Blockhouse Coffee & Kitchen
Inventory Management System
By Ace Systems Solutions

Team Members:

Joshua Wilson, Jake Simpson, Jorge Vasquez, Andy Luong, Dan Thomas, Andres Pirela, Jesse Requena, Francisco Gomez

Table of Contents

Executive Summary

Objectives

The objectives for this project is to implement a system which allows them to accurately and easily track their inventory. This can be done by developing an application connecting to a database in order to store and retrieve information. Our goal is to enable the organization, Blockhouse Coffee & Kitchen, to more easily and accurately gather and store information from their system. By doing this, we increase their efficiency, productivity, and reduce overhead costs.

Current Business Problems & Challenges

Blockhouse Coffee & Kitchen would like to expand its operations by opening more locations within the Houston area. However, they are experiencing issues that they believe would hinder this goal were they to actually do so. Specifically, their current way of managing inventory is not quick or even accurate in some cases. This leads to increased overhead costs when items are either overstocked, understocked, or ordered in incorrect amounts as a result of counting errors. Consequently, the current system is not suited to larger business operations.

Solution

During our initial interview with operations manager Brian Alfaro, we learned that while they run a fairly smooth operation, there were undoubtedly some areas where we felt we could greatly improve their efficiency and save the company money. When we asked Brian how they kept track of their inventory, he told us that the store managers conduct counts and enter them into spreadsheets on the company's Google Drive. When we continued to question him on this system, we learned that counts on higher usage items like coffee, milk, and produce are conducted at the end of every week, and a complete inventory count is conducted at the end of

every month. Brian was not hesitant to admit that their current system was inefficient and becoming more problematic as the company aged. Based on Brian's struggles, we proposed a current solution which would be to create a Graphical User Interface for a SQL Database that would allow the Owner and Managers to add, remove, edit, update items in the inventory, as well as update other elements. This solution would allow the store to accurately keep track of their work processes. Ultimately, this would automate the process that the café managers currently have to do.

Client Information

Relevant Client Information

- Owner: Cody Frederick
- Company Name: Blockhouse Coffee & Kitchen
- Address: 611 Jackson St., Suite C. Richmond, TX 77469

Client's Issues & Requirements

Our client currently updates inventory records manually. Important items counted weekly and the rest counted once a month. Under this system low priority items are so rarely counted that it is difficult to know how much of each item they have. The current inventory system is also susceptible to human error. If an item is miscounted it will take between a week and a month for that error to be corrected. The current system lacks methods for simple updates or corrections.

After inventory is collected it is put into a locally stored spreadsheet. Manually moving data between these two spreadsheets is reducing the efficiency of their data collection.

In terms of the requirements, a new system should be implemented through an application that is connected to a database. The requirements for this system are listed below.

- System should have complete list of inventoried items
- Only upper management should have permission to edit the list of items(add/remove/update)
- System should be easy to understand for new users
- System should be an available option if the business decides to use it with or without their current system
- System should be finished by Spring 2021

Benefits and Costs

The benefits of an inventory management system include but are not limited to, accurate planning; an efficient inventory management system allows the organization to plan for the future whether it be long or short term. Being able to keep a close watch on stocks of all kinds of products, will help in managing and re-ordering. The aspect of centralized storage helps in managing goods and products at different locations. The largest benefit would be improved sales and productivity by keeping complete track of all stocks, reducing loss of customers and risk of mismanagement errors.

We will compare Ace's benefits and costs ratio for the inventory application compared to the cheapest inventory management software currently on the market, QuickBooks Enterprise.

	Hypothetical $5,000 in weekly sales		
	Option 1: Ace Application	Calculations	Total
	Benefits		
A)	Increased Effeciency	10% ↑	$ 5,500.00
B)	Reduced Shrink	15% ↑	$ 5,750.00
C)	Decreased Inventory Management Time	22% ↑	$ 6,100.00
D)	**Total Benefits (A+B+C)**		**$ 17,350.00**
	Costs		
E)	Application Cost	550$	$ 550.00
F)	Application Maintenance Cost	25$	$ 25.00
G)	Application Training Cost	500$	$ 500.00
H)	**Total Costs (E+F+G)**		**$ 1,075.00**
	Benefits Cost Ratio (D/H)	16.13953488	

	Hypothetical $5,000 in weekly sales		
	Option 2: Quick Books	Calculations	Total
	Benefits		
A)	Increased Effeciency	10% ↑	$ 5,500.00
B)	Reduced Shrink	15% ↑	$ 5,750.00
C)	Decreased Inventory Management Time	22% ↑	$ 6,100.00
D)	**Total Benefits (A+B+C)**		**$17,350.00**
	Costs		
E)	Application Cost	1500$ yearly	$ 1,500.00
F)	Application Maintenance Cost	0$	$ -
G)	Application Training Cost	500$	$ 500.00
H)	**Total Costs (E+F+G)**		**$ 2,000.00**
	Benefits Cost Ratio (D/H)	8.675	

Everything being the same except the cost of the applications, the benefits and costs ratio is almost double for the ACE built inventory management system compared to the QuickBooks application. It makes fiscal sense to integrate the system compared to not having a system at all or integrating another type of system that is on the market now.

Project Approach

For our database design, the team decided to design the database around the Blockhouse Coffee & Kitchen's processes. This meant that most of the database is populated with supplier and store information. Some examples of this is that we have created tables for the supplier's dairy and meat products. Also, other tables such as dairy and meat quantity for the store were created in correlation to the suppliers. Other various tables such as invoices, staff, and event tables were created to round out the database. For implementation, our method for doing so is to populate the database with test data similar to the inputs we would expect from actual use. This way, we can ensure that we thoroughly test our application and remove bugs.

In summary from the Analysis class, we have established our goals going into the next class. Blockhouse Coffee & Kitchen is having trouble with having an efficient, effective, and accurate way of managing inventory. With this information, we have made the decision to create an application that can satisfy their requirements. We will make an application that has a log-in menu that allows users to manage items in the inventory, such as adding, deleting, and updating items. In terms of the work we have completed. The team has completed many deliverables from the last class such as the unique reports and data flow diagrams. The most important deliverables that were completed were the ERD, data dictionary, and application prototype. These three tasks carried over to this current class and assisted in continuing the project. Although some changes have been made in order to make the project more feasible, the goal and desired outcome has remained unchanged.

Create a Graphical User Interface for a SQL Database with a backup in the cloud that would allow the Owner and Managers to add, remove, create, edit, update items in the inventory.

Solution

weekly/monthly/yearly reports, and update employees. This solution would also update and display the minimum items needed per week and/or per month for the café and when new orders for these items should be placed to maintain the right amount of stock. This would automate the process that the café managers currently have to do.

Queries/Reports Overview

ORDERING REPORTS - These are used to create reports on the current quantities of product types for a given store. We then use the other queried data to automatically determine the quantity of each product that should be ordered, their subtotals, and the total.

1. For showing current bread quantity (if current less than par) of a particular store.
    a. View name: **bread_qty_to_order**
    b. WHERE store_id = {} and current_case_qty < par_case_qty
2. For showing current dairy quantity (if current less than par) of a particular store.
    a. View name: **dairy_qty_to_order**
    b. WHERE store_id = {} and current_case_qty < par_case_qty
3. For showing current coffee quantity (if current less than par) of a particular store.
    a. View name: **coffee_qty_to_order**
    b. WHERE store_id = {} and current_case_qty < par_case_qty
4. For showing current meat quantity (if current less than par) of a particular store.
    a. View name: **meat_qty_to_order**
    b. WHERE store_id = {} and current_case_qty < par_case_qty
5. For showing current produce quantity (if current less than par) of a particular store.
    a. View name: **produce_qty_to_order**
    b. WHERE store_id = {} and current_case_qty < par_case_qty
6. For showing current paper quantity (if current less than par) of a particular store.
    a. View name: **paper_qty_to_order**
    b. WHERE store_id = {} and current_case_qty < par_case_qty
7. For showing current sss quantity (if current less than par) of a particular store.
    a. View name: **sss_qty_to_order**
    b. WHERE store_id = {} and current_case_qty < par_case_qty
8. For showing current other quantity (if current less than par) of a particular store.
    a. View name: **other_qty_to_order**
    b. WHERE store_id = {} and current_case_qty < par_case_qty
9. For showing current retail quantity (if current less than par) of a particular store.
    a. View name: **retail_qty_to_order**
    b. WHERE store_id = {} and current_case_qty < par_case_qty
10. For showing current cleaning quantity (if current less than par) of a particular store.
    a. View name: **cleaning_qty_to_order**
    b. WHERE store_id = {} and current_case_qty < par_case_qty

INCOMING INVENTORY REPORTS - These are used to create reports for a given store that shows the incoming inventory, their quantity, and cost for a particular product type.

1. For showing the incoming quantity of bread products for a given store.
    a. View name: **incoming_bread_products**
    b. WHERE store_id = {} and received_date IS NULL
2. For showing the incoming quantity of dairy products for a given store.
    a. View name: **incoming_dairy_products**
    b. WHERE store_id = {} and received_date IS NULL
3. For showing the incoming quantity of coffee products for a given store.
    a. View name: **incoming_coffee_products**
    b. WHERE store_id = {} and received_date IS NULL
4. For showing the incoming quantity of meat products for a given store.
    a. View name: **incoming_meat_products**
    b. WHERE store_id = {} and received_date IS NULL
5. For showing the incoming quantity of produce products for a given store.
    a. View name: **incoming_produce_products**
    b. WHERE store_id = {} and received_date IS NULL
6. For showing the incoming quantity of paper products for a given store.
    a. View name: **incoming_paper_products**
    b. WHERE store_id = {} and received_date IS NULL
7. For showing the incoming quantity of sss products for a given store.
    a. View name: **incoming_sss_products**
    b. WHERE store_id = {} and received_date IS NULL
8. For showing the incoming quantity of other products for a given store.
    a. View name: **incoming_other_products**
    b. WHERE store_id = {} and received_date IS NULL
9. For showing the incoming quantity of retail products for a given store.
    a. View name: **incoming_retail_products**
    b. WHERE store_id = {} and received_date IS NULL

ORDER VERIFICATION REPORTS - used to verify that products (for specific product type and store) were ordered in the correct quantity. It shows any product that has a current quantity + the incoming quantity not equaling the par quantity.

1. Shows bread products that were ordered incorrectly for a given store.
    a. View name: **incorrectly_ordered_bread_products**
    b. WHERE store_id = {} and case_qty + current_case_qty <> par_case_qty and received_date is null
2. Shows dairy products that were ordered incorrectly for a given store.
    a. View name: **incorrectly_ordered_dairy_products**
    b. WHERE store_id = {} and case_qty + current_case_qty <> par_case_qty and received_date is null
3. Shows coffee products that were ordered incorrectly for a given store.
    a. View name: **incorrectly_ordered_coffee_products**

b. WHERE store_id = {} and case_qty + current_case_qty <> par_case_qty and received_date is null
4. Shows meat products that were ordered incorrectly for a given store.
    a. View name: **incorrectly_ordered_meat_products**
    b. WHERE store_id = {} and case_qty + current_case_qty <> par_case_qty and received_date is null
5. Shows produce products that were ordered incorrectly for a given store.
    a. View name: **incorrectly_ordered_produce_products**
    b. WHERE store_id = {} and case_qty + current_case_qty <> par_case_qty and received_date is null
6. Shows paper products that were ordered incorrectly for a given store.
    a. View name: **incorrectly_ordered_paper_products**
    b. WHERE store_id = {} and case_qty + current_case_qty <> par_case_qty and received_date is null
7. Shows sss products that were ordered incorrectly for a given store.
    a. View name: **incorrectly_ordered_sss_products**
    b. WHERE store_id = {} and case_qty + current_case_qty <> par_case_qty and received_date is null
8. Shows other products that were ordered incorrectly for a given store.
    a. View name: **incorrectly_ordered_other_products**
    b. WHERE store_id = {} and case_qty + current_case_qty <> par_case_qty and received_date is null
9. Shows retail products that were ordered incorrectly for a given store.
    a. View name: **incorrectly_ordered_retail_products**
    b. WHERE store_id = {} and case_qty + current_case_qty <> par_case_qty and received_date is null
10. Shows cleaning products that were ordered incorrectly for a given store.
    a. View name: **incorrectly_ordered_cleaning_products**
    b. WHERE store_id = {} and case_qty + current_case_qty <> par_case_qty and received_date is null

FULL STORE INVENTORY REPORTS - used to show the complete current inventory of a given store. It shows product names and their quantities.

1. Shows full inventory for a given store.
    a. View name: **full_store_inventory_report**
    b. Where store_id = {}

ORDER PAYMENT VERIFICATION REPORTS - used to show whether a store has any incorrectly paid, or unpaid amounts for invoices.

1. Shows all orders for a given store, their cost, and the amounts paid if the amount due and the amount paid are not equal.
    a. View name: **order_payment_verification**
    b. Where store_id = {} and amt_paid <> amt_due

CATERING EVENT REPORTS - used to provide an overview of a customer order for a catering event.

1. Shows customer contact information for the given id. And when given their corresponding event_id, it shows the items in that catering event.
    a. View name: **catering_event_report**
    b. Where customer_id = {} or event_id = {}

FULL PRODUCT ORDER REPORTS - shows a full list of products for a given order.

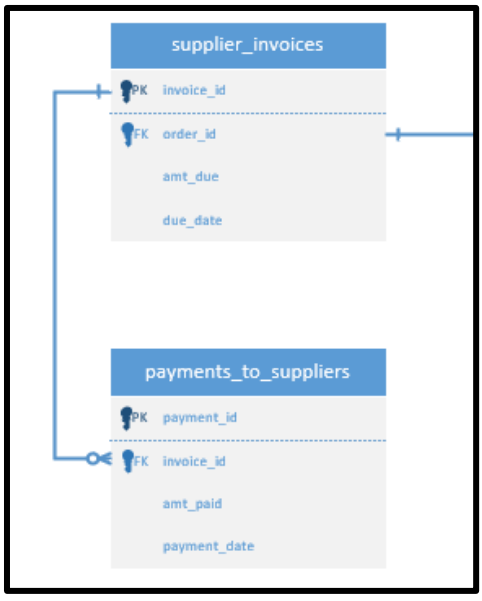
1. Shows all products names, their price per case, and their quantities for a given order.
    a. View name: **order_report**
    b. Where order_id = {}

Database Design Decisions

For our database design, we divided items into their own types. For example, we have tables specifically for meat products, coffee, and bread products. This makes the ERD more organized as well as the database.

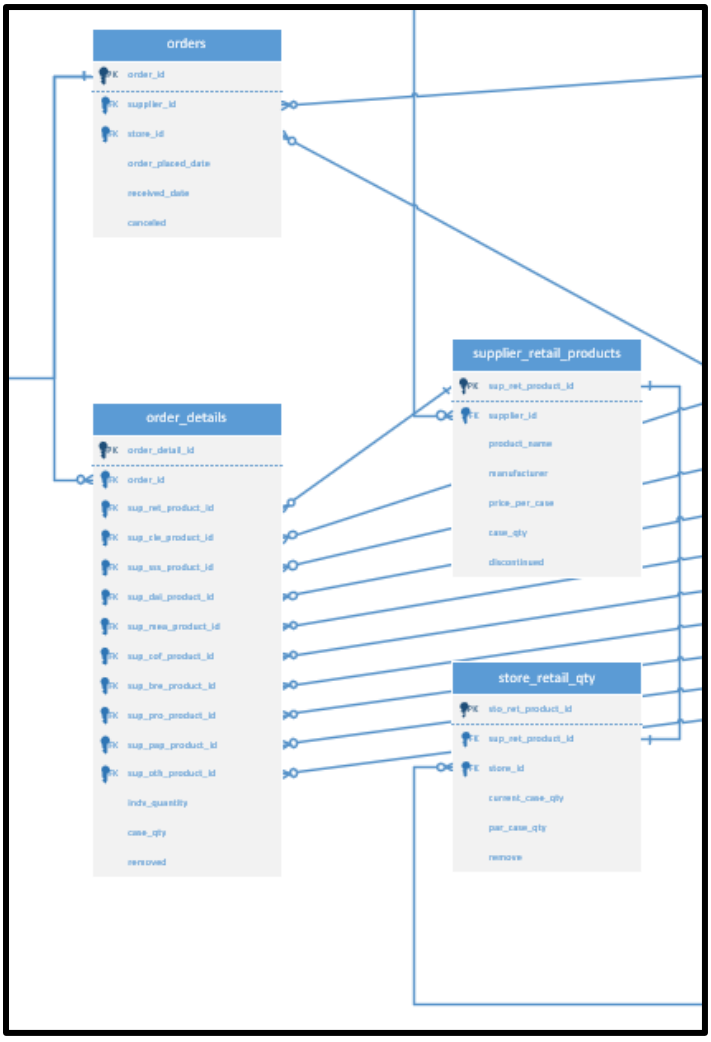
Section 1:

These two tables relate to the suppliers where it connects the invoices and the payments. This allows users to record payments for later analysis.



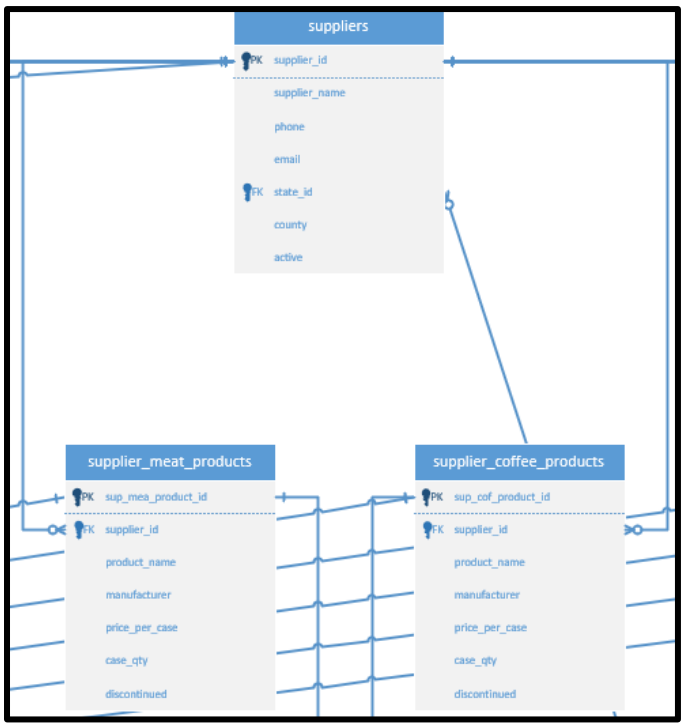
Section 2:

These tables involve the order and order details. It allows the other product tables to connect with the quantity tables. This allows for employees to keep count of their items and make sure they are organized.



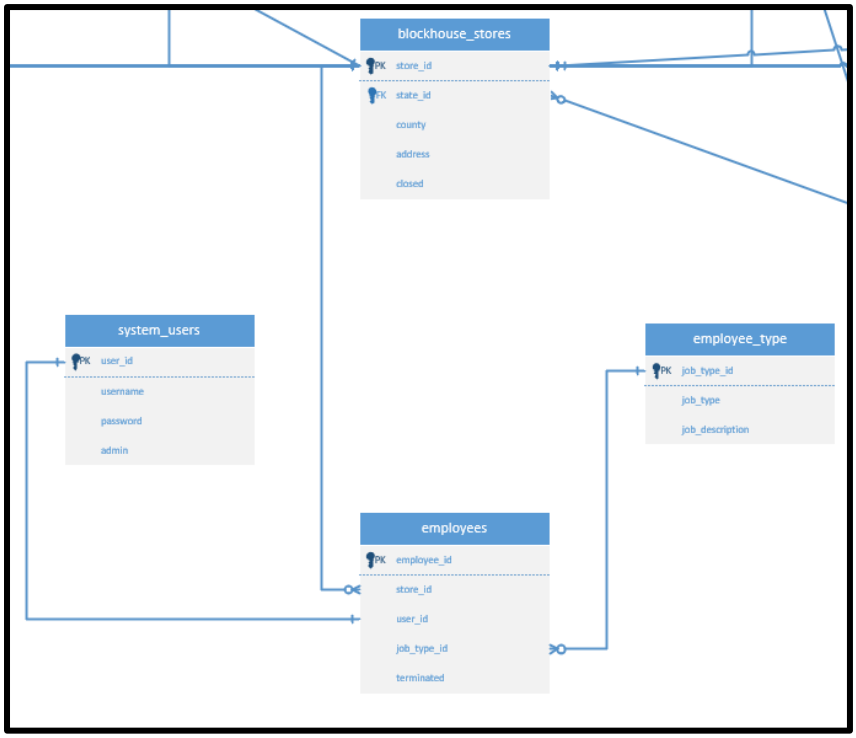
Section 3:

The supplier's tables connect with all of the product tables. This allows you to trace which supplier provided the products so you can have information for later use. Employees can verify that the products came from the correct supplier.



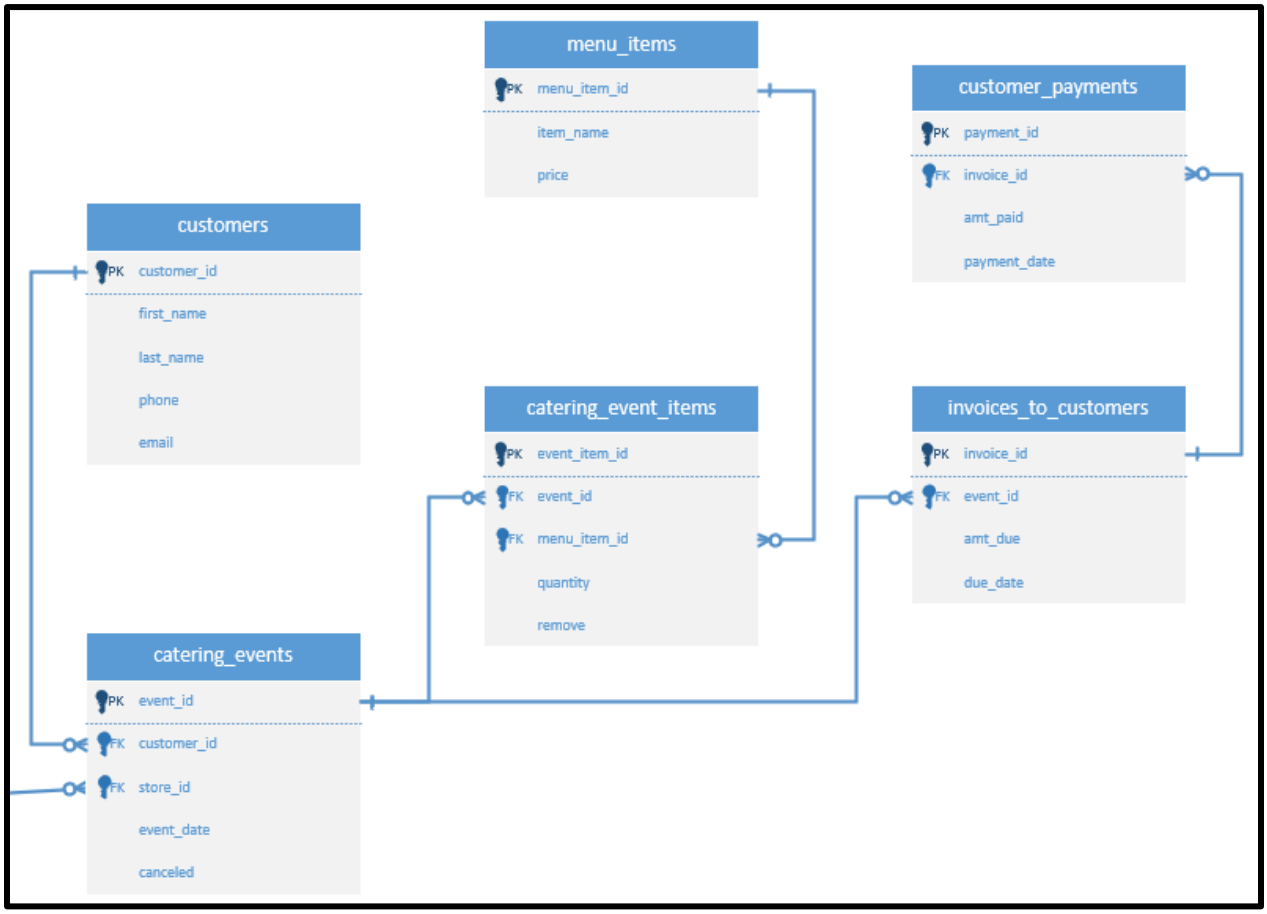
Section 4:

These tables connect employees and users to the store. Also, it allows employees to verify themselves so that they can have access to use the application.



Section 5:

These tables connect customer and catering tables. Blockhouse can keep track of customer information as well as payments. Also, they can organize catering event information in relation to customers.



Testing Process

The team's protocol for the testing usability of the database was done on Microsoft SQL Server Management Studio. In order for this to be done, each individual member had to install a VPN and set up a remote connection in order to log on to the database. Once completed, all members were able to connect to the database and revise any changes. Furthermore, the application was coded in Python and used Tkinter as the GUI toolkit. In order to write the code and test it, members were required to use their own respective IDE and share their code on Github, so others can share and review their program.

Project Improvements

There are a number of potential improvements that could be implemented in a later release. The top priority for an update would be to attempt to simplify the code for the graphical user interface. Currently there are over one thousand lines of code in the GUI script which has made altering and debugging the code far more difficult than expected. Another high priority update would be adding in a better method for archiving old database information to make room for new data. Over a long period of use the database can become bloated with data that no longer needs to be readily accessed. While this data is too important to completely remove. it may be possible to implement a method of automatically set data over a certain age as inactive. These inactive records could be moved to a separate storage location for archival, so they do not bloat the active database. The lowest priority update for the future would be the inclusion of automatically generated reports. The current system allows for the creation of weekly, monthly, and yearly data reports. It may be possible to automate report creation and have them sent to the store managers. Simplifying the creation of these reports could help the managers keep better track of their inventory.

Project Database Maintenance Issues

Data Cleanup

In order to cleanup data in the SQL database. You can delete data in the database by going to the object explorer and expanding your database. This allows you to select certain files and choose to delete them. Another tool we can use is the Maintenance Cleanup Task. This tool allows users to delete backup files and this can be done inside the SQL database. Doing these tasks allows the database to have more space.

Backup & Recovery

If your application goes into production, our database needs to be secure. Therefore, having a backup is a good way to ensure that our data is protected. This can be done by creating a backup within the database server. You can go into the SQL studio and right click the database you want to backup. After a couple of steps, you can choose a folder to store the backup and the process is finished.

Data Archival

In the event that the database gets too large with data. There should be a way to keep some of the important data from the past. A way to do this is to use the built-in partition wizard which allows you to split your large tables into smaller ones in order to run faster. This will lead into faster response times and you are able to load data for specific reasons. Another way to archive data is to mark certain records as inactive. This way the data is still inside the database, but certain operations will not check those records because they are inactive and will only check those that are still operational.

Lessons Learned

What worked?

For this project, what worked was the implementation of the database. Everyone was able to access the database and input tables and columns with their respective data. Furthermore, each member had a clear plan of what to do because as a team, we decided to divide the work into different groups. This organization allowed us to work efficiently and get the work done faster. Another part of the project that worked was the implementation of the application that we were supposed to make for this project. Our team members had a good outline on what it should look like and how it should be created. Overall, the major portions of the project were handled well.

What didn't work?

The parts of the project that did not work were the tables that we needed to come up with for the database. It seemed as though some of the tables we created were out of the project scope and did not make sense to put into the database. Because of this, we had to delete some of the tables and come up with new ones that make sense so that our project is less confusing. Ultimately, this slowed down the process of finishing our project, but was still able to complete it in time.

What we would do differently?

If we could do the project differently, it would be to make sure that the data dictionary is finalized before we proceed to the next steps of the project. This ensures that we know which items to put into our tables which would make the process of creating the database easier. Also, we would make sure the scope of our project is feasible such as the creation of the application.

Project Summary

In summarization of the project, last semester our team had decided to work with Blockhouse Coffee & Kitchen in hopes to improve their business operations. We had interviewed with Brian Alfaro, the operations manager, and discovered that the way the company tracks their inventory was inefficient. The company was beginning to be overwhelmed by the number of spreadsheets that were created keeping count of many items. Obviously this is a problem that could lead to other mistakes such as counting errors and loss of data, in which our team can step in and offer a solution.

