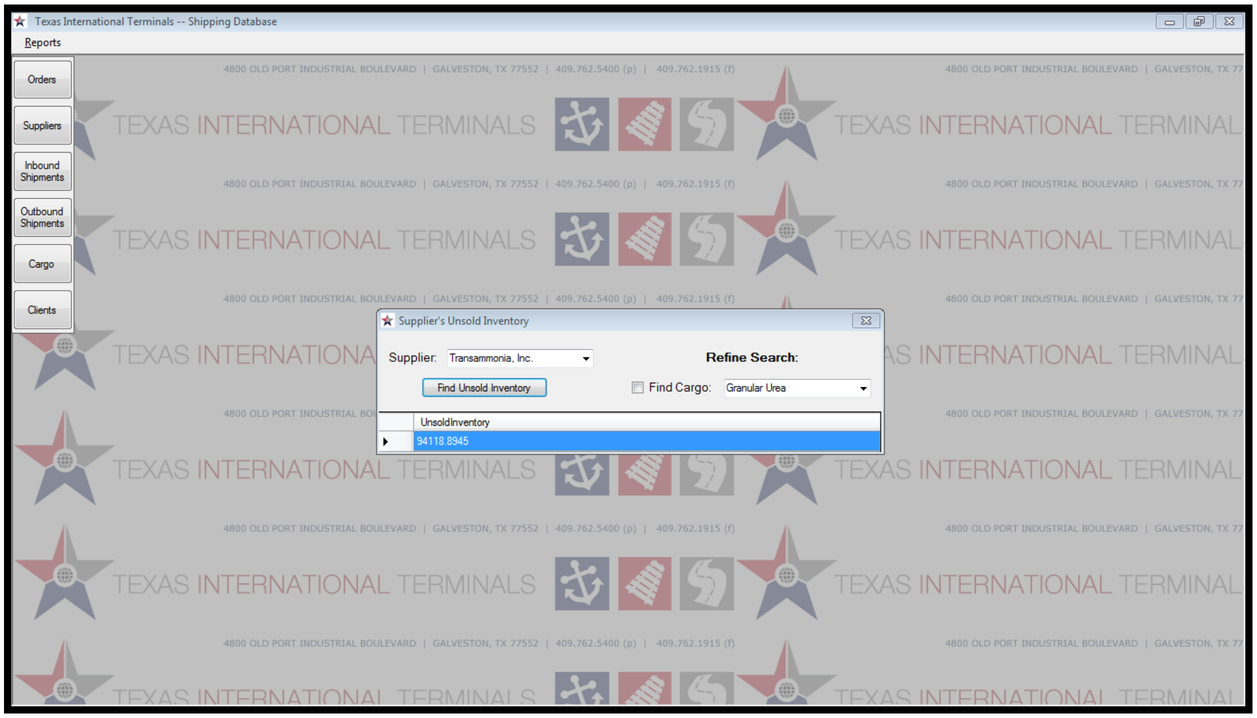
Find Cargo: Granular Urea

Find Client: Buddys Plant Plus

| OrderID | SupplierID | ClientID | TonnageShipped | ReleaseTonnage | DateOrdered |
|---|---|---|---|---|---|
| 100009 | 10000 | 10 | 93.1070 | 9151.0000 | 3/13/2010 |
| 100010 | 10000 | 11 | 697.0000 | 11954.0000 | 3/31/2010 |
| 100006 | 10000 | 12 | 48.1340 | 8513.0000 | 6/12/2010 |
| 100007 | 10000 | 12 | 697.0000 | 9637.0000 | 8/7/2010 |
| 100005 | 10000 | 11 | 25.8365 | 14985.0000 | 11/23/2010 |
| 100000 | 10000 | 11 | 201.1420 | 15000.0000 | 3/1/2011 |
| 100002 | 10000 | 12 | 27.8860 | 12500.0000 | 3/4/2011 |
| 100011 | 10000 | 10 | 697.0000 | 123412.0000 | 3/17/2011 |

### *Supplier's Unsold Inventory*

This report displays a supplier's unsold inventory. This report also allows you to narrow down your result by cargo so you may find the amount of a specific cargo that a supplier has not yet sold. This report does not allow you to export it to Excel as it consists of a single row of data.



Supplier: Transammonia, Inc.

Find Cargo: Granular Urea

| UnsoldInventory |
|---|
| 94118.8945 |

## Conclusion

This covers all the capabilities of the program at present. There is much room for improved report generation and a greater variety of reports available. Also, there is currently no convenient method to modify data for the Orders table due to time constraints and this is something that should be added before the application is considered to be fully usable for production purposes.

Other areas that are suggested to be modified:

- The way that data is pulled may be heavy on the database. A possible course of action is to pull all the data when the program first opens and then only query the database when you have modified the data. This may be inconvenient if multiple people will be modifying data simultaneously, but will put less load on the server.
- The password should be better encrypted by some sort of hashing mechanism and on top of that the hash should then be salted to ensure maximum security.
- There could be some more constraints in order to reduce human error. For example, some text boxes should only allow the input of digits (one or two do this at present). Also, the text of textboxes is input directly into a string that turns into a query run against the database. This leaves the program open to SQL injection, both malicious and inadvertent.

# Appendix

## Stored Procedures

We used stored procedures to insert any new information into the database and for calculations.

### Insert Cargo

This is when TIT is starting to handle a new product and needs to insert a new product into the database.

```
ALTER PROCEDURE [dbo].[sp_InsertCargo]
      @CargoName        VarChar(50)


AS
IF NOT EXISTS (SELECT *
                              FROM Cargo
                              WHERE CargoName=@CargoName)

BEGIN
      INSERT INTO Cargo
            VALUES (
                  (SELECT MAX(CargoID)+1
                  FROM Cargo),
                  @CargoName,
                  GETDATE(),
                  NULL
                  )
      END

ELSE
      BEGIN
            RAISERROR ('This cargo name is already in database',11,1)
      END;
```

### Suppliers Unsold Inventory

This stored procedure gives the unsold inventory for a given supplier

```
ALTER PROCEDURE [dbo].[sp_SuppliersUnsoldInventory] @supplierID INT
AS
SELECT s.SupplierName AS Supplier, (SELECT SUM(ShortTonsReceived)
                                    FROM ShipmentsReceived
                                    WHERE SupplierID = @supplierID) -
o.TotalShipped AS UnsoldInventory
      FROM Suppliers AS s

JOIN

      (SELECT SupplierID, SUM(ReleaseTonnage) AS TotalShipped
            FROM Orders
            WHERE SupplierID = @supplierID
       GROUP BY SupplierID) AS o
ON (s.SupplierID = o.SupplierID)
```

### Total Volume Shipped Per Client and Data

This give the total volume shipped for a given client between a beginning date and end date.

```
ALTER PROCEDURE [dbo].[sp_TotalVolumeShippedPerClientAndDate]
(
      @clientID VarChar(50),
      @StartDate Date,
      @EndDate Date
)
AS
BEGIN


      SELECT SUM(o.ReleaseTonnage) AS ReleaseTonnage
            FROM Clients AS c JOIN Orders AS o ON
                  (c.ClientID=o.clientId)
            WHERE o.ClientID = @clientID AND o.DateOrdered BETWEEN @StartDate
AND @EndDate
END;
```

## Triggers

### Update Date

This trigger we put on every table to insert and update the date when a tuple is updated.

```
/*This trigger inserts/updates the data in the UpdDate attribute when a tuple
is updated*/

ALTER TRIGGER [dbo].[tr_ClientsUpDateUpdDate]
ON [dbo].[Clients]
FOR UPDATE
AS
UPDATE Clients SET Clients.UpdDate=getdate()
FROM Clients INNER JOIN Inserted ON Clients.ClientID= Inserted.ClientID
```