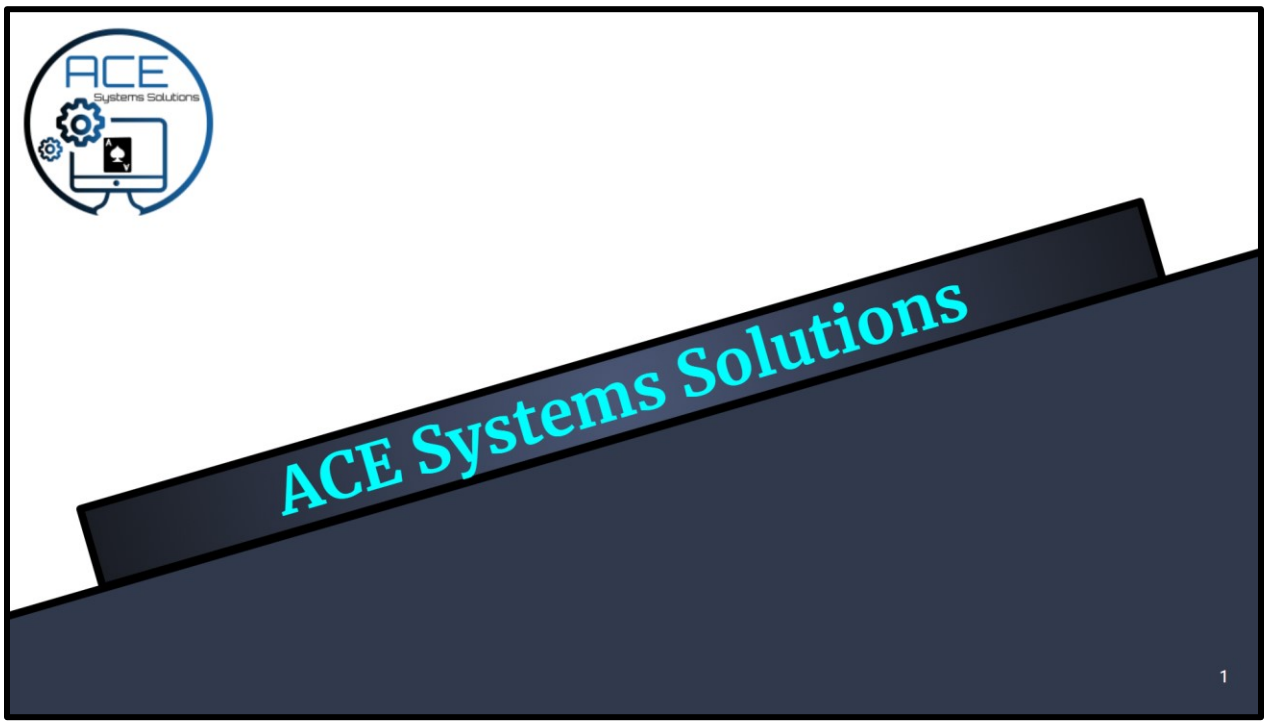
Afterwards, Brian had agreed to accept our help with the creation of a new system that can help keep track of business operations such as inventory. Based on the ideas that Brian gave us, it was in our best interest that we should create an inventory application. Blockhouse Coffee & Kitchen has more than one location whose business is continuing to expand. Since they are already successful, the addition of an application should not hurt their business, but instead give them more options on what to use. We figured that if they do not want to use our application that we create, then they can continue using their existing services. Our application allows users to input and modify items as they see fit and will be connected to a database. This allows the company to store a large amount of data in a single space and find them relatively fast.

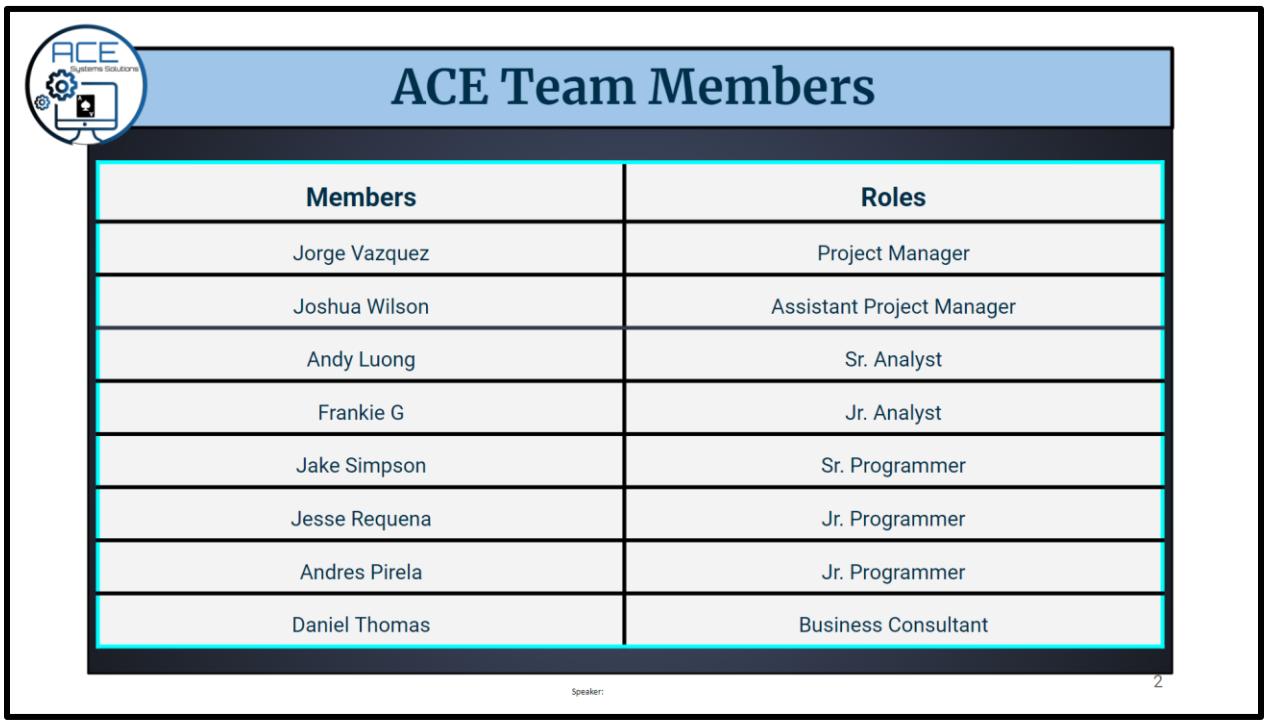
Finally, we believe that this application is going to benefit Blockhouse Coffee & Kitchen because it will increase efficiency and keep labor costs down. Less time will be spent doing inventory management which will result in more time being allocated elsewhere. Overall, this application can coordinate between both store locations and will be extremely beneficial for them to consider.

References

https://docs.microsoft.com/en-us/sql/relational-databases/databases/delete-data-or-log-files-from-a-database?view=sql-server-ver15

Presentation Slides



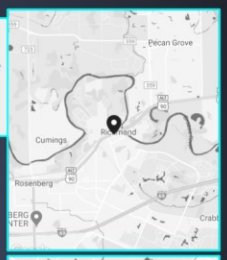


Press Esc to exit full screen

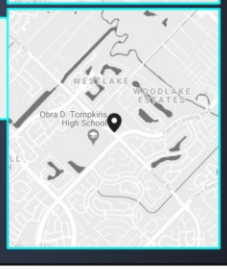
Client Organization

BLOCK HOUSE
COFFEE & KITCHEN

Richmond | Jax & 7th
611 Jackson Street Suite C, Richmond, TX 77469



Katy | Stableside
9910 Gaston Road, Suite 170, Katy, TX 77494



Speaker:

3

Press Esc to exit full screen

Problems & Requirements

Problems

- Currently they have no concise, efficient, effective, and accurate way of managing inventory
- Having individual sheets for every week and month is becoming increasingly difficult to manage on a google drive as the business ages
- Entering the counts a second time into a separate sheet to gauge par levels and place weekly orders is both time consuming and opens the possibility for more entry errors

Requirements

- System should have complete list of inventoried items
- Only upper management should have permission to edit the list of items (add/remove/update)
- Items should have the amount required to be on hand (Par levels)
- For each product category, the system should auto generate a list of items to order that are below the required par level
- System should be easy to understand for new users
- System should be an available option if the business decides to use it with or without their current system
- System should be finished by Spring 2021

Speaker:

4

Press Esc to exit full screen

System Objective

Create a SQL database hosted on Azure through which the client may store, search, and modify information as well as create useful reports through an intuitive and minimalistic GUI.

- Increase productivity
- Increase efficiency of operations
- Reduce overhead costs

Benefits the client by:

- Time spent storing and finding unorganized information is reduced 50%
- Errors in inventory counts and subsequent orders are reduced 50%

Results are measured through:

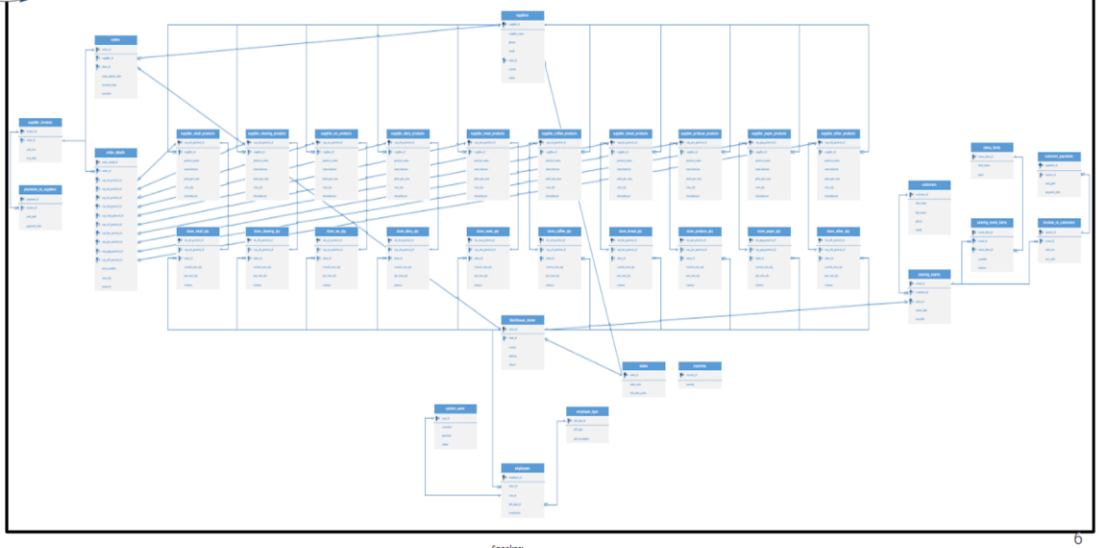
- A decrease in the time a manager or supervisor will spend searching through inventory records (1 hour) to create a list of items and their amounts to be ordered down to 30 minutes
- A decrease in the amount of overhead resulting from information errors from 10 to 20 per week down to 1 to 5 per week

Speaker:

5

Press Esc to exit full screen

ERD

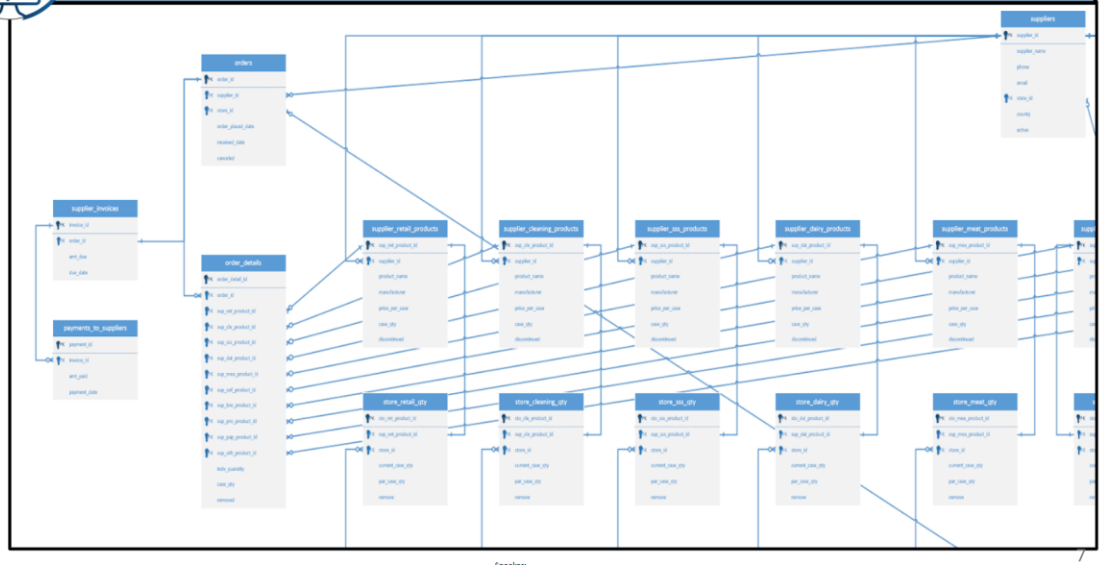


Speaker:

6

Press Esc to exit full screen

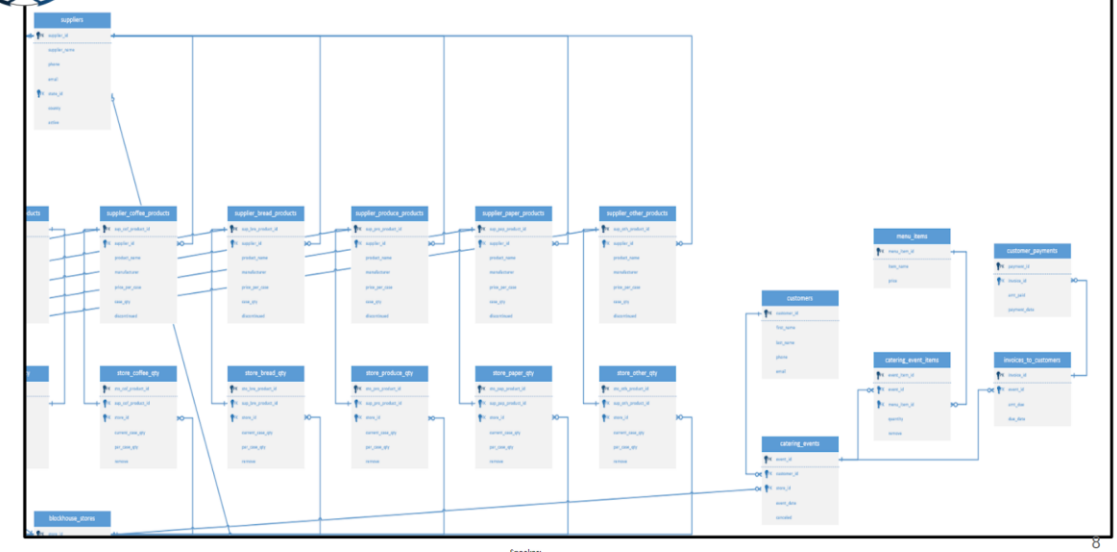
ERD - Left



Speaker:

Press Esc to exit full screen

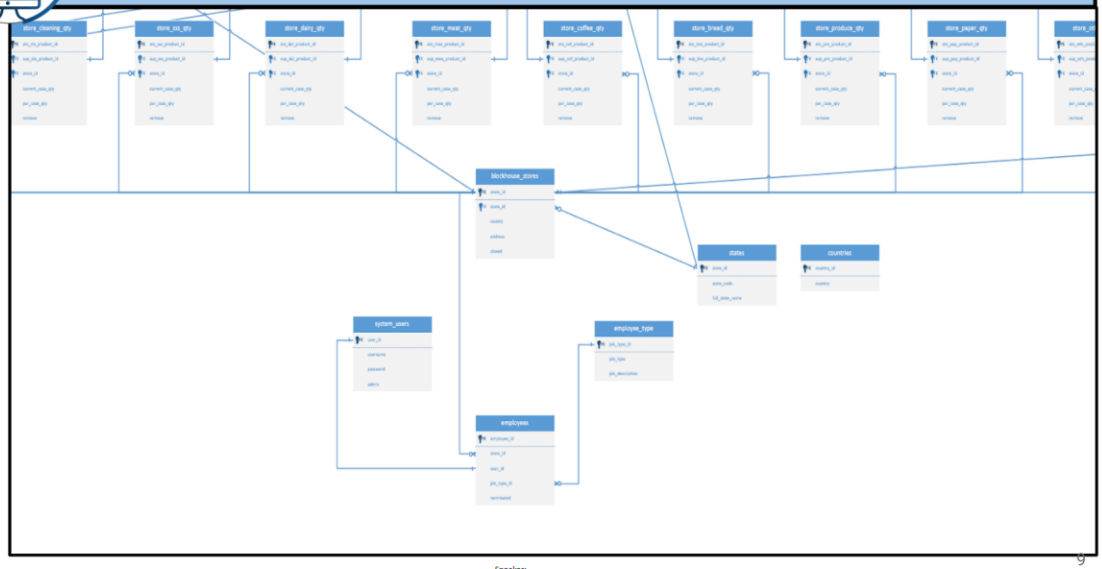
ERD - Right



Speaker:

Press Esc to exit full screen

ERD - Bottom



Speaker:

9

Press Esc to exit full screen

Application Demo

Speaker:

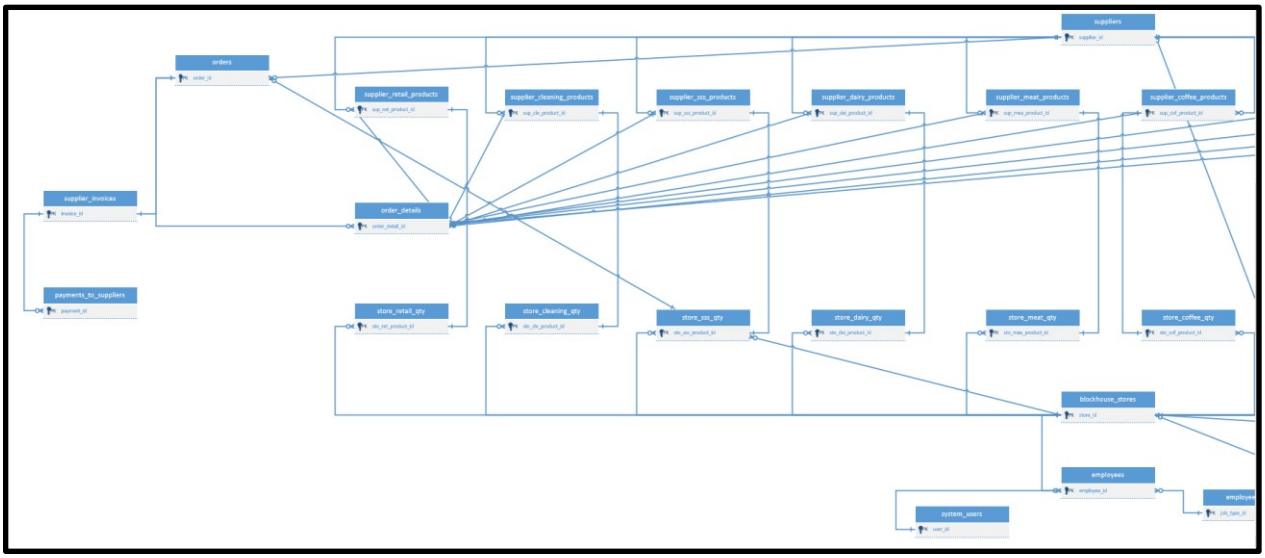
10


Press Esc to exit full screen
Conclusion
Thank you for your time!
Questions?
Speaker:
11

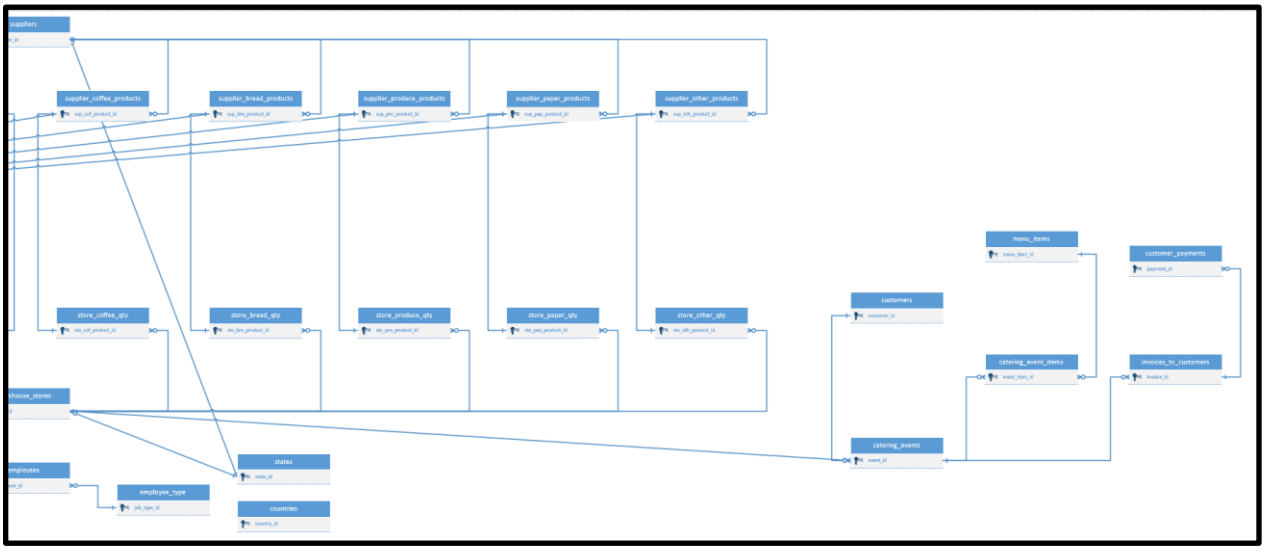
Appendices

ERD with primary keys and relationships only:

This image provides a view of the left side of the trimmed ERD

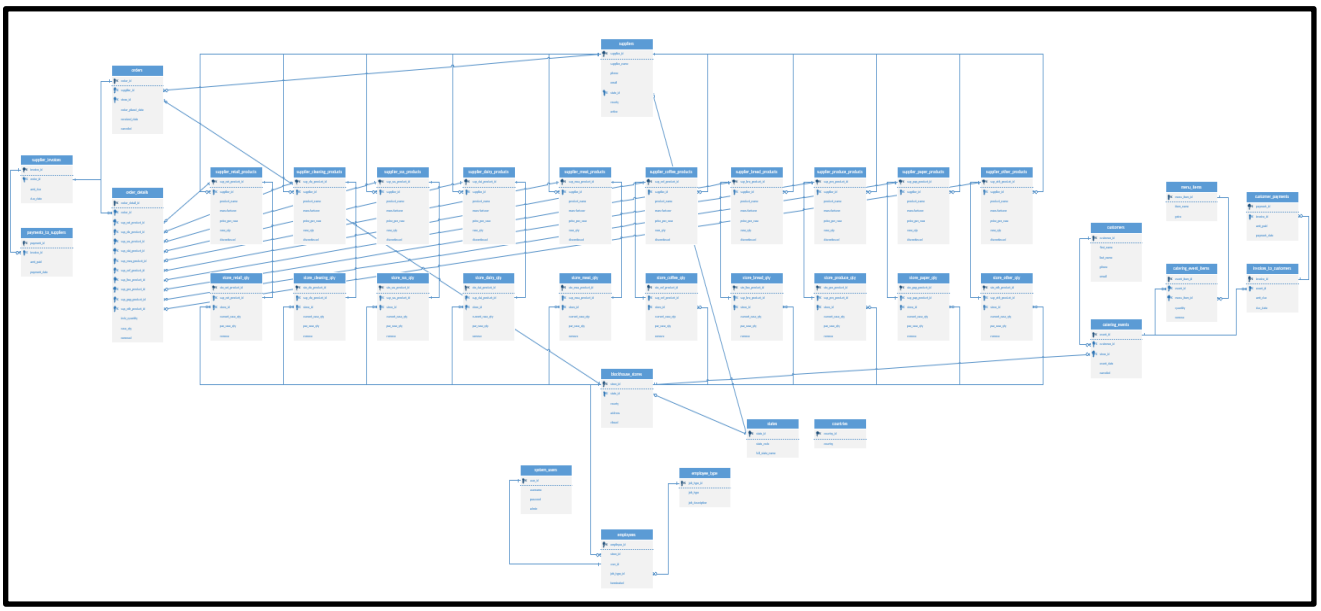


This image provides a view of the right side of the trimmed ERD

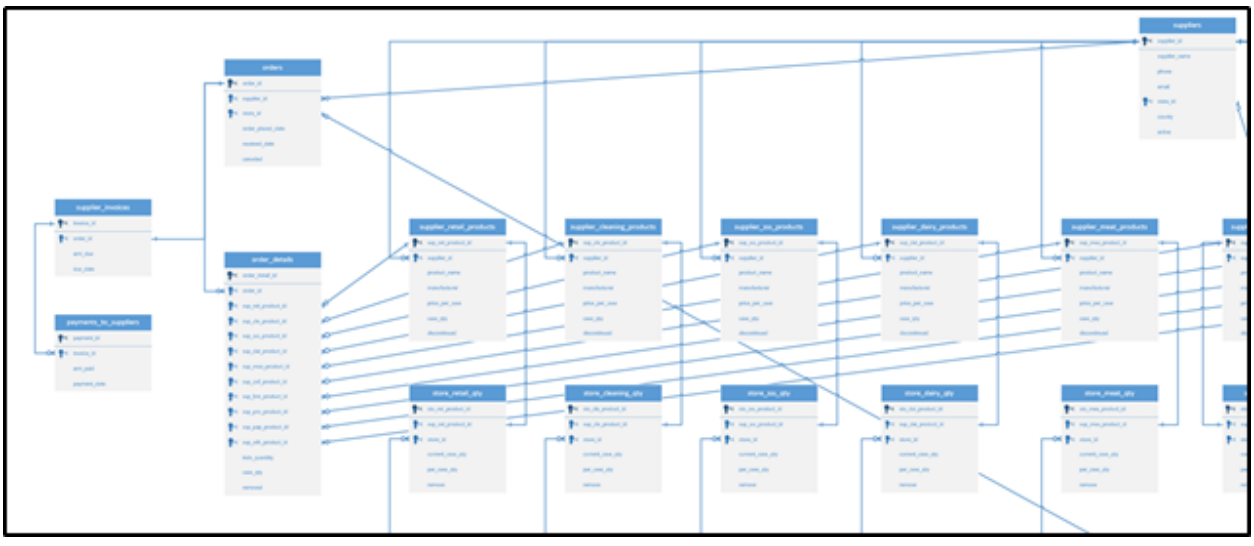


ERD with all attributes and relationships:

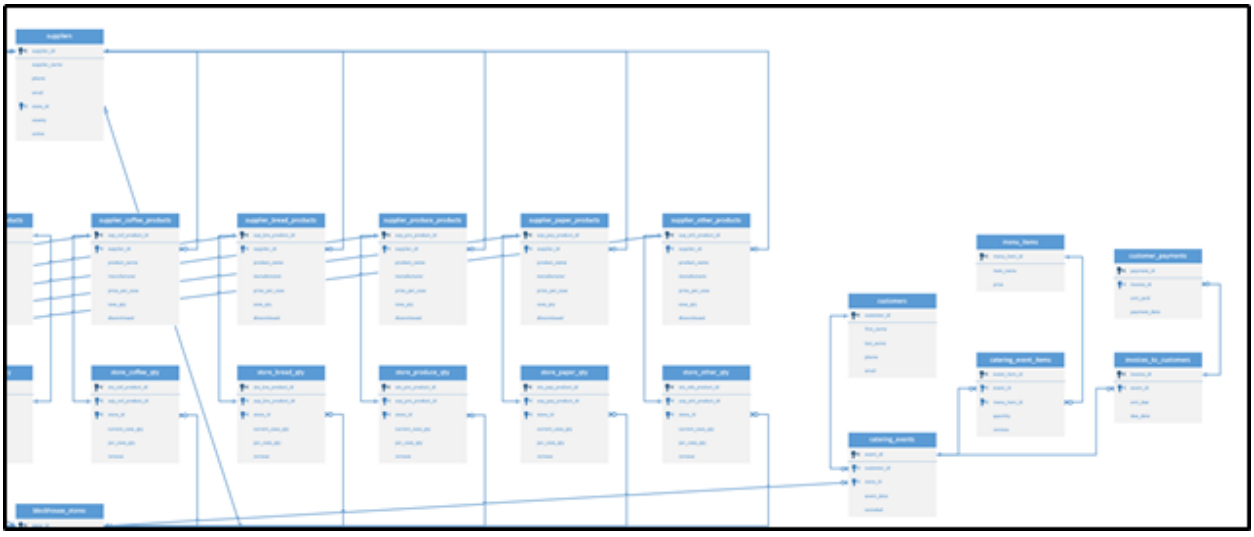
This image provides an overview of the entire ERD



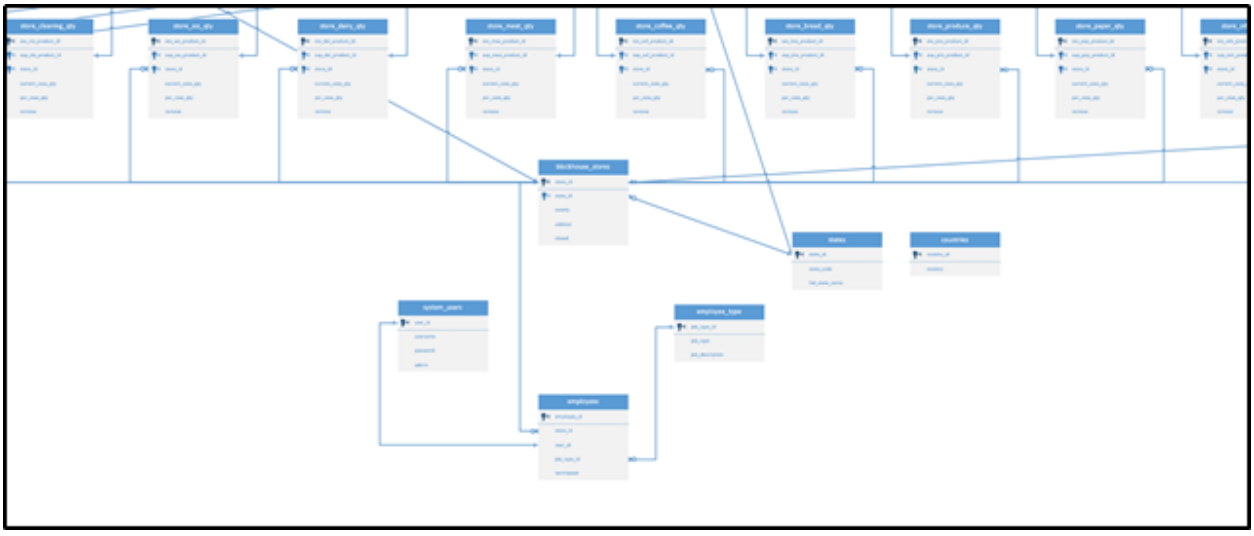
This is a view of the left side of the ERD so the details are more visible.