### Update Short Tons Shipped for Updates and Inserts

These triggers will update the short tons shipped when a shipment is either entered or updated.

```
ALTER TRIGGER [dbo].[updShortTonsShipped]
ON [dbo].[OutboundShipments]
FOR insert
AS
DECLARE @OrderID AS int
DECLARE @LbsShipped AS int
DECLARE @ShortTonsShipped AS decimal(10,4)
SET @OrderID = (SELECT OrderID FROM inserted)
SET @LbsShipped =
      (SELECT SUM(LbsShipped)
            FROM OutboundShipments
            WHERE OrderID = @OrderID)
SET @ShortTonsShipped = (@LbsShipped / 2000)
UPDATE Orders
      SET TonnageShipped = (SELECT SUM(ShortTonsShipped) FROM
OutboundShipments WHERE OrderID = @OrderID)
      WHERE OrderID = @OrderID
```

```
ALTER TRIGGER [dbo].[updShortTonsShippedUpdated]
ON [dbo].[OutboundShipments]
FOR update
AS
DECLARE @i INT, @d INT
SELECT @i = COUNT(*) FROM inserted;
SELECT @i = COUNT(*) FROM deleted;
DECLARE @OrderID AS int
DECLARE @LbsShipped AS int
DECLARE @ShortTonsShipped AS decimal(10,4)
SET @OrderID = (SELECT OrderID FROM deleted)
SET @LbsShipped =
      (SELECT SUM(LbsShipped)
            FROM OutboundShipments
            WHERE OrderID = @OrderID)
SET @ShortTonsShipped = (@LbsShipped / 2000)
UPDATE Orders
      SET TonnageShipped = (SELECT SUM(ShortTonsShipped) FROM
OutboundShipments WHERE OrderID = @OrderID)
      WHERE OrderID = @OrderID
```

### Update Accountable Shipments Received

This trigger calculates the amount of product that TIT is accountable for from each shipment.

Without the TRIGGER_NESTLEVEL, the trigger would repeat itself, With the TRIGGER_NESTLEVEL, it only fires once.

```
ALTER TRIGGER [dbo].[updAccountableShipmentReceived]

ON [dbo].[ShipmentsReceived]
FOR update, insert AS
IF TRIGGER_NESTLEVEL() > 1
      RETURN
DECLARE @ShrinkPercentage AS decimal(8,7)
DECLARE @ShrinkWeight AS decimal(10,4)
DECLARE @ShipmentID AS int
SET @ShrinkPercentage = (SELECT ShrinkPercentage FROM inserted)
SET @ShrinkWeight = (SELECT ShortTonsReceived FROM inserted) *
@ShrinkPercentage
SET @ShipmentID = (SELECT ShipmentID FROM inserted)
            UPDATE ShipmentsReceived
                  SET ShortTonsAccountable = (SELECT ShortTonsReceived FROM
inserted) - @ShrinkWeight,
                  ShrinkAmount = @ShrinkWeight WHERE ShipmentID = @ShipmentID
```