This is a view of the right side of the ERD so the details are more visible.



This is a view of the bottom of the ERD so the details are more visible.



Data dictionary:

TABLE: 1	blockhouse_stores				
column	data type	constraints	range?	required?	Cascades?
store_id (FK)	int, IDENTITY(1,1)	not nullable	0+	required	
state_id (FK)	int	not nullable	0+	required	none
city	int	not nullable	0+	required	
address	int	not nullable	0+	required	
closed?	bit	nullable	0-1	optional	

- **Description:** This table is used to store information about the Blockhouse Coffee & Kitchen locations. Each store in the table represents a different location. The purpose is to identify stores to which other related table attributes can be assigned.

TABLE: 2	store_dairy_qty				
column	data type	constraints	range?	required?	Cascades?
sto_dai_product_id (PK)	int, IDENTITY(1,1)	not nullable	0+	required	
sup_dai_product_id (FK)	int	not nullable	0+	required	none
store_id (FK)	int	not nullable	0+	required	none
current_case_qty	int	not nullable	0+	required	
par_case_qty	int	not nullable	0+	required	
remove?	bit	nullable	0-1	optional	

- **Description:** This table is used to store information about dairy products being carried at an individual Blockhouse Coffee & Kitchen location. The attributes represent individual products, their current quantities, and their par quantities. The purpose of the table is to allow a store to keep a live count of all their dairy products.

TABLE: 3	store_coffee_qty				
column	data type	constraints	range?	required?	Cascades?
sto_cof_product_id (PK)	int, IDENTITY(1,1)	not nullable	0+	required	

sup_cof_product_id (FK)	int	not nullable	0+	required	none
store_id (FK)	int	not nullable	0+	required	none
current_case_qty	int	not nullable	0+	required	
par_case_qty	int	not nullable	0+	required	
remove?	bit	nullable	0-1	optional	

- **Description:** This table is used to store information about coffee products being carried at an individual Blockhouse Coffee & Kitchen location. The attributes represent individual products, their current quantities, and their par quantities. The purpose of the table is to allow a store to keep a live count of all their coffee products.

TABLE: 4	**store_meat_qty**				
column	data type	constraints	range?	required?	Cascades?
sto_mea_product_id (PK)	int, IDENTITY(1,1)	not nullable	0+	required	
sup_mea_product_id (FK)	int	not nullable	0+	required	none
store_id (FK)	int	not nullable	0+	required	none
current_case_qty	int	not nullable	0+	required	
par_case_qty	int	not nullable	0+	required	
remove?	bit	nullable	0-1	optional	

- **Description:** This table is used to store information about meat products being carried at an individual Blockhouse Coffee & Kitchen location. The attributes represent individual products, their current quantities, and their par quantities. The purpose of the table is to allow a store to keep a live count of all their meat products.

TABLE: 5	**store_bread_qty**				
column	data type	constraints	range?	required?	Cascades?
sto_bre_product_id (PK)	int, IDENTITY(1,1)	not nullable	0+	required	
sup_bre_product_id (FK)	int	not nullable	0+	required	none
store_id (FK)	int	not nullable	0+	required	none
current_case_qty	int	not nullable	0+	required	
par_case_qty	int	not nullable	0+	required	
remove?	bit	nullable	0-1	optional	

- **Description:** This table is used to store information about bread products being carried at an individual Blockhouse Coffee & Kitchen location. The attributes represent individual products, their current quantities, and their par quantities. The purpose of the table is to allow a store to keep a live count of all their bread products.

TABLE: 6	store_produce_qty				
column	data type	constraints	range?	required?	Cascades?
sto_pro_product_id (PK)	int, IDENTITY(1,1)	not nullable	0+	required	
sup_pro_product_id (FK)	int	not nullable	0+	required	none
store_id (FK)	int	not nullable	0+	required	none
current_case_qty	int	not nullable	0+	required	
par_case_qty	int	not nullable	0+	required	
remove?	bit	nullable	0-1	optional	

- **Description:** This table is used to store information about produce products being carried at an individual Blockhouse Coffee & Kitchen location. The attributes represent individual products, their current quantities, and their par quantities. The purpose of the table is to allow a store to keep a live count of all their produce products.

TABLE: 7	store_paper_qty				
column	data type	constraints	range?	required?	Cascades?
sto_pap_product_id (PK)	int, IDENTITY(1,1)	not nullable	0+	required	
sup_pap_product_id (FK)	int	not nullable	0+	required	none
store_id (FK)	int	not nullable	0+	required	none
current_case_qty	int	not nullable	0+	required	
par_case_qty	int	not nullable	0+	required	
remove?	bit	nullable	0-1	optional	

- **Description:** This table is used to store information about paper products being carried at an individual Blockhouse Coffee & Kitchen location. The attributes represent individual products,

their current quantities, and their par quantities. The purpose of the table is to allow a store to keep a live count of all their paper products.

TABLE: 8	store_sss_qty				
column	data type	constraints	range?	required?	Cascades?
sto_sss_product_id (PK)	int, IDENTITY(1,1)	not nullable	0+	required	
sup_sss_product_id (FK)	int	not nullable	0+	required	none
store_id (FK)	int	not nullable	0+	required	none
current_case_qty	int	not nullable	0+	required	
par_case_qty	int	not nullable	0+	required	
remove?	bit	nullable	0-1	optional	

- **Description:** This table is used to store information about spices and seasoning products being carried at an individual Blockhouse Coffee & Kitchen location. The attributes represent individual products, their current quantities, and their par quantities. The purpose of the table is to allow a store to keep a live count of all their spices and seasoning products.

TABLE: 9	store_other_qty				
column	data type	constraints	range?	required?	Cascades?
sto_oth_product_id (PK)	int, IDENTITY(1,1)	not nullable	0+	required	
sup_oth_product_id (FK)	int	not nullable	0+	required	none
store_id (FK)	int	not nullable	0+	required	none
current_case_qty	int	not nullable	0+	required	
par_case_qty	int	not nullable	0+	required	
remove?	bit	nullable	0-1	optional	

- **Description:** This table is used to store information about other (not belonging to another category) products being carried at an individual Blockhouse Coffee & Kitchen location. The attributes represent individual products, their current quantities, and their par quantities. The purpose of the table is to allow a store to keep a live count of all their other products.

TABLE: 10	store_retail_qty				
column	data type	constraints	range?	required?	Cascades?
sto_ret_product_id (PK)	int, IDENTITY(1,1)	not nullable	0+	required	
sup_ret_product_id (FK)	int	not nullable	0+	required	none
store_id (FK)	int	not nullable	0+	required	none
current_case_qty	int	not nullable	0+	required	
par_case_qty	int	not nullable	0+	required	
remove?	bit	nullable	0-1	optional	

- **Description:** This table is used to store information about retail products being carried at an individual Blockhouse Coffee & Kitchen location. The attributes represent individual products, their current quantities, and their par quantities. The purpose of the table is to allow a store to keep a live count of all their retail products.

TABLE: 11	store_cleaning_qty				
column	data type	constraints	range?	required?	Cascades?
sto_cle_product_id (PK)	int, IDENTITY(1,1)	not nullable	0+	required	
sup_cle_product_id (FK)	int	not nullable	0+	required	none
store_id (FK)	int	not nullable	0+	required	none
current_case_qty	int	not nullable	0+	required	
par_case_qty	int	not nullable	0+	required	
remove?	bit	nullable	0-1	optional	

- **Description:** This table is used to store information about cleaning products being carried at an individual Blockhouse Coffee & Kitchen location. The attributes represent individual products, their current quantities, and their par quantities. The purpose of the table is to allow a store to keep a live count of all their cleaning products.

TABLE: 12	suppliers				
column	data type	constraints	range?	required?	Cascades?
supplier_id (PK)	int, IDENTITY(1,1)	not nullable	0+	required	
supplier_name	varchar(255)	not nullable	0+	required	

phone	varchar(12)	not nullable	0+	required	
email	varchar(255)	not nullable	0+	required	
state_id (FK)	int, IDENTITY(1,1)	not nullable	0+	required	none
active?	bit	nullable	0-1	optional	

- **Description:** This table is used to store information about different suppliers of products. The purpose of this table is to allow different products to be assigned to a supplier.

TABLE: 13	**supplier_dairy_products**				
column	data type	constraints	range?	required?	Cascades?
sup_dai_product_id (PK)	int, IDENTITY(1,1)	not nullable	0+	required	
supplier_id (FK)	int	not nullable	0+	required	none
product_name	int	not nullable	0+	required	
manufacturer	int	not nullable	0+	required	
price_per_case	int	not nullable	0+	required	
case_qty	int	not nullable	0+	required	
discontinued	bit	nullable	0-1	optional	

- **Description:** This table is used to store information about different dairy products offered by different suppliers. The attributes represent one product, which belongs to one supplier, that has a name / manufacturer, a price, and a case quantity. The purpose of this table is to keep track of different dairy products offered by suppliers.

TABLE: 14	**supplier_coffee_products**				
column	data type	constraints	range?	required?	Cascades?
sup_cof_product_id (PK)	int, IDENTITY(1,1)	not nullable	0+	required	
supplier_id (FK)	int	not nullable	0+	required	none
product_name	int	not nullable	0+	required	
manufacturer	int	not nullable	0+	required	
price_per_case	int	not nullable	0+	required	
case_qty	int	not nullable	0+	required	
discontinued	bit	nullable	0-1	optional	

- **Description:** This table is used to store information about different coffee products offered by different suppliers. The attributes represent one product, which belongs to one supplier, that has a name / manufacturer, a price, and a case quantity. The purpose of this table is to keep track of different coffee products offered by suppliers.

TABLE: 15	supplier_meat_products				
column	data type	constraints	range?	required?	Cascades?
sup_mea_product_id (PK)	int, IDENTITY(1,1)	not nullable	0+	required	
supplier_id (FK)	int	not nullable	0+	required	none
product_name	int	not nullable	0+	required	
manufacturer	int	not nullable	0+	required	
price_per_case	int	not nullable	0+	required	
case_qty	int	not nullable	0+	required	
discontinued	bit	nullable	0-1	optional	

- **Description:** This table is used to store information about different meat products offered by different suppliers. The attributes represent one product, which belongs to one supplier, that has a name / manufacturer, a price, and a case quantity. The purpose of this table is to keep track of different meat products offered by suppliers.

TABLE: 16	supplier_bread_products				
column	data type	constraints	range?	required?	Cascades?
sup_bre_product_id (PK)	int, IDENTITY(1,1)	not nullable	0+	required	
supplier_id (FK)	int	not nullable	0+	required	none
product_name	int	not nullable	0+	required	
manufacturer	int	not nullable	0+	required	
price_per_case	int	not nullable	0+	required	
case_qty	int	not nullable	0+	required	
discontinued	bit	nullable	0-1	optional	

- **Description:** This table is used to store information about different bread products offered by different suppliers. The attributes represent one product, which belongs to one supplier, that

has a name / manufacturer, a price, and a case quantity. The purpose of this table is to keep track of different bread products offered by suppliers.

TABLE: 17	supplier_produce_products				
column	data type	constraints	range?	required?	Cascades?
sup_pro_product_id (PK)	int, IDENTITY(1,1)	not nullable	0+	required	
supplier_id (FK)	int	not nullable	0+	required	none
product_name	int	not nullable	0+	required	
manufacturer	int	not nullable	0+	required	
price_per_case	int	not nullable	0+	required	
case_qty	int	not nullable	0+	required	
discontinued	bit	nullable	0-1	optional	

- **Description:** This table is used to store information about different produce products offered by different suppliers. The attributes represent one product, which belongs to one supplier, that has a name / manufacturer, a price, and a case quantity. The purpose of this table is to keep track of different produce products offered by suppliers.

TABLE: 18	supplier_paper_products				
column	data type	constraints	range?	required?	Cascades?
sup_pap_product_id (PK)	int, IDENTITY(1,1)	not nullable	0+	required	
supplier_id (FK)	int	not nullable	0+	required	none
product_name	int	not nullable	0+	required	
manufacturer	int	not nullable	0+	required	
price_per_case	int	not nullable	0+	required	
case_qty	int	not nullable	0+	required	
discontinued	bit	nullable	0-1	optional	

- **Description:** This table is used to store information about different paper products offered by different suppliers. The attributes represent one product, which belongs to one supplier, that has a name / manufacturer, a price, and a case quantity. The purpose of this table is to keep track of different paper products offered by suppliers.

TABLE: 19	supplier_sss_products				
column	data type	constraints	range?	required?	Cascades?
sup_sss_product_id (PK)	int, IDENTITY(1,1)	not nullable	0+	required	
supplier_id (FK)	int	not nullable	0+	required	none
product_name	int	not nullable	0+	required	
manufacturer	int	not nullable	0+	required	
price_per_case	int	not nullable	0+	required	
case_qty	int	not nullable	0+	required	
discontinued	bit	nullable	0-1	optional	

- **Description:** This table is used to store information about different spices and seasoning products offered by different suppliers. The attributes represent one product, which belongs to one supplier, that has a name / manufacturer, a price, and a case quantity. The purpose of this table is to keep track of different spices and seasoning products offered by suppliers.

TABLE: 20	supplier_other_products				
column	data type	constraints	range?	required?	Cascades?
sup_oth_product_id (PK)	int, IDENTITY(1,1)	not nullable	0+	required	
supplier_id (FK)	int	not nullable	0+	required	none
product_name	int	not nullable	0+	required	
manufacturer	int	not nullable	0+	required	
price_per_case	int	not nullable	0+	required	
case_qty	int	not nullable	0+	required	
discontinued	bit	nullable	0-1	optional	

- **Description:** This table is used to store information about different other (not belonging to another category) products offered by different suppliers. The attributes represent one product, which belongs to one supplier, that has a name / manufacturer, a price, and a case quantity. The purpose of this table is to keep track of different other products offered by suppliers.

TABLE: 21	supplier_retail_products				
column	data type	constraints	range?	required?	Cascades?
sup_ret_product_id (PK)	int, IDENTITY(1,1)	not nullable	0+	required	

supplier_id (FK)	int	not nullable	0+	required	none
product_name	int	not nullable	0+	required	
manufacturer	int	not nullable	0+	required	
price_per_case	int	not nullable	0+	required	
case_qty	int	not nullable	0+	required	
discontinued	bit	nullable	0-1	optional	

- **Description:** This table is used to store information about different retail products offered by different suppliers. The attributes represent one product, which belongs to one supplier, that has a name / manufacturer, a price, and a case quantity. The purpose of this table is to keep track of different retail products offered by suppliers.

TABLE: 22	**supplier_cleaning_products**				
column	data type	constraints	range?	required?	Cascades?
sup_cle_product_id (PK)	int, IDENTITY(1,1)	not nullable	0+	required	
supplier_id (FK)	int	not nullable	0+	required	none
product_name	int	not nullable	0+	required	
manufacturer	int	not nullable	0+	required	
price_per_case	int	not nullable	0+	required	
case_qty	int	not nullable	0+	required	
discontinued	bit	nullable	0-1	optional	

- **Description:** This table is used to store information about different cleaning products offered by different suppliers. The attributes represent one product, which belongs to one supplier, that has a name / manufacturer, a price, and a case quantity. The purpose of this table is to keep track of different cleaning products offered by suppliers.

TABLE: 23	**orders**				
column	data type	constraints	range?	required?	Cascades?
order_id (PK)	int, IDENTITY(1,1)	not nullable	0+	required	
supplier_id (FK)	int	not nullable	0+	required	none
store_id (FK)	int	not nullable	0+	required	none

order_placed_date	date	not nullable	N/A	required	
received_date	date	nullable	N/A	required	
canceled?	bit	nullable	0-1	optional	

- **Description:** This table is used to store information about different orders of products from suppliers. The attributes represent one order, belonging to a store, coming from a supplier, and whether it has been received. This purpose of this table is to allow a store to keep track of their orders of incoming and received inventory.

TABLE: 24	**order_details**				
column	data type	constraints	range?	required?	Cascades?
order_detail_id (PK)	int, IDENTITY(1,1)	not nullable	0+	required	
order_id (FK)	int	not nullable	0+	required	none
sup_dai_product_id (FK)	int	nullable	0+	optional	none
sup_cof_product_id (FK)	int	nullable	0+	optional	none
sup_mea_product_id (FK)	int	nullable	0+	optional	none
sup_bre_product_id (FK)	int	nullable	0+	optional	none
sup_pro_product_id (FK)	int	nullable	0+	optional	none
sup_pap_product_id (FK)	int	nullable	0+	optional	none
sup_sss_product_id (FK)	int	nullable	0+	optional	none
sup_oth_product_id (FK)	int	nullable	0+	optional	none
sup_ret_product_id (FK)	int	nullable	0+	optional	none
sup_cle_product_id (FK)	int	nullable	0+	optional	none
case_qty	int	not nullable	0+	required	
removed?	bit	nullable	0-1	optional	

- **Description:** This table is used to store information about what items are in a particular order. Each row represents one item belonging to an order with a quantity. Only one of the product FKs is allowed to be not null at any one time. This allows each row to correspond to only one product. The purpose of this table is to allow a store to construct an order and view the items in it.

TABLE: 25	**supplier_invoices**				

column	data type	constraints	range?	required?	Cascades?
invoice_id (PK)	int, IDENTITY(1,1)	not nullable	0+	required	
order_id (FK)	int	not nullable	0+	required	none
amt_due	float	not nullable	0+	required	
due_date	date	not nullable	N/A	required	

- **Description:** This table stores information about the costs of an order. Each row represents the cost of one order. The purpose of this table is to allow a store to keep track of their inventory costs.

TABLE: 26	**payments_to_suppliers**				
column	data type	constraints	range?	required?	Cascades?
payment_id (PK)	int, IDENTITY(1,1)	not nullable	0+	required	
invoice_id (FK)	int	not nullable	0+	required	none
amt_paid	float	not nullable	0+	required	
payment_date	date	not nullable	N/A	required	

- **Description:** This table stores information about the payments for orders. Each row represents the payment for an order. The purpose of this table is to allow a store to keep a record of all payments for orders.

TABLE: 27	**customers**				
column	data type	constraints	range?	required?	Cascades?
customer_id (PK)	int, IDENTITY(1,1)	not nullable	0+	required	
first_name	varchar(255)	not nullable	N/A	required	
last_name	varchar(255)	not nullable	N/A	required	
phone	varchar(12)	not nullable	N/A	required	
email	varchar(255)	not nullable	N/A	required	

- **Description:** This table stores information about customers. Each row represents one customer and their contact information. The purpose of this table is to allow a store to assign catering events and items to a customer.

TABLE: 28	catering_events				
column	data type	constraints	range?	required?	Cascades?
event_id (PK)	int, IDENTITY(1,1)	not nullable	0+	required	
customer_id (FK)	int	not nullable	0+	required	none
store_id (FK)	int	not nullable	0+	required	none
event_date	date	not nullable	N/A	required	
canceled	bit	nullable	0-1	optional	

- **Description:** This table stores information about a catering event. Each row represents one catering event, the store catering the event, the customer who scheduled it, and the date of the event. The purpose of this table is to create events to which items can be assigned

TABLE: 29	menu_items				
column	data type	constraints	range?	required?	Cascades?
menu_item_id (PK)	int, IDENTITY(1,1)	not nullable	0+	required	
item_name	varchar(255)	not nullable	N/A	required	
price	float	not nullable	0+	required	

- **Description:** This table stores information about items on the menu of Blockhouse Coffee & Kitchen locations. Each row represents one item and its price. The purpose of this table is to allow menu items to be assigned to an event

TABLE: 30	catering_event_items				
column	data type	constraints	range?	required?	Cascades?
event_item_id (PK)	int, IDENTITY(1,1)	not nullable	0+	required	
event_id (FK)	int	not nullable	0+	required	none
menu_item_id (FK)	int	not nullable	0+	required	none
quantity	int	not nullable	0+	required	
remove	bit	nullable	0-1	optional	

- **Description:** This table stores information about event items. Each row represents one item, belonging to an event, and the quantity that was requested. This purpose of this table is to allow a store to keep track of what items were ordered for a catering event.

TABLE: 31	invoices_to_customers				
column	data type	constraints	range?	required?	Cascades?
invoice_id (PK)	int, IDENTITY(1,1)	not nullable	0+	required	
event_id (FK)	int	not nullable	0+	required	none
amt_due	float	not nullable	0+	required	
due_date	date	not nullable	N/A	required	

- **Description:** This table stores information about the cost of events being charged to customers. Each row represents one invoice to a customer, the amount due, and the date due. The purpose of this table is to allow a store to see if a catering event has been paid for.

TABLE: 32	customer_payments				
column	data type	constraints	range?	required?	Cascades?
payment_id (PK)	int, IDENTITY(1,1)	not nullable	0+	required	
invoice_id (FK)	int	not nullable	0+	required	none
amt_paid	float	not nullable	0+	required	
payment_date	date	not nullable	N/A	required	

- **Description:** This table stores information about the payments for a catering event. Each row represents a payment for a catering event. The purpose of this table is to allow a store to see if a catering event has been paid for.

TABLE: 33	employees				
column	data type	constraints	range?	required?	Cascades?
employee_id (PK)	int, IDENTITY(1,1)	not nullable	0+	required	
first_name	varchar(255)	Not nullable	N/A	required	
last_name	varchar(255)	Not nullable	N/A	required	
store_id (FK)	int	not nullable	0+	required	none
user_id (FK)	int	not nullable	0+	required	none

job_type_id (FK)	int	not nullable	0+	required	none
terminated?	bit	nullable	0-1	optional	

- **Description:** This table stores information about Blockhouse Coffee & Kitchen employees. Each row represents one employee, the store they belong to, their system id, and their job. The purpose of this table is to allow a store to assign an employee to a store and keep track of their system credentials.

TABLE: 34	**system_users**				
column	data type	constraints	range?	required?	Cascades?
user_id (PK)	int, IDENTITY(1,1)	not nullable	0+	required	
username	varchar(255)	not nullable	0+	required	
password	varchar(255)	not nullable	0+	required	
admin?	bit	not nullable	0-1	required	

- **Description:** This table stores information about the user information for an employee. Each row represents one user and their credentials. The purpose of this table is to assign credentials to an employee.

TABLE: 35	**employee_type**				
column	data type	constraints	range?	required?	Cascades?
job_type_id (PK)	int, IDENTITY(1,1)	not nullable	0+	required	
job_type	varchar(255)	not nullable	N/A	required	
job_description	varchar(255)	not nullable	N/A	required	

- **Description:** This table stores information about the different jobs at Blockhouse Coffee & Kitchen. Each row represents one job. The purpose of this table is to assign a job to an employee.

TABLE: 36	**states**				
column	data type	constraints	range?	required?	Cascades?
state_id (PK)	int, IDENTITY(1,1)	not nullable	0+	required	
state_code	varchar(2)	not nullable	N/A	required	
full_state_name	varchar(255)	not nullable	N/A	required	

- **Description:** This table stores information about US states. Each row represents one state. This table is required by the project guidelines.

TABLE: 37	countries				
column	data type	constraints	range?	required?	Cascades?
country_id (PK)	int, IDENTITY(1,1)	not nullable	0+	required	
country	varchar(255)	not nullable	N/A	required	

- **Description:** This table stores information about countries. Each row represents one country. This table is required by the project guidelines.

Final Updated Problems and Requirements Lists:

Problems

- Currently they have no concise, efficient, effective, and accurate way of managing inventory
- Having individual sheets for every week and month is becoming increasingly difficult to manage on a google drive as the business ages
- Entering the counts a second time into a separate sheet to gauge par levels and place weekly orders is both time consuming and opens the possibility for more entry errors

Requirements

- System should have complete list of inventoried items
- Only upper management should have permission to edit the list of items (add/remove/update)
- Items should have the amount required to be on hand (Par levels)
- For each product category, the system should auto generate a list of items to order that are below the required par level
- System should be easy to understand for new users
- System should be an available option if the business decides to use it with or without their current system
- System should be finished by Spring 2021

Copy of Status Reports:

04/01/2021

This week the ACE Systems solutions team has been working on the three main parts of the project. We aim to have all parts completed by April 15th so that we can make changes as necessary. To do this, our group (8 people) has been split into 3 sub-groups. They are as follows:

Database: We have currently remade our database to fit the business requirements (mainly for inventory) of our client organization. The database has 37 tables and all data has been loaded. We have not yet updated our ERD to reflect these changes. Our next step is to update the ERD and develop new queries.

- Joshua R. Wilson
- Frankie M. Gomez
- Jorge Vasquez
- Dan Thomas (floater)

GUI: We are currently learning and working with Python Tkinter to develop a working prototype for our interface so that it can later be integrated with the database in the coming week(s).

- Jake Simpson
- Jesse Requena
- Andres Pirela
- Andy Luong (floater)

Final Report (float team): We are currently working on the report required for the project and assisting the other sub-groups where necessary.

- Andy Luong (floater)
- Dan Thomas (floater)

Please copy and paste the latest version/screenshot of your projects Entity Relational Diagram (ERD):

Our ERD has not yet been remade to reflect our new database design. Our old ERD is useless and is therefore not provided.

04/08/2021

This week the ACE Systems solutions team has been working on the three main parts of the project. We aim to have all parts completed by April 15th so that we can make changes as necessary. To do this, our group (8 people) has been split into 3 sub-groups. They are as follows:

Database: Our database is complete, our ERD has been remade, and we have completed all of our queries.

- Joshua R. Wilson
- Frankie M. Gomez
- Jorge Vasquez
- Dan Thomas (floater)

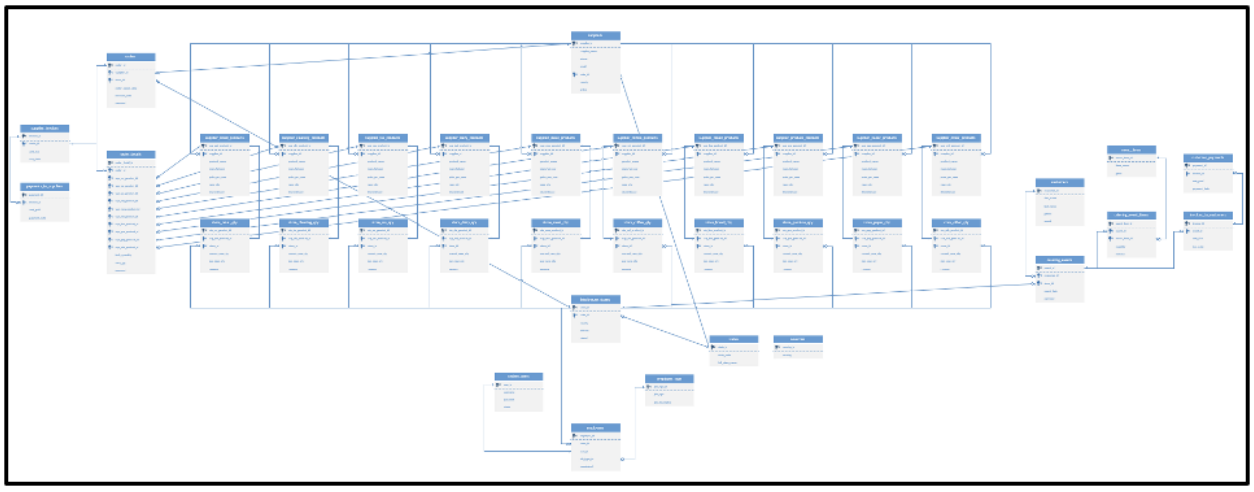
GUI: We have been developing our GUI so that we have a solid base to work with and have begun to integrate it with our database so that we can develop our forms/GUIs.

- Jake Simpson
- Jesse Requena
- Andres Pirela
- Andy Luong (floater)

Final Report (float team): We are currently working on the report required for the project and assisting the other sub-groups where necessary.

- Andy Luong (floater)
- Dan Thomas (floater)

Please copy and paste the latest version/screenshot of your projects Entity Relational Diagram (ERD):



<u>04/15/2021</u>

This week the ACE Systems solutions team has been finishing our GUI and final report. To do this, our group (8 people) has been split into 2 sub-groups. They are as follows:

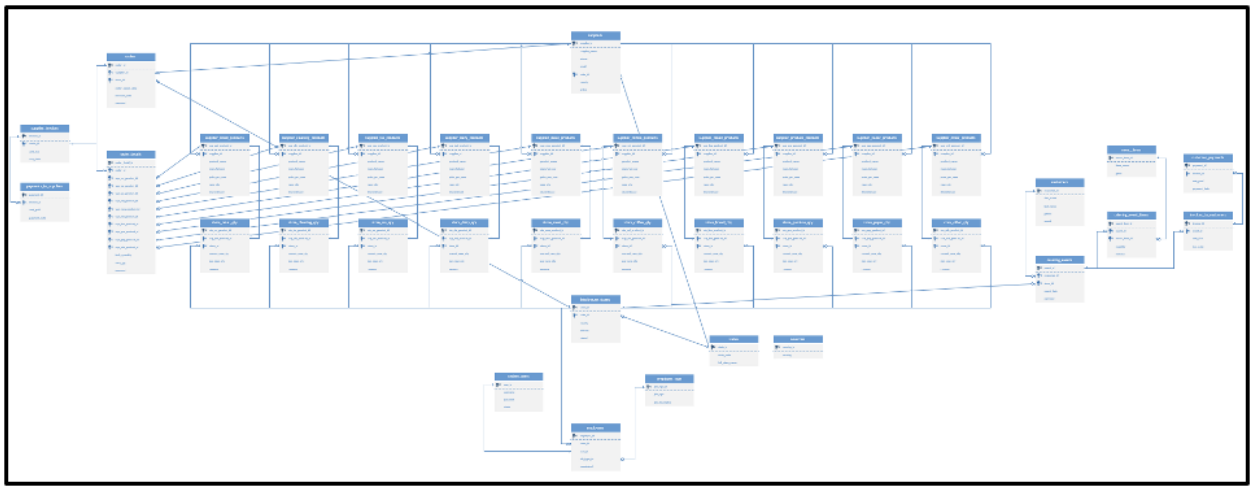
GUI: We are finalizing our GUI.

- Jake Simpson
- Jesse Requena
- Andres Pirela
- Joshua Wilson

Final Report: We are currently working on the report required for the project and assisting the GUI team as necessary.

- Andy Luong
- Dan Thomas
- Jorge Vasquez
- Frankie Gomez

Please copy and paste the latest version/screenshot of your projects Entity Relational Diagram (ERD):



Summary of Project Hours: (Please note that the hours are not exact, but they are **very** representative of the actual work distribution.)

Team Member Name	Deliverable Description	Hours Worked
Joshua Wilson		
1.	Database Design	40
4.	Database Implementation	50
5.	Project Report	25
6.	GUI Development	20
Jake Simpson		
1.	Database Design	10
4.	Database Implementation	20
5.	Project Report	20
6.	GUI Development	60
Andy Luong		
1.	Database Design	3
4.	Database Implementation	2
5.	Project Report	5
6.	GUI Development	2
Dan Thomas		
1.	Database Design	2
4.	Database Implementation	2
5.	Project Report	4
6.	GUI Development	1
Francisco Gomez		
1.	Database Design	10
4.	Database Implementation	10
5.	Project Report	8
6.	GUI Development	3

Andres Pirela

1.	Database Design	2
4.	Database Implementation	3
5.	Project Report	5
6.	GUI Development	2

Jesse Requena

1.	Database Design	3
4.	Database Implementation	2
5.	Project Report	4
6.	GUI Development	10

Jorge Vazquez

1.	Database Design	3
4.	Database Implementation	3
5.	Project Report	5
6.	GUI Development	2

Joshua Wilson Individual Contributions

QUERY/REPORT 1

REPORT NAME: Full store inventory report

BUSINESS PROBLEMS ADDRESSED: This report allows a store to see their full inventory across all categories. In terms of business problems, the user would want to use this because it allows the user to easily create a report for their store's current inventory which will then be sent to upper management.

REPORT DESCRIPTION: Used to show the complete current inventory of a given store. It shows product names and their quantities.

1. Shows full inventory for a given store.

a. View name: full_store_inventory_report

b. Where store_id = {}

SCREENSHOT OF QUERY RESULTS FOR STORE 1:

Results | Messages

	store_id	sto_dai_product_id	sto_bre_product_id	sto_cof_product_id	sto_meat_product_id	sto_pro_product_id	sto_pap_product_id	sto_sss_product_id	sto_oth_product_id	sto_ret_product_id	sto_cle_product_id	product_name	current_case_qty
1	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	1	Yellow DC33 Detergent	5
2	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	2	Blue Ultra Glass Cleaner	4
3	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	3	Purple Ultimo	4
4	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	4	SS Polish	11
5	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	5	Floor Cleaner	6
6	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	6	P&G Stainless Steel Polish	9
7	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	7	Quat Sanitizer	5
8	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	8	P&G Sanitizer	4
9	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	9	Dish Soap	8
10	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	10	Dawn Dish Soap	8
11	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	11	Pink Hand Soap	6
12	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	12	Foaming RR Hand Soap	11
13	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	13	Air Freshener	6
14	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	14	Grill Cleaner	11
15	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	15	Magic Eraser	5
16	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	16	Souring Pad	0
17	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	17	Griddle Polish Pad	1
18	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	18	60gal Trash Liner	5
19	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	19	30gal Trash Liner	3
20	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	20	Sponges	7
21	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	21	Espresso Cleaner	7
22	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	22	Medium Gloves	7
23	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	23	Large Gloves	11
24	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	1	NULL	BHCK Adult T-Shirt	2
25	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	2	NULL	BITB Onezie	5
26	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	3	NULL	BHCK Youth T-Shirt	5
27	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	4	NULL	BHCK Hat	2
28	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	5	NULL	BHCK Mugs	9
29	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	6	NULL	BHCK Patch	5
30	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	7	NULL	BHCK Retail Granola	7
31	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	8	NULL	G&G - Turkey Bacon Clu...	3
32	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	9	NULL	G&G - Chicken Salad Sa...	4
33	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	10	NULL	G&G - PB Apple Cinnam...	5
34	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	11	NULL	G&G - Cherry Chai Over...	6
35	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	12	NULL	G&G - PB Banana Over...	0
36	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	13	NULL	G&G - Yogurt & Granola	0
37	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	14	NULL	THTC - Honey - Individu...	9
38	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	15	NULL	THTC - Honey - Jar	8
39	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	16	NULL	Retail Pickles - Garlic Dil...	10
40	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	17	NULL	Retail Pickles - Habaner...	3
41	1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	18	NULL	HCSC - Custard Individu...	3

Query executed successfully. | COT-CIS3365-12 (13.0 RTM) | COUGARNET\jrwilt24 (62) | Blockhouse_DB | 00:00:00 | 242 rows

BEGIN (full_store_inventory_report) SCRIPT

```
USE Blockhouse_DB
GO

SELECT
                store_dairy_qty.store_id,
                store_dairy_qty.sto_dai_product_id,
                null as sto_bre_product_id,
```

```
                        null as sto_cof_product_id,
                        null as sto_meat_product_id,
                        null as sto_pro_product_id,
                        null as sto_pap_product_id,
                        null as sto_sss_product_id,
                        null as sto_oth_product_id,
                        null as sto_ret_product_id,
                        null as sto_cle_product_id,
                        suppliers_dairy_products.product_name,
                        store_dairy_qty.current_case_qty
FROM store_dairy_qty
                        INNER JOIN suppliers_dairy_products ON  suppliers_dairy_products.sup_dai_product_id = store_dairy_qty.sup_dai_product_id

UNION

SELECT
                        store_bread_qty.store_id,
                        null as sto_dai_product_id,
                        store_bread_qty.sto_bre_product_id,
                        null as sto_cof_product_id,
                        null as sto_meat_product_id,
                        null as sto_pro_product_id,
                        null as sto_pap_product_id,
                        null as sto_sss_product_id,
                        null as sto_oth_product_id,
                        null as sto_ret_product_id,
                        null as sto_cle_product_id,
                        suppliers_bread_products.product_name,
                        store_bread_qty.current_case_qty
FROM store_bread_qty
                        INNER JOIN suppliers_bread_products ON  suppliers_bread_products.sup_bre_product_id = store_bread_qty.sup_bre_product_id

UNION

SELECT
                        store_coffee_qty.store_id,
                        null as sto_dai_product_id,
                        null as sto_bre_product_id,
                        store_coffee_qty.sto_cof_product_id,
                        null as sto_meat_product_id,
                        null as sto_pro_product_id,
                        null as sto_pap_product_id,
                        null as sto_sss_product_id,
                        null as sto_oth_product_id,
                        null as sto_ret_product_id,
                        null as sto_cle_product_id,
                        suppliers_coffee_products.product_name,
                        store_coffee_qty.current_case_qty
FROM store_coffee_qty
                        INNER JOIN suppliers_coffee_products ON  suppliers_coffee_products.sup_cof_product_id = store_coffee_qty.sup_cof_product_id

UNION

SELECT
                        store_meat_qty.store_id,
                        null as sto_dai_product_id,
                        null as sto_bre_product_id,
                        null as sto_cof_product_id,
                        store_meat_qty.sto_mea_product_id,
                        null as sto_pro_product_id,
                        null as sto_pap_product_id,
                        null as sto_sss_product_id,
                        null as sto_oth_product_id,
                        null as sto_ret_product_id,
                        null as sto_cle_product_id,
                        suppliers_meat_products.product_name,
                        store_meat_qty.current_case_qty
FROM store_meat_qty
                        INNER JOIN suppliers_meat_products ON  suppliers_meat_products.sup_mea_product_id = store_meat_qty.sup_mea_product_id

UNION

SELECT
                        store_produce_qty.store_id,
                        null as sto_dai_product_id,
                        null as sto_bre_product_id,
                        null as sto_cof_product_id,
```

```
                        null as sto_mea_product_id,
                        store_produce_qty.sto_pro_product_id,
                        null as sto_pap_product_id,
                        null as sto_sss_product_id,
                        null as sto_oth_product_id,
                        null as sto_ret_product_id,
                        null as sto_cle_product_id,
                        suppliers_produce_products.product_name,
                        store_produce_qty.current_case_qty
FROM store_produce_qty
                        INNER JOIN suppliers_produce_products ON  suppliers_produce_products.sup_pro_product_id = store_produce_qty.sup_pro_product_id

UNION


SELECT
                        store_paper_qty.store_id,
                        null as sto_dai_product_id,
                        null as sto_bre_product_id,
                        null as sto_cof_product_id,
                        null as sto_mea_product_id,
                        null as sto_pro_product_id,
                        store_paper_qty.sto_pap_product_id,
                        null as sto_sss_product_id,
                        null as sto_oth_product_id,
                        null as sto_ret_product_id,
                        null as sto_cle_product_id,
                        suppliers_paper_products.product_name,
                        store_paper_qty.current_case_qty
FROM store_paper_qty
                        INNER JOIN suppliers_paper_products ON  suppliers_paper_products.sup_pap_product_id = store_paper_qty.sup_pap_product_id

UNION


SELECT
                         store_sss_qty.store_id,
                         null as sto_dai_product_id,
                         null as sto_bre_product_id,
                         null as sto_cof_product_id,
                         null as sto_mea_product_id,
                         null as sto_pro_product_id,
                         null as sto_pap_product_id,
                         store_sss_qty.sto_sss_product_id,
                         null as sto_oth_product_id,
                         null as sto_ret_product_id,
                         null as sto_cle_product_id,
                         suppliers_sss_products.product_name,
                         Store_sss_qty.current_case_qty
FROM store_sss_qty
                        INNER JOIN suppliers_sss_products ON  suppliers_sss_products.sup_sss_product_id = store_sss_qty.sup_sss_product_id

UNION


SELECT
                         store_other_qty.store_id,
                         null as sto_dai_product_id,
                         null as sto_bre_product_id,
                         null as sto_cof_product_id,
                         null as sto_mea_product_id,
                         null as sto_pro_product_id,
                         null as sto_pap_product_id,
                         null as sto_sss_product_id,
                         store_other_qty.sto_oth_product_id,
                         null as sto_ret_product_id,
                         null as sto_cle_product_id,
                         suppliers_other_products.product_name,
                         store_other_qty.current_case_qty
FROM store_other_qty
                        INNER JOIN suppliers_other_products ON  suppliers_other_products.sup_oth_product_id = store_other_qty.sup_oth_product_id

UNION


SELECT
                          store_retail_qty.store_id,
                          null as sto_dai_product_id,
                          null as sto_bre_product_id,
                          null as sto_cof_product_id,
                          null as sto_mea_product_id,
                          null as sto_pro_product_id,
```

```
                        null as sto_pap_product_id,
                        null as sto_sss_product_id,
                        null as sto_oth_product_id,
                        store_retail_qty.sto_ret_product_id,
                        null as sto_cle_product_id,
                        suppliers_retail_products.product_name,
                        Store_retail_qty.current_case_qty
FROM store_retail_qty
                    INNER JOIN suppliers_retail_products ON  suppliers_retail_products.sup_ret_product_id = store_retail_qty.sup_ret_product_id

UNION

SELECT
                        store_cleaning_qty.store_id,
                        null as sto_dai_product_id,
                        null as sto_bre_product_id,
                        null as sto_cof_product_id,
                        null as sto_mea_product_id,
                        null as sto_pro_product_id,
                        null as sto_pap_product_id,
                        null as sto_sss_product_id,
                        null as sto_oth_product_id,
                        null as sto_ret_product_id,
                        store_cleaning_qty.sto_cle_product_id,
                        suppliers_cleaning_products.product_name,
                        Store_cleaning_qty.current_case_qty
FROM store_cleaning_qty
                    INNER JOIN suppliers_cleaning_products ON  suppliers_cleaning_products.sup_cle_product_id = store_cleaning_qty.sup_cle_product_id
```

END (full_store_inventory_report) SCRIPT

QUERY/REPORT 2

REPORT NAME: Full Product Order Report

BUSINESS PROBLEMS ADDRESSED: This allows a user to see all items in a product order. This allows for quick checking of what has been ordered. A user would want to use this report so that they can verify they ordered what they intended to and check it against what was received.

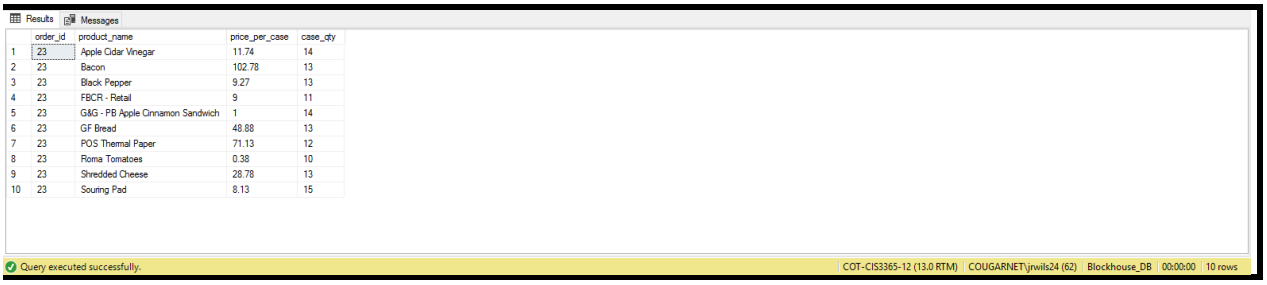
REPORT DESCRIPTION: Shows a full list of products for a given order.

1. Shows all products names, their price per case, and their quantities for a given order.

a. View name: order_report

b. Where order_id = {}

SCREENSHOT OF QUERY RESULTS FOR ORDER 23:



Results | Messages

	order_id	product_name	price_per_case	case_qty
1	23	Apple Cidar Vinegar	11.74	14
2	23	Bacon	102.78	13
3	23	Black Pepper	9.27	13
4	23	FBCR - Retail	9	11
5	23	G&G - PB Apple Cinnamon Sandwich	1	14
6	23	GF Bread	48.88	13
7	23	POS Thermal Paper	71.13	12
8	23	Roma Tomatoes	0.38	10
9	23	Shredded Cheese	28.78	13
10	23	Souring Pad	8.13	15

Query executed successfully. | COT-CIS3365-12 (13.0 RTM) | COUGARNET\jrwils24 (62) | Blockhouse_DB | 00:00:00 | 10 rows

BEGIN (order_report) SCRIPT

```
USE Blockhouse_DB
GO

SELECT
                order_details.order_id,
                suppliers_bread_products.product_name,
                suppliers_bread_products.price_per_case,
                order_details.case_qty
FROM order_details
                INNER JOIN orders ON orders.order_id = order_details.order_id
                INNER JOIN suppliers_bread_products ON suppliers_bread_products.sup_bre_product_id = order_details.sup_bre_product_id

UNION

SELECT
                order_details.order_id,
                suppliers_dairy_products.product_name,
                suppliers_dairy_products.price_per_case,
                order_details.case_qty
FROM order_details
                INNER JOIN orders ON orders.order_id = order_details.order_id
                INNER JOIN suppliers_dairy_products ON suppliers_dairy_products.sup_dai_product_id = order_details.sup_dai_product_id


UNION

SELECT
                order_details.order_id,
                suppliers_coffee_products.product_name,
                suppliers_coffee_products.price_per_case,
                order_details.case_qty
FROM order_details
                INNER JOIN orders ON orders.order_id = order_details.order_id
                INNER JOIN suppliers_coffee_products ON suppliers_coffee_products.sup_cof_product_id = order_details.sup_cof_product_id


UNION

SELECT
                order_details.order_id,
                suppliers_meat_products.product_name,
                suppliers_meat_products.price_per_case,
                order_details.case_qty
FROM order_details
                INNER JOIN orders ON orders.order_id = order_details.order_id
                INNER JOIN suppliers_meat_products ON suppliers_meat_products.sup_mea_product_id = order_details.sup_mea_product_id


UNION

SELECT
                order_details.order_id,
                suppliers_produce_products.product_name,
                suppliers_produce_products.price_per_case,
                order_details.case_qty
FROM order_details
                INNER JOIN orders ON orders.order_id = order_details.order_id
                INNER JOIN suppliers_produce_products ON suppliers_produce_products.sup_pro_product_id = order_details.sup_pro_product_id
```

```
UNION

SELECT
                order_details.order_id,
                suppliers_paper_products.product_name,
                suppliers_paper_products.price_per_case,
                order_details.case_qty
FROM order_details
                INNER JOIN orders ON orders.order_id = order_details.order_id
                INNER JOIN suppliers_paper_products ON suppliers_paper_products.sup_pap_product_id = order_details.sup_pap_product_id


UNION

SELECT
                order_details.order_id,
                suppliers_sss_products.product_name,
                suppliers_sss_products.price_per_case,
                order_details.case_qty
FROM order_details
                INNER JOIN orders ON orders.order_id = order_details.order_id
                INNER JOIN suppliers_sss_products ON suppliers_sss_products.sup_sss_product_id = order_details.sup_sss_product_id


UNION

SELECT
                order_details.order_id,
                suppliers_other_products.product_name,
                suppliers_other_products.price_per_case,
                order_details.case_qty
FROM order_details
                INNER JOIN orders ON orders.order_id = order_details.order_id
                INNER JOIN suppliers_other_products ON suppliers_other_products.sup_oth_product_id = order_details.sup_oth_product_id


UNION

SELECT
                order_details.order_id,
                suppliers_retail_products.product_name,
                suppliers_retail_products.price_per_case,
                order_details.case_qty
FROM order_details
                INNER JOIN orders ON orders.order_id = order_details.order_id
                INNER JOIN suppliers_retail_products ON suppliers_retail_products.sup_ret_product_id = order_details.sup_ret_product_id


UNION

SELECT
                order_details.order_id,
                suppliers_cleaning_products.product_name,
                suppliers_cleaning_products.price_per_case,
                order_details.case_qty
FROM order_details
                INNER JOIN orders ON orders.order_id = order_details.order_id
                INNER JOIN suppliers_cleaning_products ON suppliers_cleaning_products.sup_cle_product_id = order_details.sup_cle_product_id
```

END (order_report) SCRIPT

QUERY/REPORT 3

REPORT NAME: Catering Event Report

BUSINESS PROBLEMS ADDRESSED: This report allows a user to see what items were ordered for a catering event. The user would want to use this because they can use the contact information to check in with the customer, verify their order, and even have a list of what needs to be made when the event day comes.

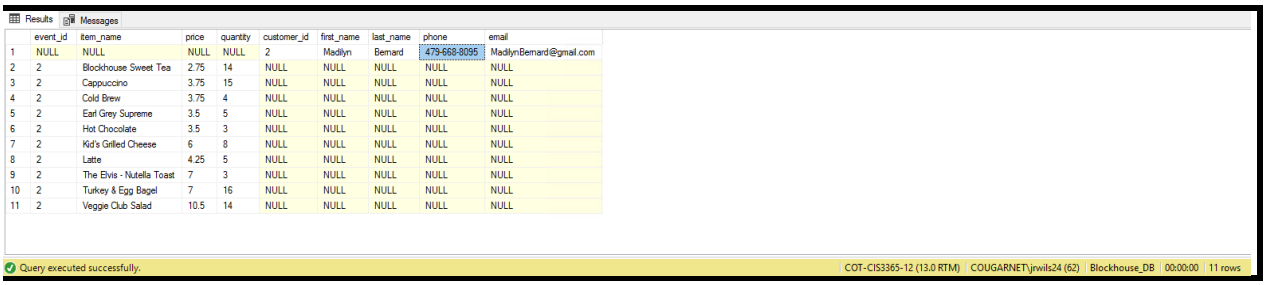
REPORT DESCRIPTION: Used to provide an overview of a customer order for a catering event.

1. Shows customer contact information for the given id. And when given their corresponding event_id, it shows the items in that catering event.

a. View name: catering_event_report

b. Where customer_id = {} or event_id = {}

SCREENSHOT OF QUERY RESULTS FOR CUSTOMER 2 WITH EVENT ID 2:



Results | Messages

	event_id	item_name	price	quantity	customer_id	first_name	last_name	phone	email
1	NULL	NULL	NULL	NULL	2	Madilyn	Bernard	479-668-8095	MadilynBernard@gmail.com
2	2	Blockhouse Sweet Tea	2.75	14	NULL	NULL	NULL	NULL	NULL
3	2	Cappuccino	3.75	15	NULL	NULL	NULL	NULL	NULL
4	2	Cold Brew	3.75	4	NULL	NULL	NULL	NULL	NULL
5	2	Earl Grey Supreme	3.5	5	NULL	NULL	NULL	NULL	NULL
6	2	Hot Chocolate	3.5	3	NULL	NULL	NULL	NULL	NULL
7	2	Kid's Grilled Cheese	6	8	NULL	NULL	NULL	NULL	NULL
8	2	Latte	4.25	5	NULL	NULL	NULL	NULL	NULL
9	2	The Elvis - Nutella Toast	7	3	NULL	NULL	NULL	NULL	NULL
10	2	Turkey & Egg Bagel	7	16	NULL	NULL	NULL	NULL	NULL
11	2	Veggie Club Salad	10.5	14	NULL	NULL	NULL	NULL	NULL

Query executed successfully. | COT-CIS3365-12 (13.0 RTM) | COUGARNET\jrwils24 (62) | Blockhouse_DB | 00:00:00 | 11 rows

BEGIN (catering_event_report) SCRIPT

```
USE Blockhouse_DB
GO

SELECT
                catering_event_items.event_id,
                menu_items.item_name,
                menu_items.price,
                catering_event_items.quantity,
                null as customer_id,
                null as first_name,
                null as last_name,
                null as phone,
                null as email
FROM catering_events
                INNER JOIN customers ON customers.customer_id = catering_events.customer_id
                INNER JOIN catering_event_items ON catering_event_items.event_id = catering_events.event_id
                INNER JOIN menu_items ON catering_event_items.menu_item_id = menu_items.menu_item_id

UNION

SELECT
                null as event_id,
                null as item_name,
```

```
                    null as price,
                    null as quantity,
                    customers.customer_id,
                    customers.first_name,
                    customers.last_name,
                    customers.phone,
                    customers.email
FROM customers
```

END (catering_event_report) SCRIPT

QUERY/REPORT 4

REPORT NAME: Order Payments Verification

BUSINESS PROBLEMS ADDRESSED: This report shows any payment amounts not equaling the cost. A user would want to use this report to verify that all invoices for orders had been paid correctly. Ideally, this report would return no results.

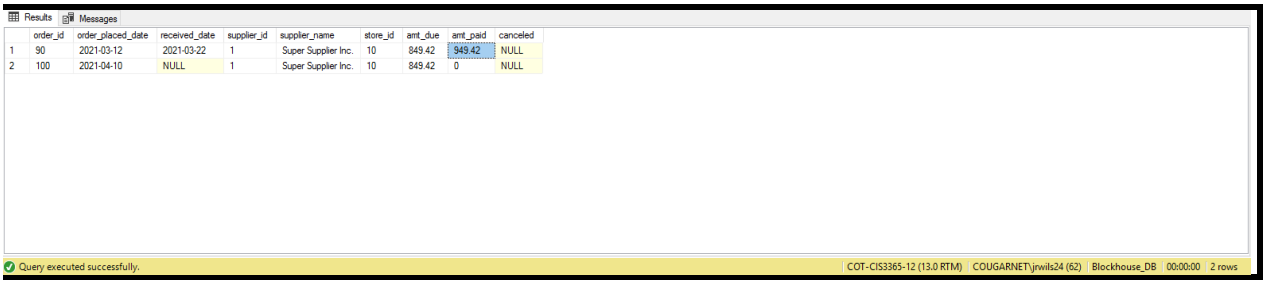
REPORT DESCRIPTION: Used to show whether a store has any incorrectly paid, or unpaid amounts on supplier invoices.