# User Interface Snippets

## Query Generation for Find Shipments by Date Range Report

This code generates a query based on different options selected in the “Find Shipments by Date Range” report. It also determines an appropriate label to display the result for the user.

```
        private void btnTotalShipped_Click_1(object sender, EventArgs e)
        {
            string query = "";
            string date1 = dateTimePicker1.Value.ToString("MM/dd/yyyy");
            string date2 = dateTimePicker2.Value.ToString("MM/dd/yyyy");

            if (chkFindOrder.Checked)
            {
                query = "SELECT ReleaseID, SupplierID, o.OrderID, DateShipped, 
LbsShipped, ShortTonsShipped FROM OutboundShipments AS os JOIN Orders AS o ON (os.OrderID 
= o.OrderID) WHERE os.OrderID = " + numOrderNo.Value.ToString() + " ORDER BY 
DateShipped;";
            }
            else
            {
                if (!chkDateRestriction.Checked)
                {
                    //Determining the Query
                    query = "SELECT ReleaseID, SupplierID, o.OrderID, DateShipped, 
LbsShipped, ShortTonsShipped FROM OutboundShipments AS os JOIN Orders AS o ON (os.OrderID 
= o.OrderID) WHERE DateShipped BETWEEN '" + date1 + "' AND '" + date2 + "' ";
                    if (chkSuppliers.Checked == true)
                    {
                        query = query + "AND SupplierID = " + 
cmbSupplierNames.SelectedValue;
                    }
                    if (chkCargo.Checked == true)
                    {
                        query = query + "AND CargoID = " + cmbCargo.SelectedValue;
                    }
                    if (chkClient.Checked == true)
                    {
                        query = query + "AND ClientID = " + cmbClient.SelectedValue;
                    }
                    query = query + " ORDER BY DateShipped;";
                }
                else
                {
                    //Determining the Query
                    query = "SELECT ReleaseID, SupplierID, o.OrderID, DateShipped, 
LbsShipped, ShortTonsShipped FROM OutboundShipments AS os JOIN Orders AS o ON (os.OrderID 
= o.OrderID) WHERE 1=1 ";
                    if (chkSuppliers.Checked == true)
                    {
                        query = query + "AND SupplierID = " + 
cmbSupplierNames.SelectedValue;
                    }
                    if (chkCargo.Checked == true)
                    {
                        query = query + "AND CargoID = " + cmbCargo.SelectedValue;
                    }
                    if (chkClient.Checked == true)
                    {
                        query = query + "AND ClientID = " + cmbClient.SelectedValue;
                    }
                    query = query + " ORDER BY DateShipped;";
                }
            }

            DataSet ds2 = new DataSet();

            SqlDataAdapter dbOrders = new SqlDataAdapter(query, myConn);

            dbOrders.FillSchema(ds2, SchemaType.Source);
            dbOrders.Fill(ds2);

            string totalShipped = ds2.Tables[0].Compute("SUM(ShortTonsShipped)",
string.Empty).ToString();
            if (totalShipped == "" || totalShipped == null)
            {
                lblShippedSum.Text = "";
            }
            else
            {
                lblShippedSum.Text = totalShipped + " tons.";
            }

            dsExp = ds2;
            dgvOutput.DataSource = ds2.Tables[0];
            dgvOutput.Columns[dgvOutput.Columns.Count - 1].AutoSizeMode =
DataGridViewAutoSizeColumnMode.Fill;

            if (totalShipped == "" || totalShipped == null)
            {
                if (chkDateRestriction.Checked)
                {
                    lblTotalSTShipped.Text = "Nothing has been shipped";
                }
                else
                {
                    lblTotalSTShipped.Text = "Nothing was shipped from " + date1 + "-" +
date2;
                }
            }
            else
            {
                if (chkDateRestriction.Checked)
                {
                    lblTotalSTShipped.Text = "Total ST Shipped";
                }
                else
                {
                    lblTotalSTShipped.Text = "Total ST Shipped from " + date1 + "-" +
date2;
                }
            }

            if(chkSuppliers.Checked == true)
            {
                dsSuppliers.Tables[0].PrimaryKey = new System.Data.DataColumn[]
{dsSuppliers.Tables[0].Columns[0]};
                DataRow foundRow =
dsSuppliers.Tables[0].Rows.Find(cmbSupplierNames.SelectedValue);
                lblTotalSTShipped.Text += " for " + foundRow[1];
            }
            if (chkCargo.Checked == true)
            {
                dsCargo.Tables[0].PrimaryKey = new System.Data.DataColumn[] {
dsCargo.Tables[0].Columns[0] };
                DataRow foundRow = dsCargo.Tables[0].Rows.Find(cmbCargo.SelectedValue);
                lblTotalSTShipped.Text += " of " + foundRow[1];
            }
            if (chkClient.Checked == true)
            {
                dsClients.Tables[0].PrimaryKey = new System.Data.DataColumn[] {
dsClients.Tables[0].Columns[0] };
                DataRow foundRow =
dsClients.Tables[0].Rows.Find(cmbClient.SelectedValue);
                lblTotalSTShipped.Text += " to " + foundRow[1];
            }

            lblTotalSTShipped.Text += ":";
            lblShippedSum.Left = lblTotalSTShipped.Right-3;
        }
```