1. Shows all orders for a given store, their cost, and the amounts paid if the amount due and the amount paid are not equal.

a. View name: **order_payment_verification**

b. Where store_id = {} and amt_paid <> amt_due

SCREENSHOT OF QUERY RESULTS FOR STORE 10:


Results Messages

	order_id	order_placed_date	received_date	supplier_id	supplier_name	store_id	amt_due	amt_paid	canceled
1	90	2021-03-12	2021-03-22	1	Super Supplier Inc.	10	849.42	949.42	NULL
2	100	2021-04-10	NULL	1	Super Supplier Inc.	10	849.42	0	NULL

Query executed successfully. COT-CIS3365-12 (13.0 RTM) COUGARNET\jrwills24 (62) Blockhouse_DB 00:00:00 2 rows

BEGIN (order_payment_verification) SCRIPT

```
USE Blockhouse_DB
GO
```

```
SELECT
		orders.order_id,
		orders.order_placed_date,
		orders.received_date,
		suppliers.supplier_id,
		suppliers.supplier_name,
		blockhouse_stores.store_id,
		supplier_invoices.amt_due,
		payments_to_suppliers.amt_paid,
		orders.canceled
FROM orders
		INNER JOIN blockhouse_stores ON blockhouse_stores.store_id = orders.store_id
		INNER JOIN suppliers ON suppliers.supplier_id = orders.supplier_id
		INNER JOIN supplier_invoices ON supplier_invoices.order_id = orders.order_id
		INNER JOIN payments_to_suppliers ON payments_to_suppliers.invoice_id = supplier_invoices.invoice_id
```

END (order_payment_verification) SCRIPT

COPIES OF CREATE SCRIPTS

```
CREATE TABLE orders (
order_id INT IDENTITY(1,1) PRIMARY KEY,
supplier_id INT NOT NULL,
store_id INT NOT NULL,
order_placed_date DATE NOT NULL,
received_date DATE,
canceled BIT
);
```

```
CREATE TABLE order_details  (
order_detail_id INT IDENTITY(1,1) PRIMARY KEY,
order_id INT NOT NULL,
sup_dai_product_id INT,
sup_cof_product_id INT,
sup_mea_product_id INT,
sup_bre_product_id INT,
sup_pro_product_id INT,
sup_pap_product_id INT,
sup_sss_product_id INT,
sup_oth_product_id INT,
sup_ret_product_id INT,
sup_cle_product_id INT,
case_qty INT NOT NULL,
removed BIT
);
```

```
CREATE TABLE supplier_invoices (
invoice_id int IDENTITY(1,1) PRIMARY KEY,
order_id int NOT NULL,
amt_due float NOT NULL,
due_date date NOT NULL
);
```

```
CREATE TABLE payments_to_suppliers (
payment_id int IDENTITY(1,1) PRIMARY KEY,
invoice_id int NOT NULL,
amt_paid float NOT NULL,
payment_date date NOT NULL
);
```

```
CREATE TABLE blockhouse_stores (
store_id INT IDENTITY(1,1) PRIMARY KEY,
state_id INT NOT NULL,
city varchar(255) NOT NULL,
address varchar(255) NOT NULL,
closed BIT,
);
```

COPIES OF ALTER TABLE SCRIPTS

```
ALTER TABLE Orders
ADD FOREIGN KEY (supplier_id) REFERENCES suppliers(supplier_id),
ADD FOREIGN KEY (store_id) REFERENCES blockhouse_stores(store_id);
```

```
ALTER TABLE order_details
ADD FOREIGN KEY (order_id) REFERENCES orders(order_id),
ADD FOREIGN KEY (sup_dai_product_id) REFERENCES
suppliers_dairy_products(sup_dai_product_id),
ADD FOREIGN KEY (sup_cof_product_id) REFERENCES
suppliers_coffee_products(sup_cof_product_id),
```

```
ADD FOREIGN KEY (sup_mea_product_id) REFERENCES
suppliers_meat_products(sup_mea_product_id),
ADD FOREIGN KEY (sup_bre_product_id) REFERENCES
suppliers_bread_products(sup_bre_product_id),
ADD FOREIGN KEY (sup_pro_product_id) REFERENCES
suppliers_produce_products(sup_pro_product_id),
ADD FOREIGN KEY (sup_pap_product_id) REFERENCES
suppliers_paper_products(sup_pap_product_id),
ADD FOREIGN KEY (sup_sss_product_id) REFERENCES
suppliers_sss_products(sup_sss_product_id),
ADD FOREIGN KEY (sup_oth_product_id) REFERENCES
suppliers_other_products(sup_oth_product_id),
ADD FOREIGN KEY (sup_ret_product_id) REFERENCES
suppliers_retail_products(sup_ret_product_id),
ADD FOREIGN KEY (sup_cle_product_id) REFERENCES
suppliers_cleaning_products(sup_cle_product_id);
```

```
ALTER TABLE supplier_invoices
ADD FOREIGN KEY (order_id) REFERENCES orders(order_id);
```

```
ALTER TABLE payments_to_suppliers
ADD FOREIGN KEY (invoice_id) REFERENCES supplier_invoices(invoice_id);
```

```
ALTER TABLE order_details
ADD CONSTRAINT one_is_not_null
CHECK(
        (sup_dai_product_id IS NOT NULL AND sup_cof_product_id IS NULL AND sup_mea_product_id IS
NULL AND sup_bre_product_id IS NULL AND sup_pro_product_id IS NULL AND sup_pap_product_id IS
NULL AND sup_sss_product_id IS NULL AND sup_oth_product_id IS NULL AND sup_ret_product_id IS
NULL AND sup_cle_product_id IS NULL)
OR  (sup_dai_product_id IS NULL AND sup_cof_product_id IS NOT NULL AND sup_mea_product_id IS
NULL AND sup_bre_product_id IS NULL AND sup_pro_product_id IS NULL AND sup_pap_product_id IS
NULL AND sup_sss_product_id IS NULL AND sup_oth_product_id IS NULL AND sup_ret_product_id IS
NULL AND sup_cle_product_id IS NULL)
OR  (sup_dai_product_id IS NULL AND sup_cof_product_id IS NULL AND sup_mea_product_id IS NOT
NULL AND sup_bre_product_id IS NULL AND sup_pro_product_id IS NULL AND sup_pap_product_id IS
NULL AND sup_sss_product_id IS NULL AND sup_oth_product_id IS NULL AND sup_ret_product_id IS
NULL AND sup_cle_product_id IS NULL)
OR  (sup_dai_product_id IS NULL AND sup_cof_product_id IS NULL AND sup_mea_product_id IS NULL
AND sup_bre_product_id IS NOT NULL AND sup_pro_product_id IS NULL AND sup_pap_product_id IS
NULL AND sup_sss_product_id IS NULL AND sup_oth_product_id IS NULL AND sup_ret_product_id IS
NULL AND sup_cle_product_id IS NULL)
```

```
OR  (sup_dai_product_id IS NULL AND sup_cof_product_id IS NULL AND sup_mea_product_id IS NULL
AND sup_bre_product_id IS NULL AND sup_pro_product_id IS NOT NULL AND sup_pap_product_id IS
NULL AND sup_sss_product_id IS NULL AND sup_oth_product_id IS NULL AND sup_ret_product_id IS
NULL AND sup_cle_product_id IS NULL)
OR  (sup_dai_product_id IS NULL AND sup_cof_product_id IS NULL AND sup_mea_product_id IS NULL
AND sup_bre_product_id IS NULL AND sup_pro_product_id IS NULL AND sup_pap_product_id IS NOT
NULL AND sup_sss_product_id IS NULL AND sup_oth_product_id IS NULL AND sup_ret_product_id IS
NULL AND sup_cle_product_id IS NULL)
OR  (sup_dai_product_id IS NULL AND sup_cof_product_id IS NULL AND sup_mea_product_id IS NULL
AND sup_bre_product_id IS NULL AND sup_pro_product_id IS NULL AND sup_pap_product_id IS NULL
AND sup_sss_product_id IS NOT NULL AND sup_oth_product_id IS NULL AND sup_ret_product_id IS NULL
AND sup_cle_product_id IS NULL)
OR  (sup_dai_product_id IS NULL AND sup_cof_product_id IS NULL AND sup_mea_product_id IS NULL
AND sup_bre_product_id IS NULL AND sup_pro_product_id IS NULL AND sup_pap_product_id IS NULL
AND sup_sss_product_id IS NULL AND sup_oth_product_id IS NOT NULL AND sup_ret_product_id IS NULL
AND sup_cle_product_id IS NULL)
OR  (sup_dai_product_id IS NULL AND sup_cof_product_id IS NULL AND sup_mea_product_id IS NULL
AND sup_bre_product_id IS NULL AND sup_pro_product_id IS NULL AND sup_pap_product_id IS NULL
AND sup_sss_product_id IS NULL AND sup_oth_product_id IS NULL AND sup_ret_product_id IS NOT NULL
AND sup_cle_product_id IS NULL)
OR  (sup_dai_product_id IS NULL AND sup_cof_product_id IS NULL AND sup_mea_product_id IS NULL
AND sup_bre_product_id IS NULL AND sup_pro_product_id IS NULL AND sup_pap_product_id IS NULL
AND sup_sss_product_id IS NULL AND sup_oth_product_id IS NULL AND sup_ret_product_id IS NULL AND
sup_cle_product_id IS NOT NULL)
);
```

COPIES OF BULK INSERT SCRIPTS

```
BULK INSERT order_details
FROM 'C:\order_details.txt'
WITH (
        FIELDTERMINATOR = ',',
        ROWTERMINATOR = '\n',
        KEEPNULLS
);
```

```
BULK INSERT orders
FROM 'C:\orders.txt'
WITH (
        FIELDTERMINATOR = ',',
        ROWTERMINATOR = '\n',
        KEEPNULLS
```

```
);
```

```
BULK INSERT supplier_invoices
FROM 'C:\supplier_invoices.txt'
WITH (
        FIELDTERMINATOR = ',',
        ROWTERMINATOR = '\n',
        KEEPNULLS
);
```

```
BULK INSERT payments_to_suppliers
FROM 'C:\payments_to_suppliers.txt'
WITH (
        FIELDTERMINATOR = ',',
        ROWTERMINATOR = '\n',
        KEEPNULLS
);
```

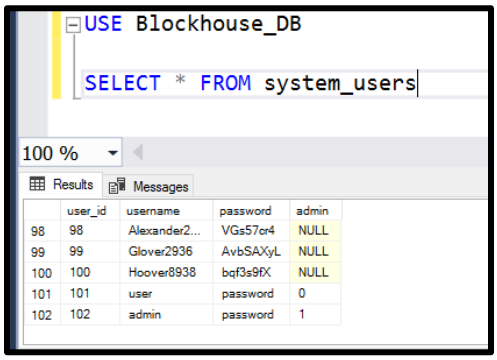
```
BULK INSERT blockhouse_stores
FROM 'C:\blockhouse_stores.txt'
WITH (
        FIELDTERMINATOR = ',',
        ROWTERMINATOR = '\n',
        KEEPNULLS
);
```

COPIES OF ALL SQL UPDATE SCRIPTS

INSERT SCRIPT #1 FOR SYSTEM_USERS

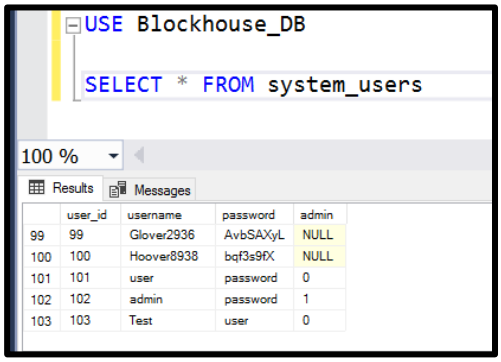
```
USE Blockhouse_DB
INSERT system_users
VALUES('Test','user',' ');
```

Before:



	user_id	username	password	admin
98	98	Alexander2...	VGs57cr4	NULL
99	99	Glover2936	AvbSAXyL	NULL
100	100	Hoover8938	bqf3s9fX	NULL
101	101	user	password	0
102	102	admin	password	1

After:

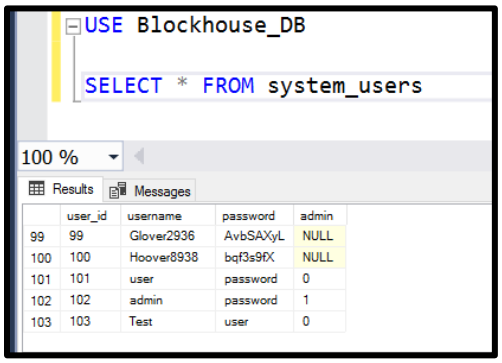


	user_id	username	password	admin
99	99	Glover2936	AvbSAXyL	NULL
100	100	Hoover8938	bqf3s9fX	NULL
101	101	user	password	0
102	102	admin	password	1
103	103	Test	user	0

UPDATE SCRIPT #1 FOR SYSTEM_USERS

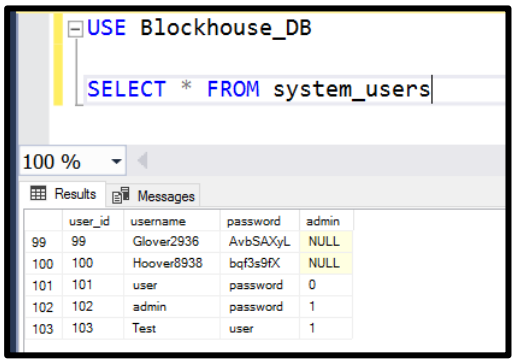
```
USE Blockhouse_DB
UPDATE system_users
SET admin = '1'
WHERE user_id = 103;
```

Before:



	user_id	username	password	admin
99	99	Glover2936	AvbSAXyL	NULL
100	100	Hoover8938	bqf3s9fX	NULL
101	101	user	password	0
102	102	admin	password	1
103	103	Test	user	0

After:

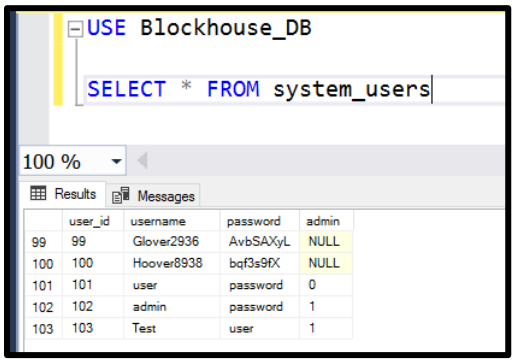


	user_id	username	password	admin
99	99	Glover2936	AvbSAXyL	NULL
100	100	Hoover8938	bqf3s9fX	NULL
101	101	user	password	0
102	102	admin	password	1
103	103	Test	user	1

DELETE SCRIPT #1 FOR SYSTEM_USERS

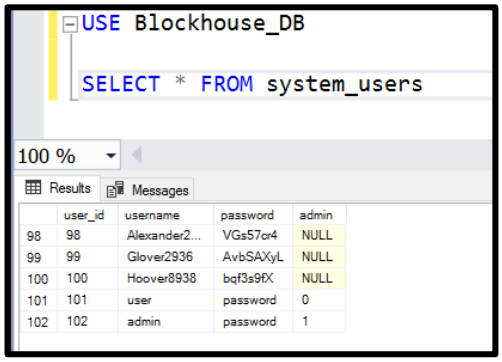
```
USE Blockhouse_DB
DELETE FROM system_users
WHERE user_id = 103;
```

Before:



	user_id	username	password	admin
99	99	Glover2936	AvbSAXyL	NULL
100	100	Hoover8938	bqf3s9fX	NULL
101	101	user	password	0
102	102	admin	password	1
103	103	Test	user	1

After:

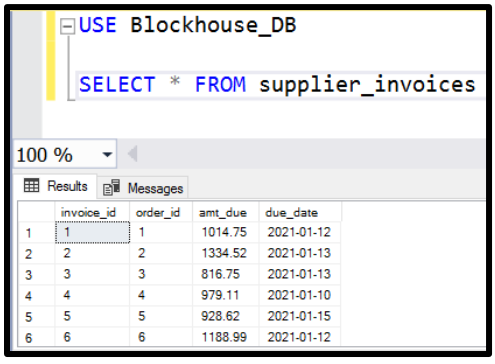


	user_id	username	password	admin
98	98	Alexander2...	VGs57cr4	NULL
99	99	Glover2936	AvbSAXyL	NULL
100	100	Hoover8938	bqf3s9fX	NULL
101	101	user	password	0
102	102	admin	password	1

INSERT SCRIPT #2 FOR SUPPLIER_INVOICES

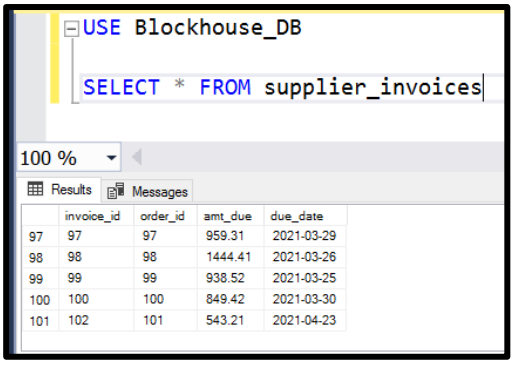
```
USE Blockhouse_DB
INSERT supplier_invoices
VALUES('101','543.21','2021-04-23');
```

Before:



	invoice_id	order_id	amt_due	due_date
1	1	1	1014.75	2021-01-12
2	2	2	1334.52	2021-01-13
3	3	3	816.75	2021-01-13
4	4	4	979.11	2021-01-10
5	5	5	928.62	2021-01-15
6	6	6	1188.99	2021-01-12

After:

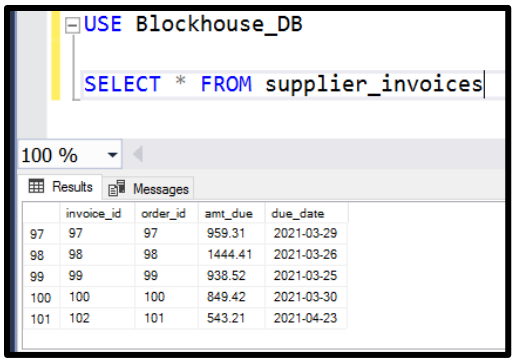


	invoice_id	order_id	amt_due	due_date
97	97	97	959.31	2021-03-29
98	98	98	1444.41	2021-03-26
99	99	99	938.52	2021-03-25
100	100	100	849.42	2021-03-30
101	102	101	543.21	2021-04-23

UPDATE SCRIPT #2 FOR SUPPLIER_INVOICES

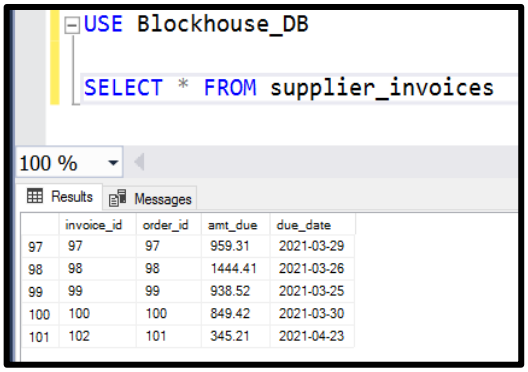
```
USE Blockhouse_DB
UPDATE supplier_invoices
SET amt_due = '345.21'
WHERE invoice_id = 102;
```

Before:



	invoice_id	order_id	amt_due	due_date
97	97	97	959.31	2021-03-29
98	98	98	1444.41	2021-03-26
99	99	99	938.52	2021-03-25
100	100	100	849.42	2021-03-30
101	102	101	543.21	2021-04-23

After:

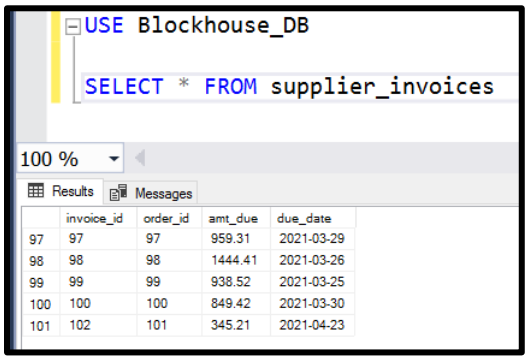


	invoice_id	order_id	amt_due	due_date
97	97	97	959.31	2021-03-29
98	98	98	1444.41	2021-03-26
99	99	99	938.52	2021-03-25
100	100	100	849.42	2021-03-30
101	102	101	345.21	2021-04-23

DELETE SCRIPT #2 FOR SUPPLIER_INVOICES

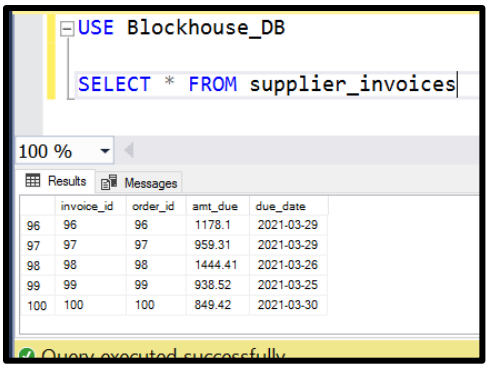
```
USE Blockhouse_DB
DELETE FROM supplier_invoices
WHERE invoice_id = 102;
```

Before:



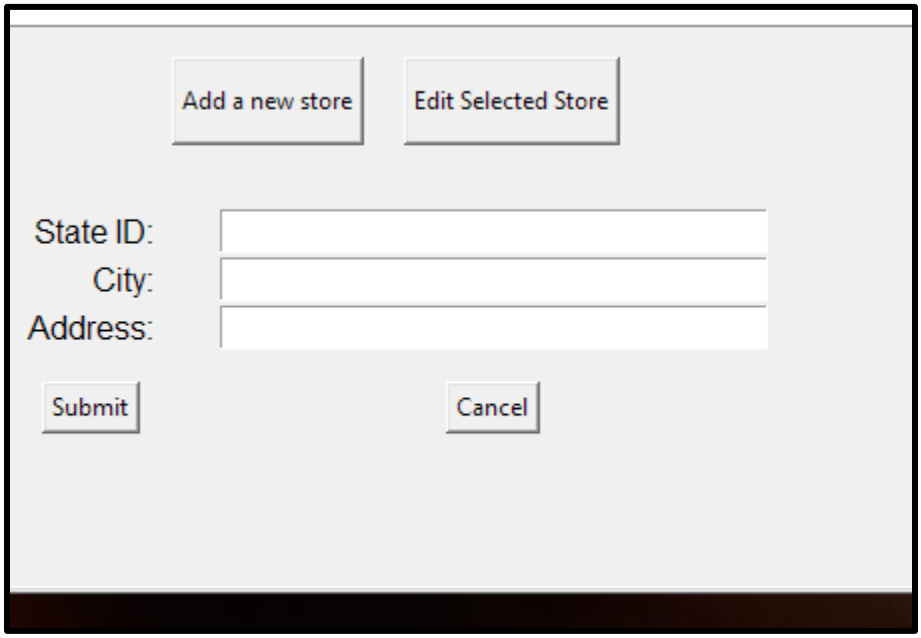
	invoice_id	order_id	amt_due	due_date
97	97	97	959.31	2021-03-29
98	98	98	1444.41	2021-03-26
99	99	99	938.52	2021-03-25
100	100	100	849.42	2021-03-30
101	102	101	345.21	2021-04-23

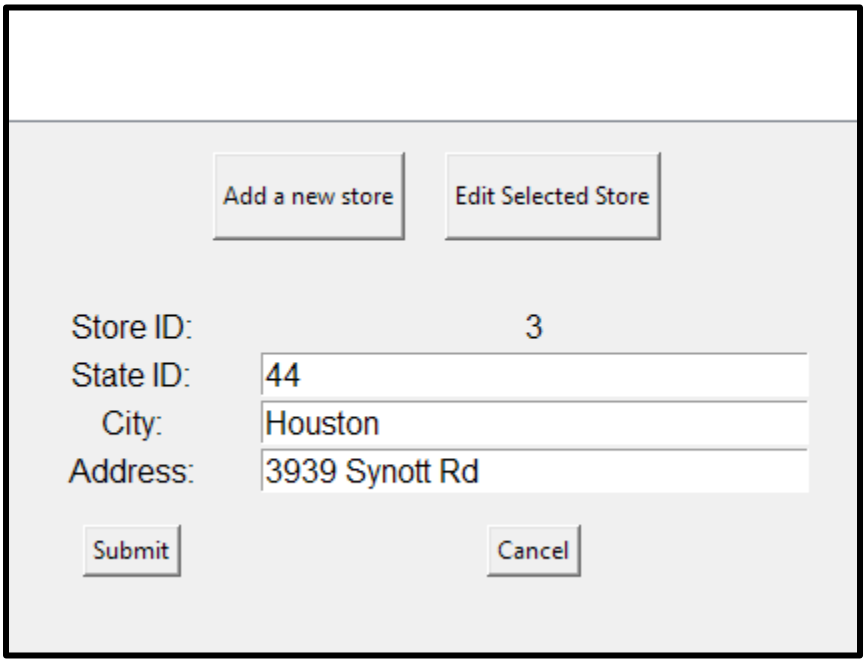
After:

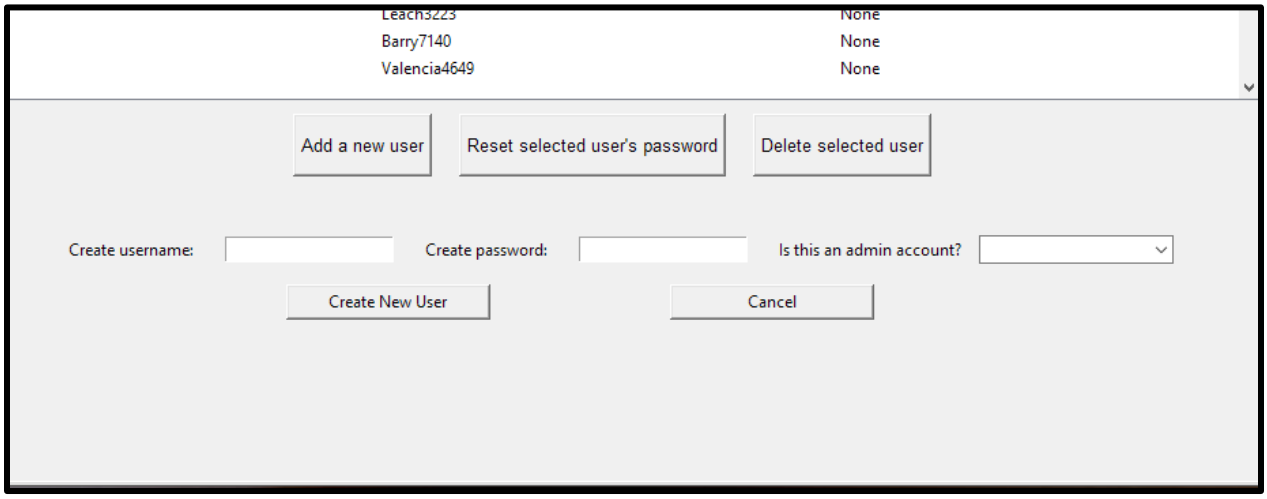


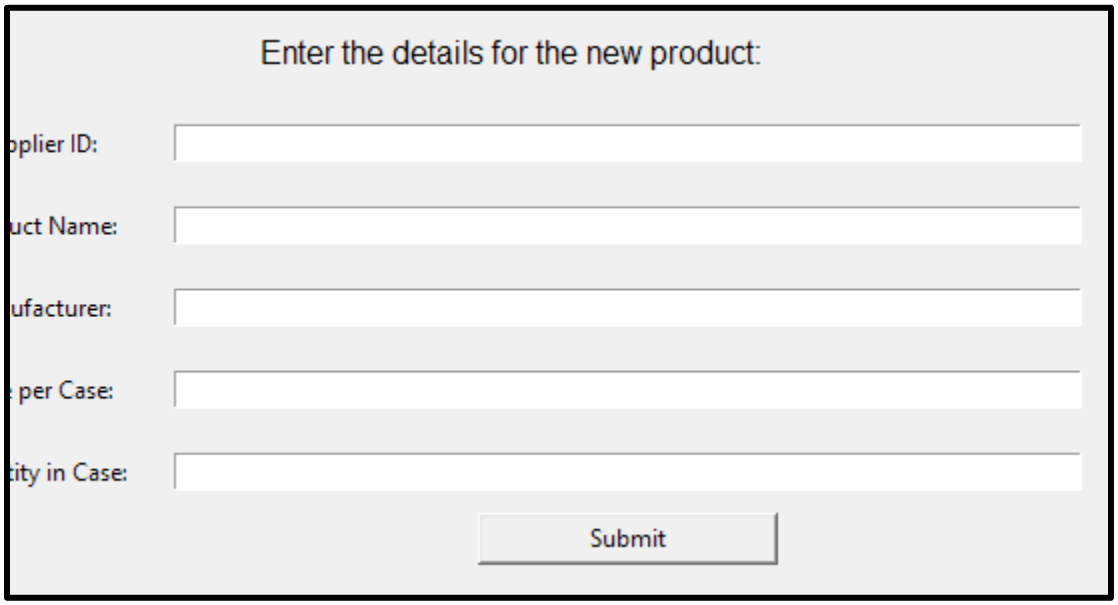
	invoice_id	order_id	amt_due	due_date
96	96	96	1178.1	2021-03-29
97	97	97	959.31	2021-03-29
98	98	98	1444.41	2021-03-26
99	99	99	938.52	2021-03-25
100	100	100	849.42	2021-03-30

SCREENSHOTS OF FORMS/GUIs






Leach3223
None
Barry7140
None
Valencia4649
None
Add a new user
Reset selected user's password
Delete selected user
Create username:
Create password:
Is this an admin account?
Create New User
Cancel


Enter the details for the new product:
pplier ID:
uct Name:
ufacturer:
per Case:
ity in Case:
Submit

Jake Simpson Individual Contributions

QUERY/REPORT 1

REPORT NAME: Produce To Order Report

BUSINESS PROBLEMS ADDRESSED: This report shows which produce products need to be ordered. A user would want to use this report because it allows them to quickly determine which produce products are not currently at par levels and they can construct an order appropriately.