## Exporting a DataSet to Excel

This exports a DataSet to an Excel spreadsheet. It converts the column names of the given DataSet to bold and underlined header cells.

```
        public void export(DataSet ds)
        {
            try
            {
                //Create a new Excel Application instance, with a new workbook with one
worksheet.
                excelApp = new Microsoft.Office.Interop.Excel.Application();
                workbook = excelApp.Workbooks.Add(XlWBATemplate.xlWBATWorksheet);
                sheet = (Worksheet)workbook.Worksheets[1];

                //Rename the worksheet.
                sheet.Name = "Yodito Computer Inventory";

                DataRow[] foundRows;
                int numCols = ds.Tables[0].Columns.Count;

                //Create an array from our Dataset.
                foundRows = ds.Tables[0].Select();

                //Make the header cells.
                sheet.get_Range("A2", "L2").Font.Bold = true;
                sheet.get_Range("A2", "L2").Font.Underline = true;

                //Populate the rows below until we've populated all the rows we found in
our Dataset.
                for (int j = 0; j < foundRows.Length; j++)
                {
                    for (int i = 0; i < numCols; i++)
                    {
                        sheet.Cells[2, i + 1] = ds.Tables[0].Columns[i].Caption;
                        sheet.Cells[j + 3, i + 1] = foundRows[j][i];
                    }
                }

                //Once everything is populated, show us the Excel application.
                excelApp.Visible = true;
            }
            finally
            {
                //Empty the objects we used.
                excelApp = null;
                workbook = null;
                sheet = null;
            }
        }
```

## A Class to Add a Context Menu to a DataGridView

This class adds a context menu to a data grid view that gives the user the option to delete a cell. It also handles the ability to right click on a row and select it (as opposed to the default of only being able to left click and select a row)

```
    class addDGVContextMenu
    {
        static ContextMenuStrip mnu = new ContextMenuStrip();
        static ToolStripMenuItem mnuDelete = new ToolStripMenuItem("Delete");
        static DataGridView dgvSelect;
        static string t;
        static string id;
        static SqlConnection myConn;
        static Button refresh;

        public static int userLevel;

        public static void addMenu(DataGridView dgv, string Table, string pkID,
SqlConnection conn, Button btnRefresh)
        {
            mnu = new ContextMenuStrip();
            mnuDelete = new ToolStripMenuItem("Delete");
            dgvSelect = null;
            t = null;
            id = null;

            if (userLevel == 1)
            {
                //Assign event handlers
                mnuDelete.Click += new EventHandler(mnuDelete_Click);
                //Add to main context menu
                mnu.Items.AddRange(new ToolStripItem[] { mnuDelete });
                //Assign to datagridview
                dgvSelect = dgv;
                dgvSelect.CellMouseClick += new
DataGridViewCellMouseEventHandler(dgvClick);
                t = Table;
                id = pkID;
                myConn = conn;
                refresh = btnRefresh;
            }
        }

        private static void dgvClick(object sender, DataGridViewCellMouseEventArgs e)
        {
            if (e.RowIndex >= 0 && e.ColumnIndex >= 0 && e.Button == MouseButtons.Right)
            {
                dgvSelect.ClearSelection();
                dgvSelect.Rows[e.RowIndex].Selected = true;
                Rectangle r = dgvSelect.GetCellDisplayRectangle(e.ColumnIndex,
e.RowIndex, true);

                mnu.Show((Control)sender, r.Left + e.X, r.Top + e.Y);
            }
        }

        private static void mnuDelete_Click(object sender, EventArgs e)
        {
            string query = "";

            try
            {
                if (dgvSelect.SelectedRows[0].Cells[0].ValueType.ToString() ==
"System.String")
                {
                    query = "DELETE FROM " + t + " WHERE " + id + " = '" +
(string)dgvSelect.SelectedRows[0].Cells[0].Value.ToString() + "';";
                }
                else
                {
                    query = "DELETE FROM " + t + " WHERE " + id + " = " +
dgvSelect.SelectedRows[0].Cells[0].Value.ToString() + ";";
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error occurred.");
                return;
            }

            if (MessageBox.Show("Are you sure you want to delete this row?", "Are you
sure?", MessageBoxButtons.YesNo) == DialogResult.Yes)
            {
                SqlCommand insertCommand = new SqlCommand(query, myConn);

                try
                {
                    insertCommand.ExecuteNonQuery();
                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.ToString());
                    return;
                }
                MessageBox.Show("Row deleted.", "Deleted.");
                refresh.PerformClick();
            }
            else
            {
                MessageBox.Show("User cancelled row deletion.", "Deletion cancelled.");
            }
        }
    }
```