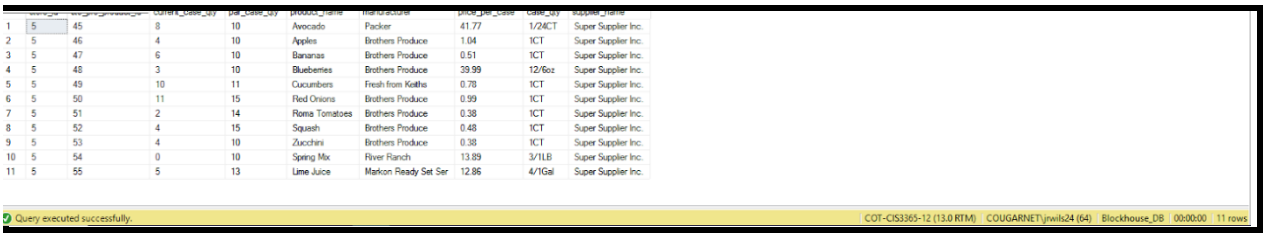
REPORT DESCRIPTION: Used to show the current produce inventory of a given store if it is less than the corresponding par level. It shows product names and their quantities.

1. For showing current produce quantity (if current less than par) of a particular store.

a. View name: produce_qty_to_order

b. WHERE store_id = {} and current_case_qty < par_case_qty

SCREENSHOT OF QUERY RESULTS FOR STORE 5:



1	5	45	8	10	Avocado	Packer	41.77	1/24CT	Super Supplier Inc.
2	5	46	4	10	Apples	Brothers Produce	1.04	1CT	Super Supplier Inc.
3	5	47	6	10	Bananas	Brothers Produce	0.51	1CT	Super Supplier Inc.
4	5	48	3	10	Blueberries	Brothers Produce	39.99	12/6oz	Super Supplier Inc.
5	5	49	10	11	Cucumbers	Fresh from Keiths	0.78	1CT	Super Supplier Inc.
6	5	50	11	15	Red Onions	Brothers Produce	0.99	1CT	Super Supplier Inc.
7	5	51	2	14	Roma Tomatoes	Brothers Produce	0.38	1CT	Super Supplier Inc.
8	5	52	4	15	Squash	Brothers Produce	0.48	1CT	Super Supplier Inc.
9	5	53	4	10	Zucchini	Brothers Produce	0.38	1CT	Super Supplier Inc.
10	5	54	0	10	Spring Mix	River Ranch	13.89	3/1LB	Super Supplier Inc.
11	5	55	5	13	Lime Juice	Markon Ready Set Ser	12.86	4/1Gal	Super Supplier Inc.

Query executed successfully. COT-CIS3365-12 (13.0 RTM) COUGARNET\jrwils24 (64) Blockhouse_DB 00:00:00 11 rows

BEGIN (produce_qty_to_order) SCRIPT

```
SELECT
                blockhouse_stores.store_id,
                store_produce_qty.sto_pro_product_id,
                store_produce_qty.current_case_qty,
                store_produce_qty.par_case_qty,
                suppliers_produce_products.product_name,
                suppliers_produce_products.manufacturer,
                suppliers_produce_products.price_per_case,
                Suppliers_produce_products.case_qty,
                suppliers.supplier_name
FROM blockhouse_stores
                INNER JOIN store_produce_qty ON blockhouse_stores.store_id = store_produce_qty.store_id
                INNER JOIN suppliers_produce_products ON store_produce_qty.sup_pro_product_id = suppliers_produce_products.sup_pro_product_id
                INNER JOIN suppliers ON suppliers_produce_products.supplier_id = suppliers.supplier_id
```

END (produce_qty_to_order) SCRIPT

QUERY/REPORT 2

REPORT NAME: Incoming Produce Inventory Report

BUSINESS PROBLEMS ADDRESSED: This report shows incoming produce products. A user would want to use this report to quickly determine whether enough produce products have been ordered to fulfill their par level needs.

REPORT DESCRIPTION: used to show the incoming quantities of produce products. It displays the incoming inventory, their quantity, and cost.

1. For showing the incoming quantity of produce products for a given store.

a. View name: incoming_produce_products

b. WHERE store_id = {} and received_date IS NULL

SCREENSHOT OF QUERY RESULTS FOR STORE 5:

	store_id	order_id	order_detail_id	order_placed_date	received_date	case_qty	sup_pro_product_id	product_name	manufacturer	price_per_case	removed
1	5	95	948	2021-03-17	NULL	13	3	Bananas	Brothers Produce	0.51	NULL

Query executed successfully. COT-CIS3365-12 (13.0 RTM) COUGARNET\jrwils24 (64) Blockhouse_DB 00:00:00 1 rows

BEGIN (incoming_produce_products) SCRIPT

```
SELECT
                blockhouse_stores.store_id,
                orders.order_id,
                order_details.order_detail_id,
                orders.order_placed_date,
                orders.received_date,
                order_details.case_qty,
                suppliers_produce_products.sup_pro_product_id,
                suppliers_produce_products.product_name,
                suppliers_produce_products.manufacturer,
                suppliers_produce_products.price_per_case,
                order_details.removed
FROM order_details
                INNER JOIN orders ON orders.order_id = order_details.order_id
                INNER JOIN suppliers_produce_products ON suppliers_produce_products.sup_pro_product_id = order_details.sup_pro_product_id
                INNER JOIN blockhouse_stores ON blockhouse_stores.store_id = orders.store_id
```

END (incoming_produce_products) SCRIPT

QUERY/REPORT 3

REPORT NAME: Produce Order Errors Report

BUSINESS PROBLEMS ADDRESSED: This report shows any ordered produce products where the amounts ordered + the current quantity don't equal the par quantity. A user would want to use this report to quickly determine whether they should adjust or drop the item from the order.

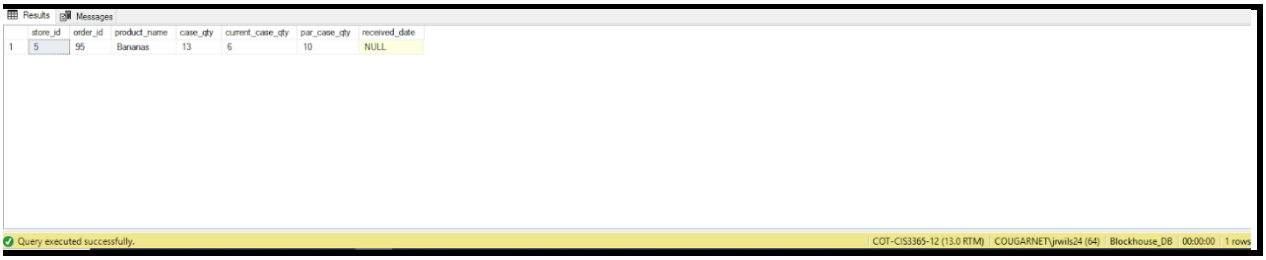
REPORT DESCRIPTION: used to verify that produce products were ordered correctly for a given store.

1. Shows produce products that were ordered incorrectly for a given store.

a. View name: incorrectly_ordered_produce_products

b. WHERE store_id = {} and case_qty + current_case_qty <> par_case_qty and received_date is null

SCREENSHOT OF QUERY RESULTS FOR STORE 5:



BEGIN (incorrectly_ordered_produce_products) SCRIPT

```
SELECT
                orders.store_id,
                orders.order_id,
                suppliers_produce_products.product_name,
                order_details.case_qty,
                store_produce_qty.current_case_qty,
                store_produce_qty.par_case_qty,
                orders.received_date
FROM order_details
                INNER JOIN orders ON order_details.order_id = orders.order_id
                INNER JOIN store_produce_qty ON store_produce_qty.sup_pro_product_id = order_details.sup_pro_product_id and store_produce_qty.store_id =
orders.store_id
                INNER JOIN suppliers_produce_products ON suppliers_produce_products.sup_pro_product_id = order_details.sup_pro_product_id
```

END (incorrectly_ordered_produce_products) SCRIPT

QUERY/REPORT 4

REPORT NAME: Incoming Retail Inventory Report

BUSINESS PROBLEMS ADDRESSED: This report shows incoming retail products. A user would want to use this report to quickly determine whether enough retail products have been ordered to fulfill their par level needs.

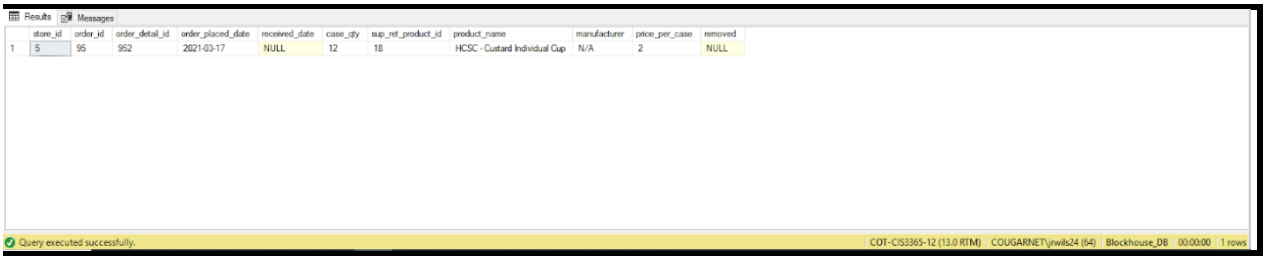
REPORT DESCRIPTION: used to show the incoming quantities of retail products. It displays the incoming inventory, their quantity, and cost.

1. For showing the incoming quantity of retail products for a given store.

a. View name: incoming_retail_products

b. WHERE store_id = {} and received_date IS NULL

SCREENSHOT OF QUERY RESULTS FOR STORE 5:



BEGIN (incoming_retail_products) SCRIPT

```
SELECT
                blockhouse_stores.store_id,
                orders.order_id,
                order_details.order_detail_id,
                orders.order_placed_date,
                orders.received_date,
                order_details.case_qty,
                suppliers_retail_products.sup_ret_product_id,
                suppliers_retail_products.product_name,
                suppliers_retail_products.manufacturer,
                suppliers_retail_products.price_per_case,
                order_details.removed
FROM order_details
                INNER JOIN orders ON orders.order_id = order_details.order_id
                INNER JOIN suppliers_retail_products ON suppliers_retail_products.sup_ret_product_id = order_details.sup_ret_product_id
                INNER JOIN blockhouse_stores ON blockhouse_stores.store_id = orders.store_id
```

END (incoming_retail_products) SCRIPT

QUERY/REPORT 5

REPORT NAME: Retail To Order Report

BUSINESS PROBLEMS ADDRESSED: This report shows which retail products need to be ordered. A user would want to use this report because it allows them to quickly determine which retail products are not currently at par levels and they can construct an order appropriately.

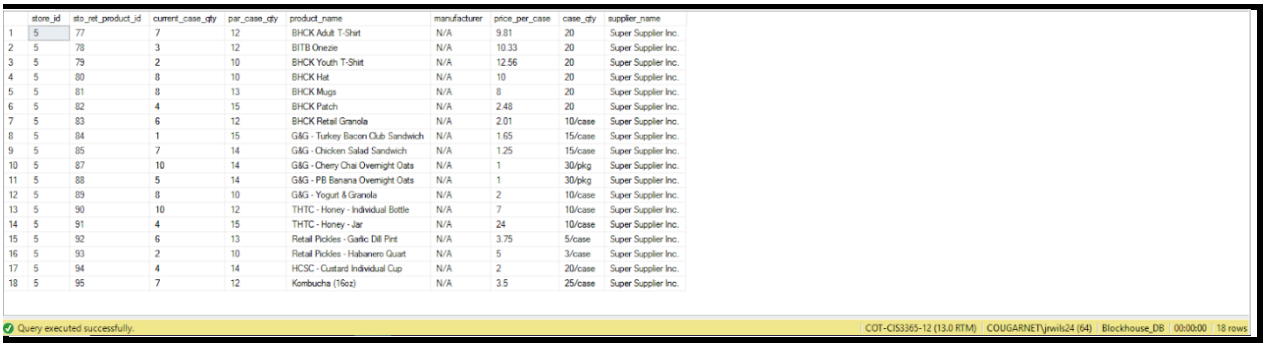
REPORT DESCRIPTION: Used to show the current produce inventory of a given store if it is less than the corresponding par level. It shows product names and their quantities.

1. For showing current retail quantity (if current less than par) of a particular store.

a. View name: retail_qty_to_order

b. WHERE store_id = {} and current_case_qty < par_case_qty

SCREENSHOT OF QUERY RESULTS FOR STORE 5:



	store_id	sto_ret_product_id	current_case_qty	par_case_qty	product_name	manufacturer	price_per_case	case_qty	supplier_name
1	5	77	7	12	BHCK Adult T-Shirt	N/A	9.81	20	Super Supplier Inc.
2	5	78	3	12	BITB Onezie	N/A	10.33	20	Super Supplier Inc.
3	5	79	2	10	BHCK Youth T-Shirt	N/A	12.56	20	Super Supplier Inc.
4	5	80	8	10	BHCK Hat	N/A	10	20	Super Supplier Inc.
5	5	81	8	13	BHCK Mugs	N/A	8	20	Super Supplier Inc.
6	5	82	4	15	BHCK Patch	N/A	2.48	20	Super Supplier Inc.
7	5	83	6	12	BHCK Retail Granola	N/A	2.01	10/case	Super Supplier Inc.
8	5	84	1	15	G&G - Turkey Bacon Club Sandwich	N/A	1.65	15/case	Super Supplier Inc.
9	5	85	7	14	G&G - Chicken Salad Sandwich	N/A	1.25	15/case	Super Supplier Inc.
10	5	87	10	14	G&G - Cherry Chai Overnight Oats	N/A	1	30/pkg	Super Supplier Inc.
11	5	88	5	14	G&G - PB Banana Overnight Oats	N/A	1	30/pkg	Super Supplier Inc.
12	5	89	8	10	G&G - Yogurt & Granola	N/A	2	10/case	Super Supplier Inc.
13	5	90	10	12	THTC - Honey - Individual Bottle	N/A	7	10/case	Super Supplier Inc.
14	5	91	4	15	THTC - Honey - Jar	N/A	24	10/case	Super Supplier Inc.
15	5	92	6	13	Retail Pickles - Garlic Dill Pint	N/A	3.75	5/case	Super Supplier Inc.
16	5	93	2	10	Retail Pickles - Habanero Quart	N/A	5	3/case	Super Supplier Inc.
17	5	94	4	14	HCSC - Custard Individual Cup	N/A	2	20/case	Super Supplier Inc.
18	5	95	7	12	Kombucha (16oz)	N/A	3.5	25/case	Super Supplier Inc.

Query executed successfully. COT-CIS3365-12 (13.0 RTM) COUGARNET\jrwils24 (64) Blockhouse_DB 00:00:00 18 rows

BEGIN (retail_qty_to_order) SCRIPT

```
SELECT
                    blockhouse_stores.store_id,
                    store_retail_qty.sto_ret_product_id,
                    store_retail_qty.current_case_qty,
                    store_retail_qty.par_case_qty,
                    suppliers_retail_products.product_name,
                    suppliers_retail_products.manufacturer,
                    suppliers_retail_products.price_per_case,
                    Suppliers_retail_products.case_qty,
                    suppliers.supplier_name
FROM blockhouse_stores
                    INNER JOIN store_retail_qty ON blockhouse_stores.store_id = store_retail_qty.store_id
                    INNER JOIN suppliers_retail_products ON store_retail_qty.sup_ret_product_id = suppliers_retail_products.sup_ret_product_id
                    INNER JOIN suppliers ON suppliers_retail_products.supplier_id = suppliers.supplier_id
```

END (retail_qty_to_order) SCRIPT

COPIES OF CREATE SCRIPTS

```
CREATE TABLE customers (
customer_id int IDENTITY(1,1) PRIMARY KEY,
first_name varchar(255) NOT NULL,
last_name varchar(255) NOT NULL,
phone varchar(12) NOT NULL,
email varchar(255) NOT NULL
);
```

```
CREATE TABLE catering_events (
event_id int IDENTITY(1,1) PRIMARY KEY,
customer_id int NOT NULL,
store_id int NOT NULL,
event_date date NOT NULL,
canceled bit
);
```

```
CREATE TABLE menu_items (
menu_item_id int IDENTITY(1,1) PRIMARY KEY,
item_name varchar(255) NOT NULL,
price float
);
```

```
CREATE TABLE catering_event_items (
event_item_id int IDENTITY(1,1) PRIMARY KEY,
event_id int NOT NULL,
menu_item_id int NOT NULL,
quantity int,
UNIQUE(event_id,menu_item_id)
);
```

```
CREATE TABLE invoices_to_customers (
invoice_id int IDENTITY(1,1) PRIMARY KEY,
event_id int NOT NULL,
amt_due float NOT NULL,
```

```
due_date date NOT NULL
);
```

```
CREATE TABLE customer_payments (
payment_id int IDENTITY(1,1) PRIMARY KEY,
invoice_id int NOT NULL,
amt_paid float NOT NULL,
payment_date date NOT NULL
);
```

COPIES OF ALTER TABLE SCRIPTS

```
ALTER TABLE catering_event
ADD FOREIGN KEY (customer_id) REFERENCES customers (customer_id),
ADD FOREIGN KEY (store_id) REFERENCES blockhouse_stores (store_id);
```

```
ALTER TABLE catering_event_items
ADD FOREIGN KEY (event_id) REFERENCES catering_events (event_id),
ADD FOREIGN KEY (menu_item_id) REFERENCES menu_items (menu_item_id);
```

```
ALTER TABLE invoices_to_customers
ADD FOREIGN KEY (event_id) REFERENCES catering_event (event_id);
```

```
ALTER TABLE customer_payments
ADD FOREIGN KEY (invoice_id) REFERENCES invoices_to_customers (invoice_id);
```

COPIES OF BULK INSERT SCRIPTS

```
BULK INSERT invoices_to_customers
FROM 'C:\invoices_to_customers.txt'
WITH (
	FIELDTERMINATOR = ',',
	ROWTERMINATOR = '\n',
	KEEPNULLS
);
```

```
BULK INSERT catering_event
FROM 'C:\catering_event.txt'
WITH (
	FIELDTERMINATOR = ',',
	ROWTERMINATOR = '\n',
	KEEPNULLS
);
```

```
BULK INSERT catering_event_items
FROM 'C:\catering_event_items.txt'
WITH (
	FIELDTERMINATOR = ',',
	ROWTERMINATOR = '\n',
	KEEPNULLS
);
```

```
BULK INSERT menu_items
FROM 'C:\menu_items.txt'
WITH (
	FIELDTERMINATOR = ',',
	ROWTERMINATOR = '\n',
	KEEPNULLS
);
```

```
BULK INSERT invoices_to_customers
FROM 'C:\invoices_to_customers.txt'
WITH (
	FIELDTERMINATOR = ',',
	ROWTERMINATOR = '\n',
	KEEPNULLS
);
```

```
BULK INSERT customer_payments
```

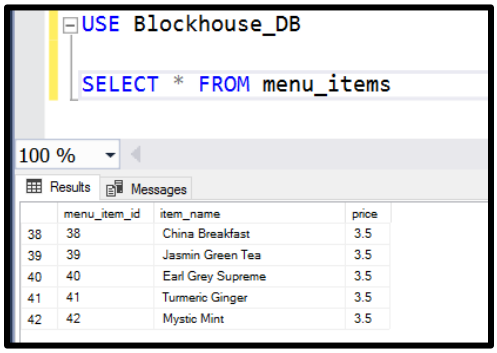
```
FROM "C:\customer_payments.txt"
WITH (
        FIELDTERMINATOR = ',',
        ROWTERMINATOR = '\n',
        KEEPNULLS
);
```

COPIES OF ALL SQL UPDATE SCRIPTS

INSERT SCRIPT #1 FOR MENU_ITEMS

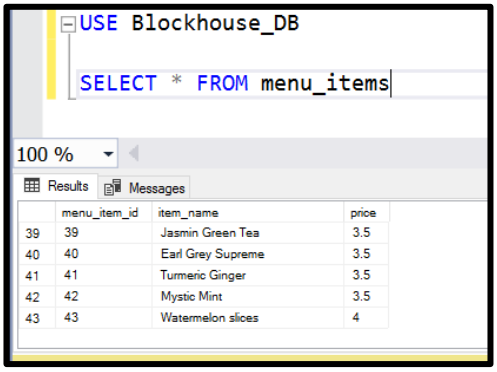
```
USE Blockhouse_DB
INSERT menu_items
VALUES('Watermelon slices','4');
```

Before:



	menu_item_id	item_name	price
38	38	China Breakfast	3.5
39	39	Jasmin Green Tea	3.5
40	40	Earl Grey Supreme	3.5
41	41	Turmeric Ginger	3.5
42	42	Mystic Mint	3.5

After:

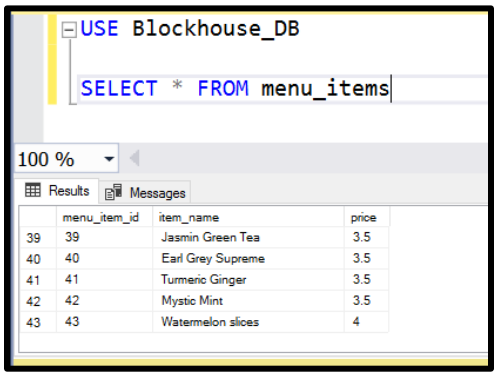


	menu_item_id	item_name	price
39	39	Jasmin Green Tea	3.5
40	40	Earl Grey Supreme	3.5
41	41	Turmeric Ginger	3.5
42	42	Mystic Mint	3.5
43	43	Watermelon slices	4

UPDATE SCRIPT #1 FOR MENU_ITEMS

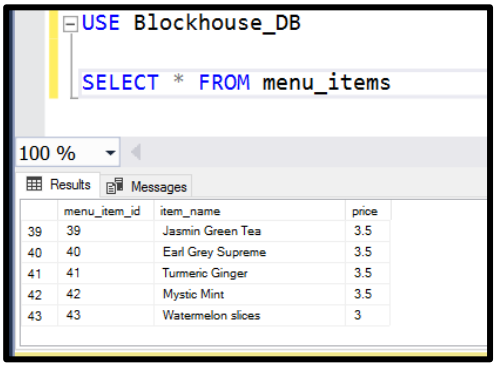
```
USE Blockhouse_DB
UPDATE menu_items
SET price =’3’
WHERE menu_item_id = 43;
```

Before:



	menu_item_id	item_name	price
39	39	Jasmin Green Tea	3.5
40	40	Earl Grey Supreme	3.5
41	41	Turmeric Ginger	3.5
42	42	Mystic Mint	3.5
43	43	Watermelon slices	4

After:

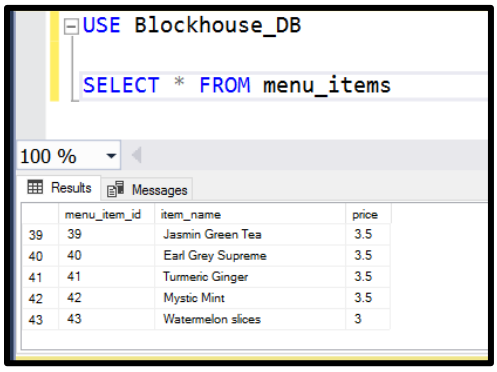


	menu_item_id	item_name	price
39	39	Jasmin Green Tea	3.5
40	40	Earl Grey Supreme	3.5
41	41	Turmeric Ginger	3.5
42	42	Mystic Mint	3.5
43	43	Watermelon slices	3

DELETE SCRIPT #1 FOR MENU_ITEMS

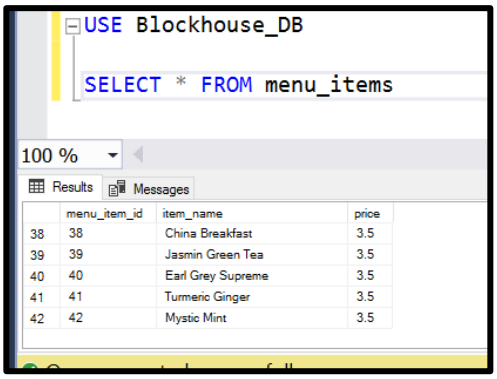
```
USE Blockhouse_DB
DELETE FROM menu_items
WHERE menu_item_id = 43;
```

Before:



	menu_item_id	item_name	price
39	39	Jasmin Green Tea	3.5
40	40	Earl Grey Supreme	3.5
41	41	Turmeric Ginger	3.5
42	42	Mystic Mint	3.5
43	43	Watermelon slices	3

After:

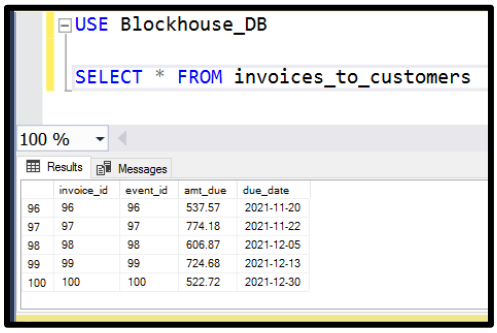


	menu_item_id	item_name	price
38	38	China Breakfast	3.5
39	39	Jasmin Green Tea	3.5
40	40	Earl Grey Supreme	3.5
41	41	Turmeric Ginger	3.5
42	42	Mystic Mint	3.5

INSERT SCRIPT #2 FOR INVOICES_TO_CUSTOMERS

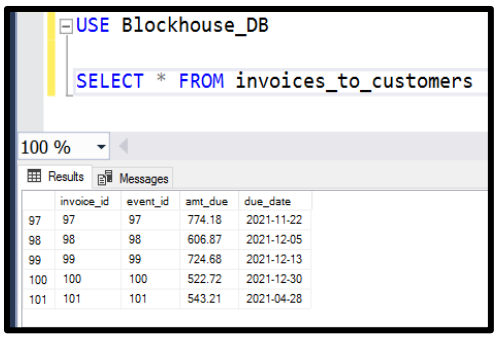
```
USE Blockhouse_DB
INSERT invoices_to_customers
VALUES('101','543.21','2021-04-28');
```

Before:



	invoice_id	event_id	amt_due	due_date
96	96	96	537.57	2021-11-20
97	97	97	774.18	2021-11-22
98	98	98	606.87	2021-12-05
99	99	99	724.68	2021-12-13
100	100	100	522.72	2021-12-30

After:

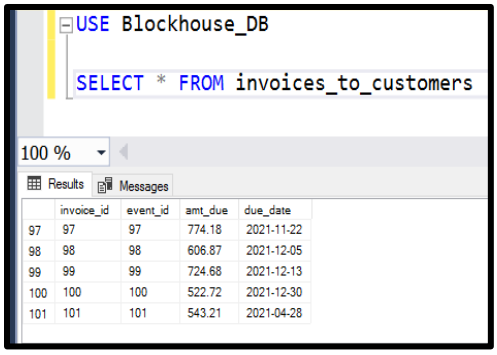


	invoice_id	event_id	amt_due	due_date
97	97	97	774.18	2021-11-22
98	98	98	606.87	2021-12-05
99	99	99	724.68	2021-12-13
100	100	100	522.72	2021-12-30
101	101	101	543.21	2021-04-28

UPDATE SCRIPT #2 FOR INVOICES_TO_CUSTOMERS

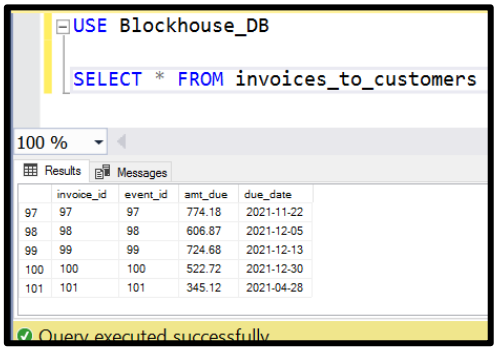
```
USE Blockhouse_DB
UPDATE invoices_to_customers
SET amt_due = '345.12'
WHERE invoice_id = 101;
```

Before:



	invoice_id	event_id	amt_due	due_date
97	97	97	774.18	2021-11-22
98	98	98	606.87	2021-12-05
99	99	99	724.68	2021-12-13
100	100	100	522.72	2021-12-30
101	101	101	543.21	2021-04-28

After:

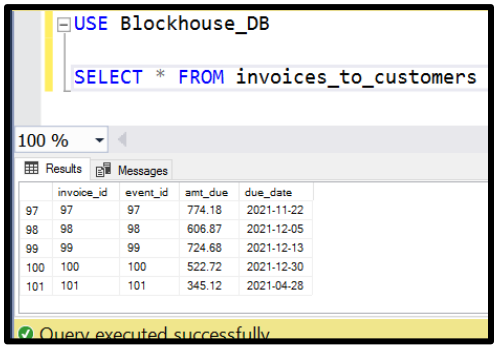


	invoice_id	event_id	amt_due	due_date
97	97	97	774.18	2021-11-22
98	98	98	606.87	2021-12-05
99	99	99	724.68	2021-12-13
100	100	100	522.72	2021-12-30
101	101	101	345.12	2021-04-28

DELETE SCRIPT #2 FOR INVOICES_TO_CUSTOMERS

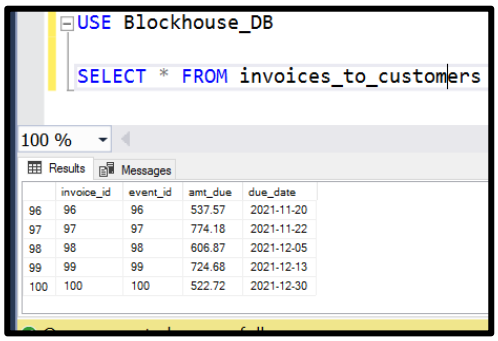
```
USE Blockhouse_DB
DELETE FROM invoices_to_customers
WHERE invoice_id = 101;
```

Before:



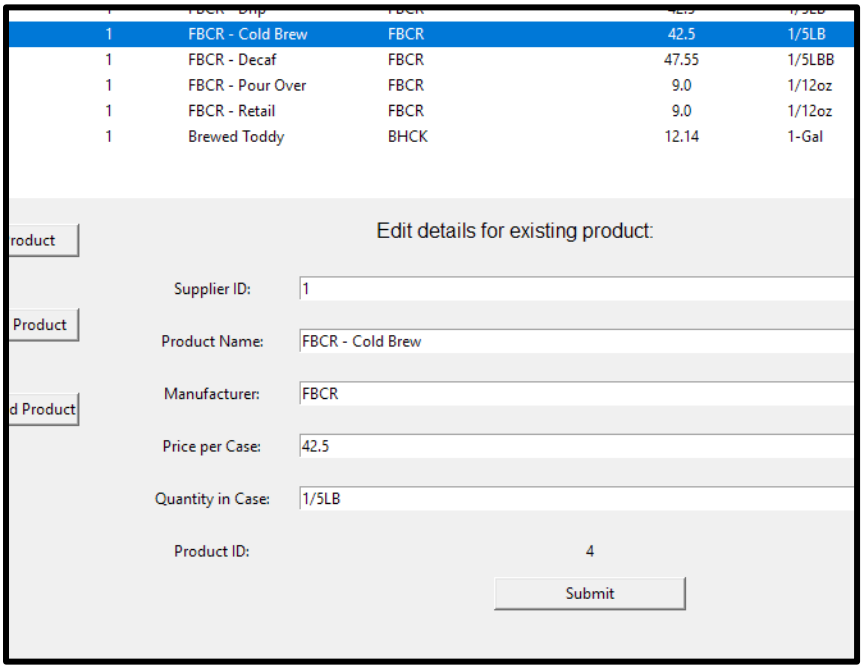
	invoice_id	event_id	amt_due	due_date
97	97	97	774.18	2021-11-22
98	98	98	606.87	2021-12-05
99	99	99	724.68	2021-12-13
100	100	100	522.72	2021-12-30
101	101	101	345.12	2021-04-28

After:

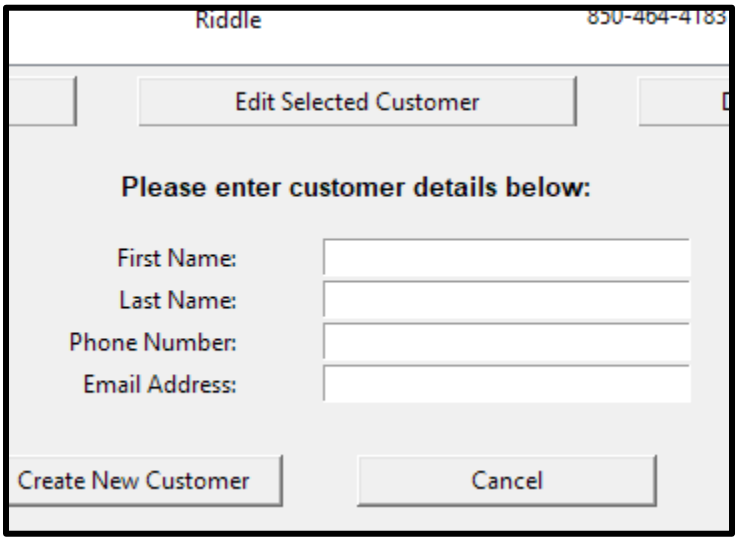


	invoice_id	event_id	amt_due	due_date
96	96	96	537.57	2021-11-20
97	97	97	774.18	2021-11-22
98	98	98	606.87	2021-12-05
99	99	99	724.68	2021-12-13
100	100	100	522.72	2021-12-30

SCREENSHOTS OF FORMS/GUIs




Edit the supplier details:
Supplier Name
Super Supplier Inc.
Phone Number
456-791-1800
Email
SuperSuppliers@gmail.com
State ID
44
County
Harris
Supplier ID
1
Submit


Riddle
850-464-4183
Edit Selected Customer
Please enter customer details below:
First Name:
Last Name:
Phone Number:
Email Address:
Create New Customer
Cancel

Jorge Vasquez Individual Contributions

QUERY/REPORT 1

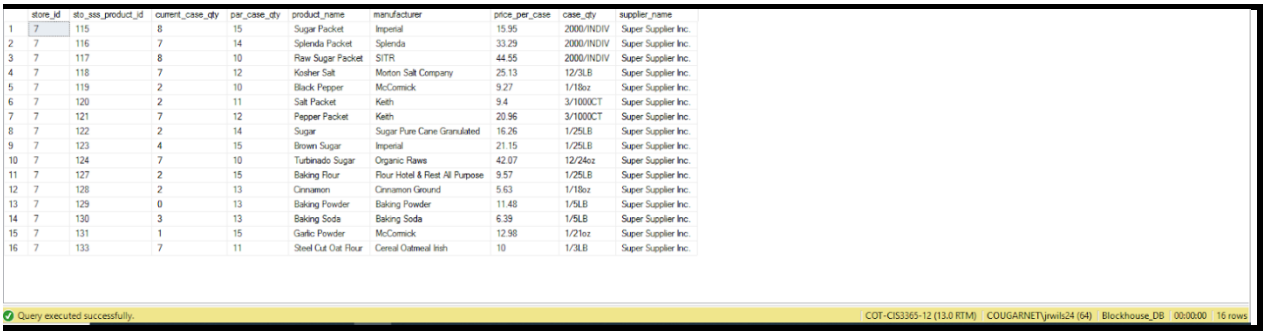
REPORT NAME: sss To Order Report

BUSINESS PROBLEMS ADDRESSED: This report shows which sugars & spices products need to be ordered. A user would want to use this report because it allows them to quickly determine which sugars & spices products are not currently at par levels and they can construct an order appropriately.

REPORT DESCRIPTION: Used to show the current sss inventory of a given store if it is less than the corresponding par level. It shows product names and their quantities.

1. For showing current sss quantity (if current less than par) of a particular store.
    a. View name: sss_qty_to_order
    b. WHERE store_id = {} and current_case_qty < par_case_qty

SCREENSHOT OF QUERY RESULTS FOR STORE 7:



	store_id	sto_sss_product_id	current_case_qty	par_case_qty	product_name	manufacturer	price_per_case	case_qty	supplier_name
1	7	115	8	15	Sugar Packet	Imperial	15.95	2000/INDIV	Super Supplier Inc.
2	7	116	7	14	Splenda Packet	Splenda	33.29	2000/INDIV	Super Supplier Inc.
3	7	117	8	10	Raw Sugar Packet	SITR	44.55	2000/INDIV	Super Supplier Inc.
4	7	118	7	12	Kosher Salt	Morton Salt Company	25.13	12/3LB	Super Supplier Inc.
5	7	119	2	10	Black Pepper	McCormick	9.27	1/18oz	Super Supplier Inc.
6	7	120	2	11	Salt Packet	Keth	9.4	3/1000CT	Super Supplier Inc.
7	7	121	7	12	Pepper Packet	Keth	20.96	3/1000CT	Super Supplier Inc.
8	7	122	2	14	Sugar	Sugar Pure Cane Granulated	16.26	1/25LB	Super Supplier Inc.
9	7	123	4	15	Brown Sugar	Imperial	21.15	1/25LB	Super Supplier Inc.
10	7	124	7	10	Turbinado Sugar	Organic Raws	42.07	12/24oz	Super Supplier Inc.
11	7	127	2	15	Baking Flour	Flour Hotel & Rest All Purpose	9.57	1/25LB	Super Supplier Inc.
12	7	128	2	13	Cinnamon	Cinnamon Ground	5.63	1/18oz	Super Supplier Inc.
13	7	129	0	13	Baking Powder	Baking Powder	11.48	1/5LB	Super Supplier Inc.
14	7	130	3	13	Baking Soda	Baking Soda	6.39	1/5LB	Super Supplier Inc.
15	7	131	1	15	Garlic Powder	McCormick	12.98	1/21oz	Super Supplier Inc.
16	7	133	7	11	Steel Cut Oat Flour	Cereal Oatmeal Irish	10	1/3LB	Super Supplier Inc.

Query executed successfully. COT-CIS3365-12 (13.0 RTM) COUGARNET\jrwils24 (64) Blockhouse_DB 00:00:00 16 rows

BEGIN (sss_qty_to_order) SCRIPT

```
SELECT
                    blockhouse_stores.store_id,
                    store_sss_qty.sto_sss_product_id,
                    store_sss_qty.current_case_qty,
                    store_sss_qty.par_case_qty,
                    suppliers_sss_products.product_name,
                    suppliers_sss_products.manufacturer,
                    suppliers_sss_products.price_per_case,
                    Suppliers_sss_products.case_qty,
                    suppliers.supplier_name
FROM blockhouse_stores
                    INNER JOIN store_sss_qty ON blockhouse_stores.store_id = store_sss_qty.store_id
                    INNER JOIN suppliers_sss_products ON store_sss_qty.sup_sss_product_id = suppliers_sss_products.sup_sss_product_id
                    INNER JOIN suppliers ON suppliers_sss_products.supplier_id = suppliers.supplier_id
```

END (sss_qty_to_order) SCRIPT

QUERY/REPORT 2

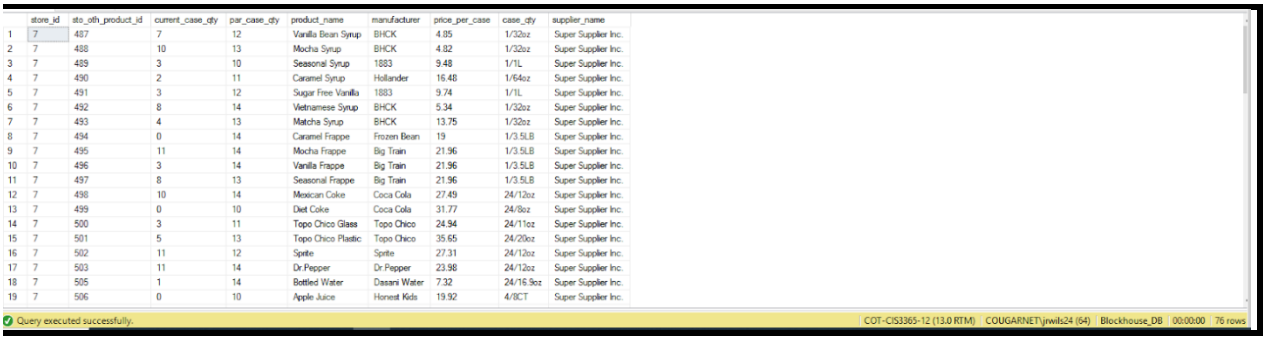
REPORT NAME: Other To Order Report

BUSINESS PROBLEMS ADDRESSED: This report shows which other products need to be ordered. A user would want to use this report because it allows them to quickly determine which other products are not currently at par levels and they can construct an order appropriately.

REPORT DESCRIPTION: Used to show the current other inventory of a given store if it is less than the corresponding par level. It shows product names and their quantities.

1. For showing current other quantity (if current less than par) of a particular store.
    a. View name: other_qty_to_order
    b. WHERE store_id = {} and current_case_qty < par_case_qty

SCREENSHOT OF QUERY RESULTS FOR STORE 7:



	store_id	sto_oth_product_id	current_case_qty	par_case_qty	product_name	manufacturer	price_per_case	case_qty	supplier_name
1	7	487	7	12	Vanilla Bean Syrup	BHCK	4.85	1/32oz	Super Supplier Inc.
2	7	488	10	13	Mocha Syrup	BHCK	4.82	1/32oz	Super Supplier Inc.
3	7	489	3	10	Seasonal Syrup	1883	9.48	1/1L	Super Supplier Inc.
4	7	490	2	11	Caramel Syrup	Hollander	16.48	1/64oz	Super Supplier Inc.
5	7	491	3	12	Sugar Free Vanilla	1883	9.74	1/1L	Super Supplier Inc.
6	7	492	8	14	Vietnamese Syrup	BHCK	5.34	1/32oz	Super Supplier Inc.
7	7	493	4	13	Matcha Syrup	BHCK	13.75	1/32oz	Super Supplier Inc.
8	7	494	0	14	Caramel Frappe	Frozen Bean	19	1/3.5LB	Super Supplier Inc.
9	7	495	11	14	Mocha Frappe	Big Train	21.96	1/3.5LB	Super Supplier Inc.
10	7	496	3	14	Vanilla Frappe	Big Train	21.96	1/3.5LB	Super Supplier Inc.
11	7	497	8	13	Seasonal Frappe	Big Train	21.96	1/3.5LB	Super Supplier Inc.
12	7	498	10	14	Mexican Coke	Coca Cola	27.49	24/12oz	Super Supplier Inc.
13	7	499	0	10	Diet Coke	Coca Cola	31.77	24/8oz	Super Supplier Inc.
14	7	500	3	11	Topo Chico Glass	Topo Chico	24.94	24/11oz	Super Supplier Inc.
15	7	501	5	13	Topo Chico Plastic	Topo Chico	35.65	24/20oz	Super Supplier Inc.
16	7	502	11	12	Sprite	Sprite	27.31	24/12oz	Super Supplier Inc.
17	7	503	11	14	Dr.Pepper	Dr.Pepper	23.98	24/12oz	Super Supplier Inc.
18	7	505	1	14	Bottled Water	Dasani Water	7.32	24/16.9oz	Super Supplier Inc.
19	7	506	0	10	Apple Juice	Honest Kids	19.92	4/8CT	Super Supplier Inc.

Query executed successfully. COT-CIS3365-12 (13.0 RTM) COUGARNET\jrwils24 (64) Blockhouse_DB 00:00:00 76 rows

BEGIN (other_qty_to_order) SCRIPT

```
SELECT
                    blockhouse_stores.store_id,
                    store_other_qty.sto_oth_product_id,
                    store_other_qty.current_case_qty,
                    store_other_qty.par_case_qty,
                    suppliers_other_products.product_name,
                    suppliers_other_products.manufacturer,
                    suppliers_other_products.price_per_case,
                    Suppliers_other_products.case_qty,
                    suppliers.supplier_name
FROM blockhouse_stores
                    INNER JOIN store_other_qty ON blockhouse_stores.store_id = store_other_qty.store_id
                    INNER JOIN suppliers_other_products ON store_other_qty.sup_oth_product_id = suppliers_other_products.sup_oth_product_id
                    INNER JOIN suppliers ON suppliers_other_products.supplier_id = suppliers.supplier_id
```

END (other_qty_to_order) SCRIPT

QUERY/REPORT 3

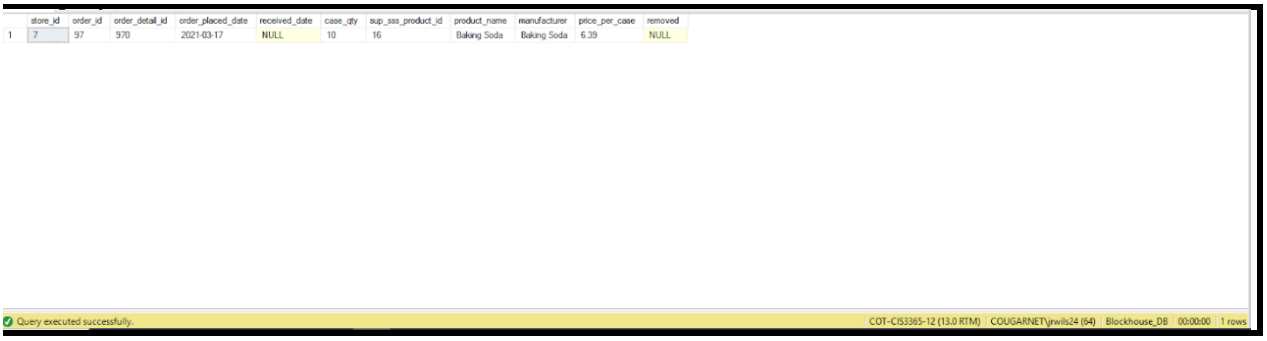
REPORT NAME: Incoming sss Inventory Report

BUSINESS PROBLEMS ADDRESSED: This report shows incoming sugars & spices products. A user would want to use this report to quickly determine whether enough sugars & spices products have been ordered to fulfill their par level needs.

REPORT DESCRIPTION: used to show the incoming quantities of sss products. It displays the incoming inventory, their quantity, and cost.

1. For showing the incoming quantity of sss products for a given store.
    a. View name: incoming_sss_products
    b. WHERE store_id = {} and received_date IS NULL

SCREENSHOT OF QUERY RESULTS FOR STORE 7:



	store_id	order_id	order_detail_id	order_placed_date	received_date	case_qty	sup_sss_product_id	product_name	manufacturer	price_per_case	removed
1	7	97	970	2021-03-17	NULL	10	16	Baking Soda	Baking Soda	6.39	NULL

BEGIN (incoming_sss_products) SCRIPT

```
SELECT
                    blockhouse_stores.store_id,
                    orders.order_id,
                    order_details.order_detail_id,
                    orders.order_placed_date,
                    orders.received_date,
                    order_details.case_qty,
                    suppliers_sss_products.sup_sss_product_id,
                    suppliers_sss_products.product_name,
                    suppliers_sss_products.manufacturer,
                    suppliers_sss_products.price_per_case,
                    order_details.removed
FROM order_details
                    INNER JOIN orders ON orders.order_id = order_details.order_id
                    INNER JOIN suppliers_sss_products ON suppliers_sss_products.sup_sss_product_id = order_details.sup_sss_product_id
                    INNER JOIN blockhouse_stores ON blockhouse_stores.store_id = orders.store_id
```

END (incoming_sss_products) SCRIPT

QUERY/REPORT 4

REPORT NAME: sss Order Errors Report

BUSINESS PROBLEMS ADDRESSED: This report shows any ordered sugars & spices products where the amounts ordered + the current quantity don't equal the par quantity. A user would want to use this report to quickly determine whether they should adjust or drop the item from the order.

REPORT DESCRIPTION: used to verify that sss products were ordered correctly for a given store.

1. Shows sss products that were ordered incorrectly for a given store.
    a. View name: incorrectly_ordered_sss_products
    b. WHERE store_id = {} and case_qty + current_case_qty <> par_case_qty and received_date is null

SCREENSHOT OF QUERY RESULTS FOR STORE 7:

	store_id	order_id	product_name	case_qty	current_case_qty	par_case_qty	received_date
1	8	98	Turbinado Sugar	10	2	10	NULL

Query executed successfully. COT-CIS3365-12 (13.0 RTM) COUGARNET\jrwils24 (64) Blockhouse_DB 00:00:00 1 rows

BEGIN (incorrectly_ordered_sss_products) SCRIPT

```
SELECT
                orders.store_id,
                orders.order_id,
                suppliers_sss_products.product_name,
                order_details.case_qty,
                store_sss_qty.current_case_qty,
                store_sss_qty.par_case_qty,
                orders.received_date
FROM order_details
                INNER JOIN orders ON order_details.order_id = orders.order_id
                INNER JOIN store_sss_qty ON store_sss_qty.sup_sss_product_id = order_details.sup_sss_product_id and store_sss_qty.store_id = orders.store_id
                INNER JOIN suppliers_sss_products ON suppliers_sss_products.sup_sss_product_id = order_details.sup_sss_product_id
```

END (incorrectly_ordered_sss_products) SCRIPT

COPIES OF CREATE SCRIPTS

```
CREATE TABLE suppliers (
supplier_id int IDENTITY(1,1) PRIMARY KEY,
supplier_name varchar(255) NOT NULL,
phone varchar(12) NOT NULL,
email varchar(255) NOT NULL,
state_id int NOT NULL,
county varchar(255) NOT NULL,
active bit
);
```

```
CREATE TABLE store_dairy_qty (
sto_dai_product_id INT IDENTITY(1,1) PRIMARY KEY,
sup_dai_product_id INT NOT NULL,
store_id INT NOT NULL,
current_case_qty INT NOT NULL,
par_case_qty INT NOT NULL,
remove BIT,
UNIQUE(sup_dai_product_id, store_id)
);
```

```
CREATE TABLE store_coffee_qty (
sto_cof_product_id INT IDENTITY(1,1) PRIMARY KEY,
sup_cof_product_id INT NOT NULL,
store_id INT NOT NULL,
current_case_qty INT NOT NULL,
par_case_qty INT NOT NULL,
remove BIT,
UNIQUE(sup_cof_product_id, store_id)
);
```

```
CREATE TABLE suppliers_dairy_products (
sup_dai_product_id int IDENTITY(1,1) PRIMARY KEY,
supplier_id int NOT NULL,
product_name varchar(255) NOT NULL,
manufacturer varchar(255) NOT NULL,
price_per_case float NOT NULL,
case_qty varchar(255),
discontinued bit
UNIQUE(supplier_id, product_name, manufacturer)
```

```
);
```

```
CREATE TABLE suppliers_coffee_products (
sup_cof_product_id int IDENTITY(1,1) PRIMARY KEY,
supplier_id int NOT NULL,
product_name varchar(255) NOT NULL,
manufacturer varchar(255) NOT NULL,
price_per_case float NOT NULL,
case_qty varchar(255),
discontinued bit,
UNIQUE(supplier_id, product_name, manufacturer)
);
```

COPIES OF ALTER TABLE SCRIPTS

```
ALTER TABLE suppliers
ADD FOREIGN KEY (state_id) REFERENCES states (state_id);
```

```
ALTER TABLE store_dairy_qty
ADD FOREIGN KEY (sup_dai_product_id) REFERENCES suppliers_dairy_products (sup_dai_product_id);
```

```
ALTER TABLE  store_coffee_qty
ADD FOREIGN KEY (sup_cof_product_id) REFERENCES suppliers_coffee_products (sup_cof_product_id);
```

```
ALTER TABLE suppliers_dairy_products
ADD FOREIGN KEY (supplier_id) REFERENCES suppliers (supplier_id);
```

```
ALTER TABLE suppliers_coffee_products
ADD FOREIGN KEY (supplier_id) REFERENCES suppliers (supplier_id);
```

COPIES OF BULK INSERT SCRIPTS

```
BULK INSERT suppliers
FROM 'C:\suppliers.txt'
WITH (
	FIELDTERMINATOR = ',',
	ROWTERMINATOR = '\n',
	KEEPNULLS
);
```

```
BULK INSERT store_dairy_qty
FROM 'C:\store_dairy_qty.txt'
WITH (
	FIELDTERMINATOR = ',',
	ROWTERMINATOR = '\n',
	KEEPNULLS
);
```

```
BULK INSERT store_coffee_qty
FROM 'C:\store_coffee_qty.txt'
WITH (
	FIELDTERMINATOR = ',',
	ROWTERMINATOR = '\n',
	KEEPNULLS
);
```

```
BULK INSERT suppliers_dairy_products
FROM 'C:\suppliers_dairy_products.txt'
WITH (
	FIELDTERMINATOR = ',',
	ROWTERMINATOR = '\n',
	KEEPNULLS
);
```

```
BULK INSERT suppliers_coffee_products
FROM 'C:\suppliers_coffee_products.txt'
WITH (
	FIELDTERMINATOR = ',',
	ROWTERMINATOR = '\n',
```

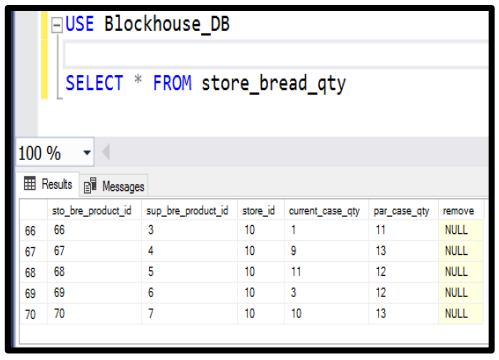
KEEPNULLS
);

COPIES OF ALL SQL UPDATE SCRIPTS

INSERT SCRIPT #1 FOR STORE_BREAD_QTY

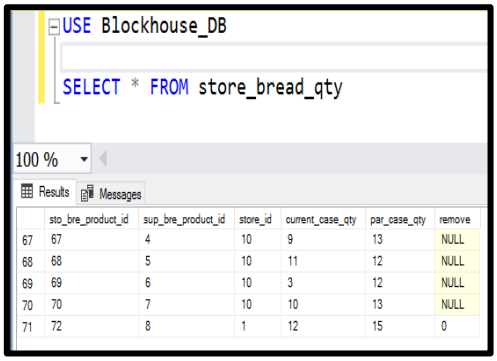
```
USE Blockhouse_DB
INSERT store_bread_qty
VALUES('1','1','12','15',' ');
```

Before:



	sto_bre_product_id	sup_bre_product_id	store_id	current_case_qty	par_case_qty	remove
66	66	3	10	1	11	NULL
67	67	4	10	9	13	NULL
68	68	5	10	11	12	NULL
69	69	6	10	3	12	NULL
70	70	7	10	10	13	NULL

After:

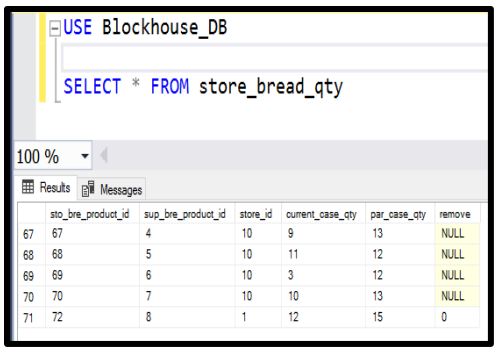


	sto_bre_product_id	sup_bre_product_id	store_id	current_case_qty	par_case_qty	remove
67	67	4	10	9	13	NULL
68	68	5	10	11	12	NULL
69	69	6	10	3	12	NULL
70	70	7	10	10	13	NULL
71	72	8	1	12	15	0

UPDATE SCRIPT #1 STORE_BREAD_QTY

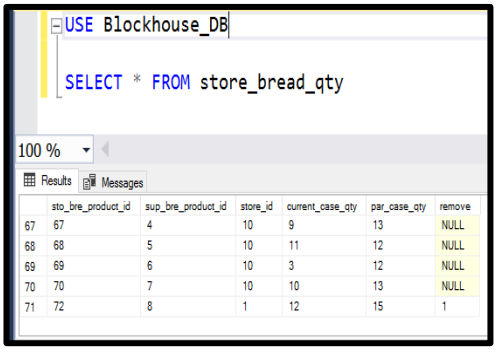
```
USE Blockhouse_DB
UPDATE store_bread_qty
SET remove = ’1’
WHERE sto_bre_product_id = 72;
```

Before:



	sto_bre_product_id	sup_bre_product_id	store_id	current_case_qty	par_case_qty	remove
67	67	4	10	9	13	NULL
68	68	5	10	11	12	NULL
69	69	6	10	3	12	NULL
70	70	7	10	10	13	NULL
71	72	8	1	12	15	0

After:

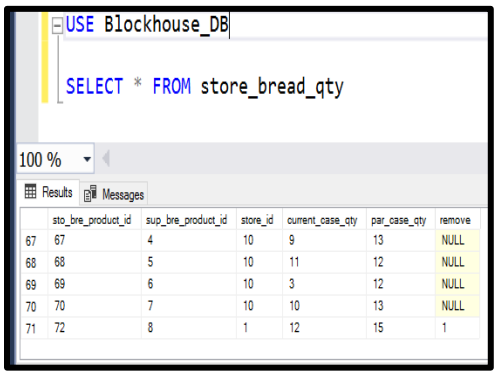


	sto_bre_product_id	sup_bre_product_id	store_id	current_case_qty	par_case_qty	remove
67	67	4	10	9	13	NULL
68	68	5	10	11	12	NULL
69	69	6	10	3	12	NULL
70	70	7	10	10	13	NULL
71	72	8	1	12	15	1

DELETE SCRIPT #1 STORE_BREAD_QTY

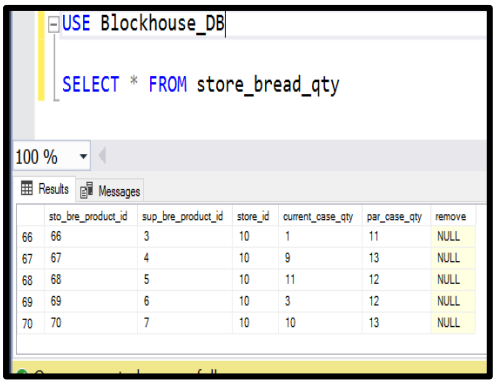
```
USE Blockhouse_DB
DELETE FROM store_bread_qty
WHERE sto_bre_product_id = 72;
```

Before:



	sto_bre_product_id	sup_bre_product_id	store_id	current_case_qty	par_case_qty	remove
67	67	4	10	9	13	NULL
68	68	5	10	11	12	NULL
69	69	6	10	3	12	NULL
70	70	7	10	10	13	NULL
71	72	8	1	12	15	1

After:

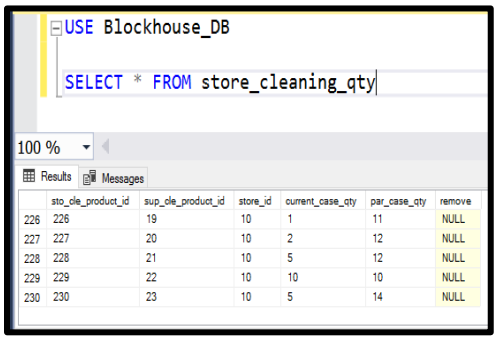


	sto_bre_product_id	sup_bre_product_id	store_id	current_case_qty	par_case_qty	remove
66	66	3	10	1	11	NULL
67	67	4	10	9	13	NULL
68	68	5	10	11	12	NULL
69	69	6	10	3	12	NULL
70	70	7	10	10	13	NULL

INSERT SCRIPT #2 FOR STORE_CLEANING_QTY

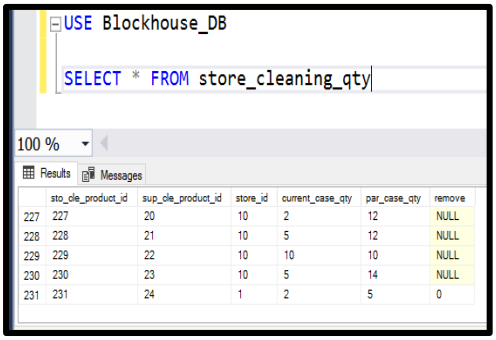
```
USE Blockhouse_DB
INSERT store_cleaning_qty
VALUES('24','1','2','5',' ');
```

Before:



	sto_cle_product_id	sup_cle_product_id	store_id	current_case_qty	par_case_qty	remove
226	226	19	10	1	11	NULL
227	227	20	10	2	12	NULL
228	228	21	10	5	12	NULL
229	229	22	10	10	10	NULL
230	230	23	10	5	14	NULL

After:

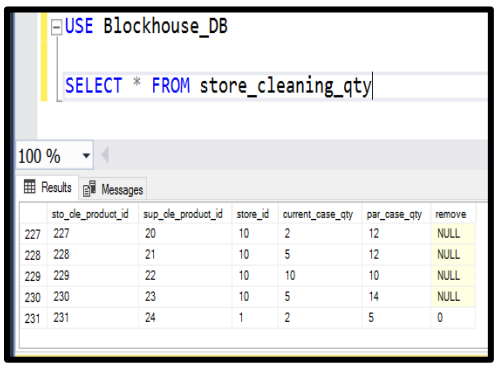


	sto_cle_product_id	sup_cle_product_id	store_id	current_case_qty	par_case_qty	remove
227	227	20	10	2	12	NULL
228	228	21	10	5	12	NULL
229	229	22	10	10	10	NULL
230	230	23	10	5	14	NULL
231	231	24	1	2	5	0

UPDATE SCRIPT #2 FOR STORE_CLEANING_QTY

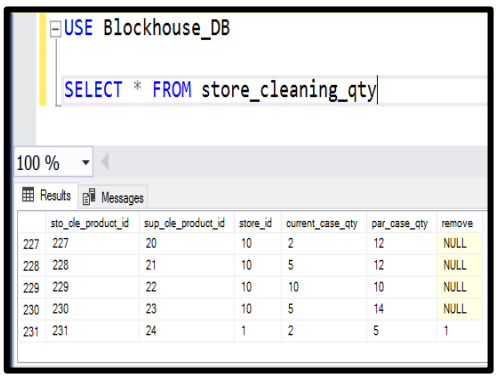
```
USE Blockhouse_DB
UPDATE store_cleaning_qty
SET remove = '1'
WHERE sto_cle_product_id = 231;
```

Before:



	sto_cle_product_id	sup_cle_product_id	store_id	current_case_qty	par_case_qty	remove
227	227	20	10	2	12	NULL
228	228	21	10	5	12	NULL
229	229	22	10	10	10	NULL
230	230	23	10	5	14	NULL
231	231	24	1	2	5	0

After:

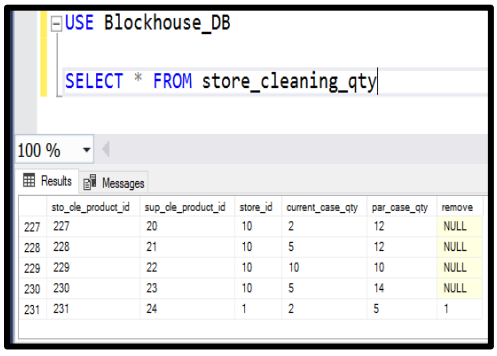


	sto_cle_product_id	sup_cle_product_id	store_id	current_case_qty	par_case_qty	remove
227	227	20	10	2	12	NULL
228	228	21	10	5	12	NULL
229	229	22	10	10	10	NULL
230	230	23	10	5	14	NULL
231	231	24	1	2	5	1

DELETE SCRIPT #2 FOR STORE_CLEANING_QTY

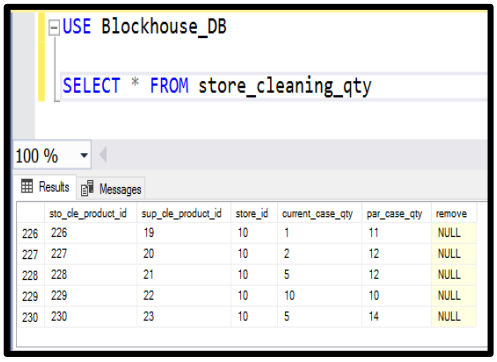
```
USE Blockhouse_DB
DELETE FROM store_cleaning_qty
WHERE sto_cle_product_id = 231;
```

Before:



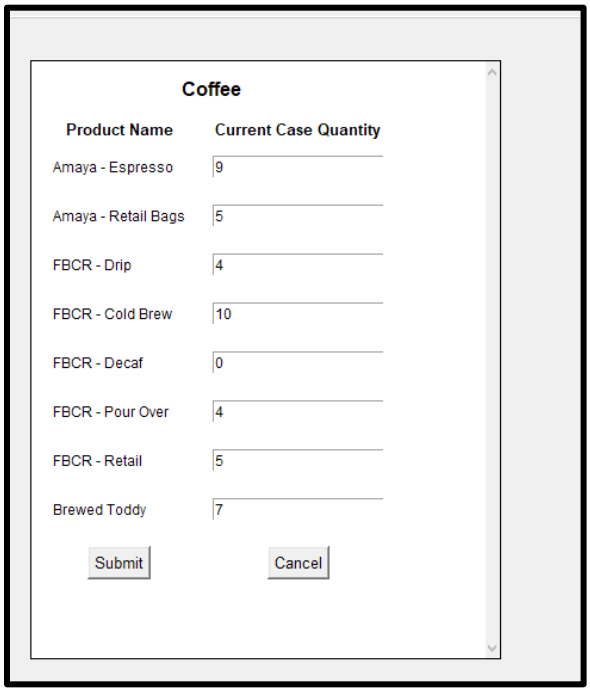
	sto_cle_product_id	sup_cle_product_id	store_id	current_case_qty	par_case_qty	remove
227	227	20	10	2	12	NULL
228	228	21	10	5	12	NULL
229	229	22	10	10	10	NULL
230	230	23	10	5	14	NULL
231	231	24	1	2	5	1

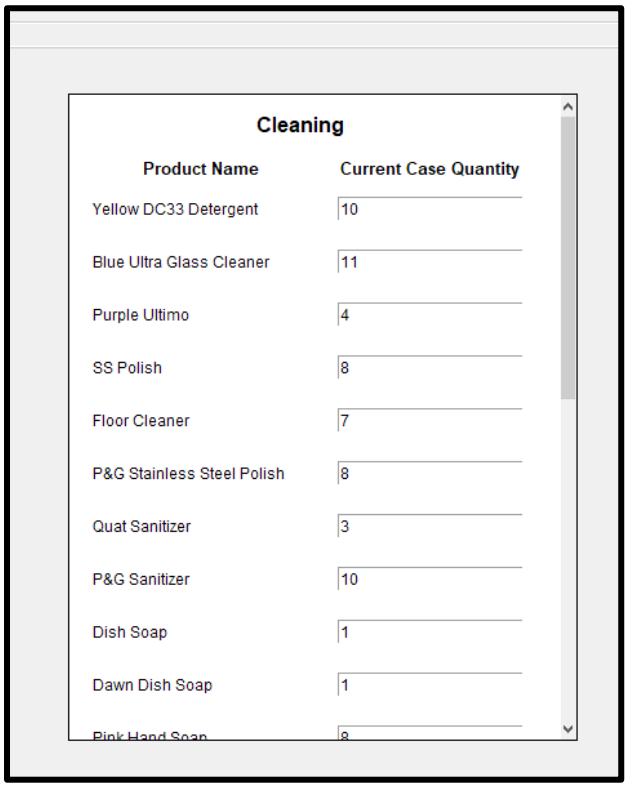
After:

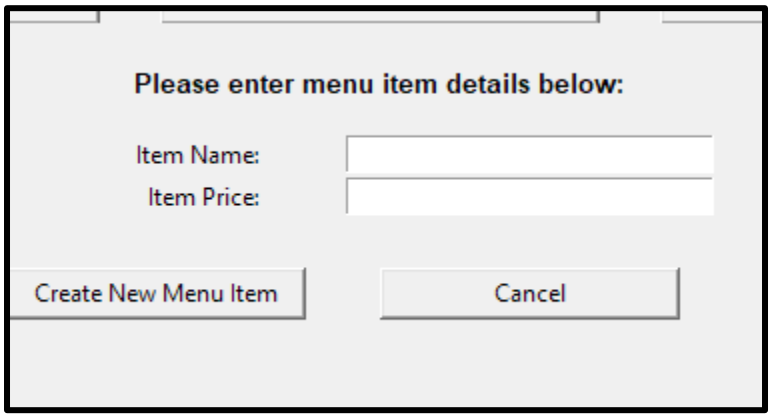


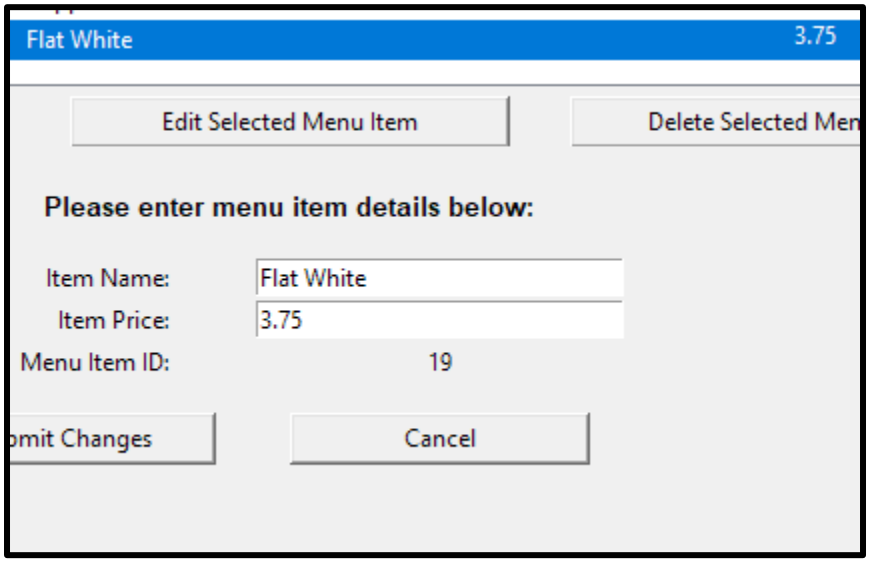
	sto_cle_product_id	sup_cle_product_id	store_id	current_case_qty	par_case_qty	remove
226	226	19	10	1	11	NULL
227	227	20	10	2	12	NULL
228	228	21	10	5	12	NULL
229	229	22	10	10	10	NULL
230	230	23	10	5	14	NULL

SCREENSHOTS OF FORMS/GUIs






Please enter menu item details below:
Item Name:
Item Price:
Create New Menu Item
Cancel


Flat White
3.75
Edit Selected Menu Item
Delete Selected Men
Please enter menu item details below:
Item Name:
Flat White
Item Price:
3.75
Menu Item ID:
19
mit Changes
Cancel

Jesse Requena Individual Contributions

QUERY/REPORT 1

REPORT NAME: Coffee To Order Report

BUSINESS PROBLEMS ADDRESSED: Coffee is the main product sold at Blockhouse, this query will help keep coffee stocked up. Coffee inventory was usually counted at the end of the week, but with this application and query, it will be possible to do a daily count of the coffee stock and order more if needed.

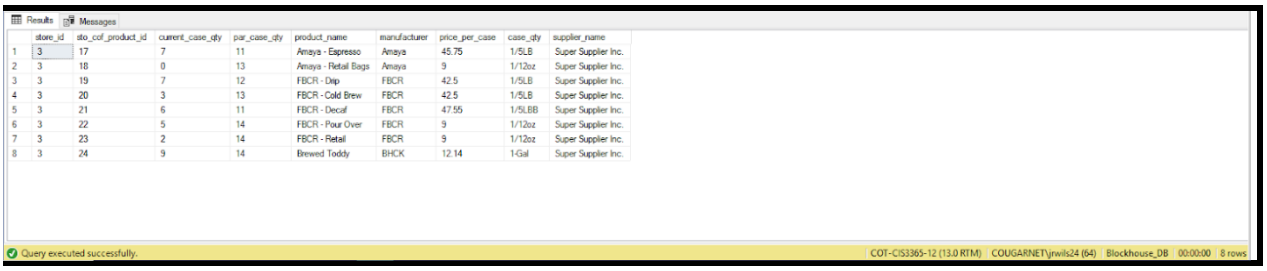
REPORT DESCRIPTION: Used to show the current Coffee inventory of a given store if it is less than the corresponding par level. It shows product names and their quantities.

1. For showing current coffee quantity (if current less than par) of a particular store.

a. View name: coffee_qty_to_order

b. WHERE store_id = {} and current_case_qty < par_case_qty

SCREENSHOT OF QUERY RESULTS FOR STORE 3:



Results Messages

	store_id	sto_cof_product_id	current_case_qty	par_case_qty	product_name	manufacturer	price_per_case	case_qty	supplier_name
1	3	17	7	11	Amaya - Espresso	Amaya	45.75	1/5LB	Super Supplier Inc.
2	3	18	0	13	Amaya - Retail Bags	Amaya	9	1/12oz	Super Supplier Inc.
3	3	19	7	12	FBCR - Drip	FBCR	42.5	1/5LB	Super Supplier Inc.
4	3	20	3	13	FBCR - Cold Brew	FBCR	42.5	1/5LB	Super Supplier Inc.
5	3	21	6	11	FBCR - Decaf	FBCR	47.55	1/5LBB	Super Supplier Inc.
6	3	22	5	14	FBCR - Pour Over	FBCR	9	1/12oz	Super Supplier Inc.
7	3	23	2	14	FBCR - Retail	FBCR	9	1/12oz	Super Supplier Inc.
8	3	24	9	14	Brewed Toddy	BHCK	12.14	1-Gal	Super Supplier Inc.

Query executed successfully. COT-CIS3365-12 (13.0 RTM) COUGARNET\jrwils24 (64) Blockhouse_DB 00:00:00 8 rows

BEGIN (coffee_qty_to_order) SCRIPT

```
SELECT
                blockhouse_stores.store_id,
                store_coffee_qty.sto_cof_product_id,
                store_coffee_qty.current_case_qty,
                store_coffee_qty.par_case_qty,
                suppliers_coffee_products.product_name,
                suppliers_coffee_products.manufacturer,
                suppliers_coffee_products.price_per_case,
                Suppliers_coffee_products.case_qty,
                suppliers.supplier_name
FROM blockhouse_stores
                INNER JOIN store_coffee_qty ON blockhouse_stores.store_id = store_coffee_qty.store_id
                INNER JOIN suppliers_coffee_products ON store_coffee_qty.sup_cof_product_id = suppliers_coffee_products.sup_cof_product_id
                INNER JOIN suppliers ON suppliers_coffee_products.supplier_id = suppliers.supplier_id
```

END (coffee_qty_to_order) SCRIPT

QUERY/REPORT 2

REPORT NAME: Incoming Coffee Inventory Report

BUSINESS PROBLEMS ADDRESSED: This coffee query will help show what coffee products are incoming and make quick adjustments if needed or if the coffee ordered needs to be increased at a moments notice.

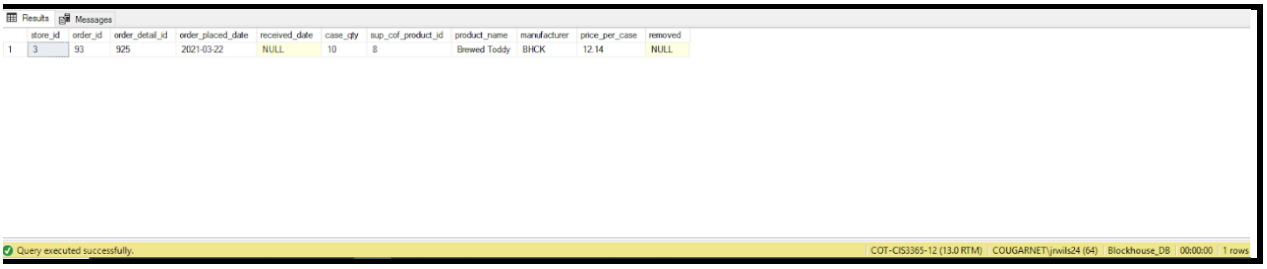
REPORT DESCRIPTION: used to show the incoming quantities of coffee products. It displays the incoming inventory, their quantity, and cost.

1. For showing the incoming quantity of coffee products for a given store.

a. View name: incoming_coffee_products

b. WHERE store_id = {} and received_date IS NULL

SCREENSHOT OF QUERY RESULTS FOR STORE 3:



Results Messages

	store_id	order_id	order_detail_id	order_placed_date	received_date	case_qty	sup_cof_product_id	product_name	manufacturer	price_per_case	removed
1	3	93	925	2021-03-22	NULL	10	8	Brewed Toddy	BHCK	12.14	NULL

Query executed successfully. COT-CIS3365-12 (13.0 RTM) COUGARNET\jrwils24 (64) Blockhouse_DB 00:00:00 1 rows

BEGIN (incoming_coffee_products) SCRIPT

```
SELECT
                    blockhouse_stores.store_id,
                    orders.order_id,
                    order_details.order_detail_id,
                    orders.order_placed_date,
                    orders.received_date,
                    order_details.case_qty,
                    suppliers_coffee_products.sup_cof_product_id,
                    suppliers_coffee_products.product_name,
                    suppliers_coffee_products.manufacturer,
                    suppliers_coffee_products.price_per_case,
                    order_details.removed
FROM order_details
                    INNER JOIN orders ON orders.order_id = order_details.order_id
                    INNER JOIN suppliers_coffee_products ON suppliers_coffee_products.sup_cof_product_id = order_details.sup_cof_product_id
                    INNER JOIN blockhouse_stores ON blockhouse_stores.store_id = orders.store_id
```

END (incoming_coffee_products) SCRIPT

QUERY/REPORT 3

REPORT NAME: Coffee Order Errors Report

BUSINESS PROBLEMS ADDRESSED: This query provides a much needed check on any errors in ordering coffee. As a coffee shop, telling the customer that there is no more of a type of coffee is not ideal. Being able to correct such a mistake will help keep customer satisfaction high and inventory stocked up.

REPORT DESCRIPTION: used to verify that Coffee products were ordered correctly for a given store.

1. Shows coffee products that were ordered incorrectly for a given store.

a. View name: incorrectly_ordered_coffee_products

b. WHERE store_id = {} and case_qty + current_case_qty <> par_case_qty and received_date is null

SCREENSHOT OF QUERY RESULTS FOR STORE 3:

BEGIN (incorrectly_ordered_coffee_products) SCRIPT

```
SELECT
                orders.store_id,
                orders.order_id,
                suppliers_coffee_products.product_name,
                order_details.case_qty,
                store_coffee_qty.current_case_qty,
                store_coffee_qty.par_case_qty,
                orders.received_date
FROM order_details
                INNER JOIN orders ON order_details.order_id = orders.order_id
                INNER JOIN store_coffee_qty ON store_coffee_qty.sup_cof_product_id = order_details.sup_cof_product_id and store_coffee_qty.store_id =
orders.store_id
                INNER JOIN suppliers_coffee_products ON suppliers_coffee_products.sup_cof_product_id = order_details.sup_cof_product_id
```

END (incorrectly_ordered_coffee_products) SCRIPT

QUERY/REPORT 4

REPORT NAME: Cleaning To Order Report

BUSINESS PROBLEMS ADDRESSED: Keeping a store clean is a top priority. A clean establishment will keep bringing customers back and content. A stocked up cleaning closet will help provide this service easily.

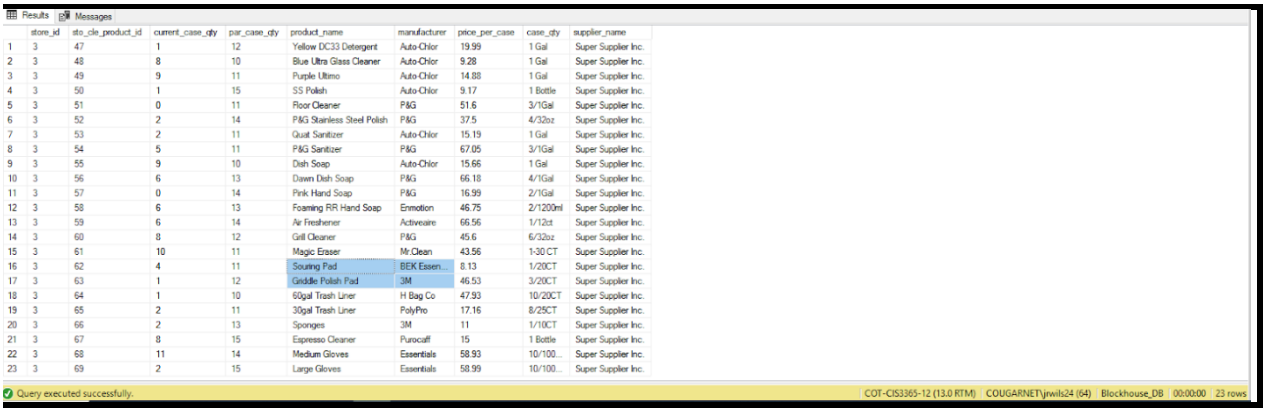
REPORT DESCRIPTION: Used to show the current cleaning inventory of a given store if it is less than the corresponding par level. It shows product names and their quantities.

1. For showing current cleaning quantity (if current less than par) of a particular store.

a. View name: cleaning_qty_to_order

b. WHERE store_id = {} and current_case_qty < par_case_qty

SCREENSHOT OF QUERY RESULTS FOR STORE 3:



Results | Messages

	store_id	sto_cle_product_id	current_case_qty	par_case_qty	product_name	manufacturer	price_per_case	case_qty	supplier_name
1	3	47	1	12	Yellow DC33 Detergent	Auto-Chlor	19.99	1 Gal	Super Supplier Inc.
2	3	48	8	10	Blue Ultra Glass Cleaner	Auto-Chlor	9.28	1 Gal	Super Supplier Inc.
3	3	49	9	11	Purple Ultimo	Auto-Chlor	14.88	1 Gal	Super Supplier Inc.
4	3	50	1	15	SS Polish	Auto-Chlor	9.17	1 Bottle	Super Supplier Inc.
5	3	51	0	11	Floor Cleaner	P&G	51.6	3/1Gal	Super Supplier Inc.
6	3	52	2	14	P&G Stainless Steel Polish	P&G	37.5	4/32oz	Super Supplier Inc.
7	3	53	2	11	Quat Sanitizer	Auto-Chlor	15.19	1 Gal	Super Supplier Inc.
8	3	54	5	11	P&G Sanitizer	P&G	67.05	3/1Gal	Super Supplier Inc.
9	3	55	9	10	Dish Soap	Auto-Chlor	15.66	1 Gal	Super Supplier Inc.
10	3	56	6	13	Dawn Dish Soap	P&G	66.18	4/1Gal	Super Supplier Inc.
11	3	57	0	14	Pink Hand Soap	P&G	16.99	2/1Gal	Super Supplier Inc.
12	3	58	6	13	Foaming RR Hand Soap	Enmotion	46.75	2/1200ml	Super Supplier Inc.
13	3	59	6	14	Air Freshener	Activeaire	66.56	1/12ct	Super Supplier Inc.
14	3	60	8	12	Grill Cleaner	P&G	45.6	6/32oz	Super Supplier Inc.
15	3	61	10	11	Magic Eraser	Mr.Clean	43.56	1-30 CT	Super Supplier Inc.
16	3	62	4	11	Souring Pad	BEK Essen...	8.13	1/20CT	Super Supplier Inc.
17	3	63	1	12	Griddle Polish Pad	3M	46.53	3/20CT	Super Supplier Inc.
18	3	64	1	10	60gal Trash Liner	H Bag Co	47.93	10/20CT	Super Supplier Inc.
19	3	65	2	11	30gal Trash Liner	PolyPro	17.16	8/25CT	Super Supplier Inc.
20	3	66	2	13	Sponges	3M	11	1/10CT	Super Supplier Inc.
21	3	67	8	15	Espresso Cleaner	Purocaff	15	1 Bottle	Super Supplier Inc.
22	3	68	11	14	Medium Gloves	Essentials	58.93	10/100...	Super Supplier Inc.
23	3	69	2	15	Large Gloves	Essentials	58.99	10/100...	Super Supplier Inc.

Query executed successfully. | COT-CIS3365-12 (13.0 RTM) | COUGARNET\jrwils24 (64) | Blockhouse_DB | 00:00:00 | 23 rows

BEGIN (cleaning_qty_to_order) SCRIPT

```
SELECT
                    blockhouse_stores.store_id,
                    store_cleaning_qty.sto_cle_product_id,
                    store_cleaning_qty.current_case_qty,
                    store_cleaning_qty.par_case_qty,
                    suppliers_cleaning_products.product_name,
                    suppliers_cleaning_products.manufacturer,
                    suppliers_cleaning_products.price_per_case,
                    Suppliers_cleaning_products.case_qty,
                    suppliers.supplier_name
FROM blockhouse_stores
                    INNER JOIN store_cleaning_qty ON blockhouse_stores.store_id = store_cleaning_qty.store_id
                    INNER JOIN suppliers_cleaning_products ON store_cleaning_qty.sup_cle_product_id = suppliers_cleaning_products.sup_cle_product_id
                    INNER JOIN suppliers ON suppliers_cleaning_products.supplier_id = suppliers.supplier_id
```

END (cleaning_qty_to_order) SCRIPT

COPIES OF CREATE SCRIPTS

```
CREATE TABLE store_meat_qty (
sto_mea_product_id INT IDENTITY(1,1) PRIMARY KEY,
sup_mea_product_id INT NOT NULL,
store_id INT NOT NULL,
current_case_qty INT NOT NULL,
par_case_qty INT NOT NULL,
remove BIT,
UNIQUE(sup_mea_product_id, store_id)
);
```

```
CREATE TABLE store_bread_qty (
sto_bre_product_id INT IDENTITY(1,1) PRIMARY KEY,
sup_bre_product_id INT NOT NULL,
store_id INT NOT NULL,
current_case_qty INT NOT NULL,
par_case_qty INT NOT NULL,
remove BIT,
UNIQUE(sup_bre_product_id, store_id)
);
```

```
CREATE TABLE suppliers_meat_products (
sup_mea_product_id int IDENTITY(1,1) PRIMARY KEY,
supplier_id int NOT NULL,
product_name varchar(255) NOT NULL,
manufacturer varchar(255) NOT NULL,
price_per_case float NOT NULL,
case_qty varchar(255),
discontinued bit,
UNIQUE(supplier_id, product_name, manufacturer));
```

```
CREATE TABLE suppliers_bread_products (
sup_bre_product_id int IDENTITY(1,1) PRIMARY KEY,
supplier_id int NOT NULL,
product_name varchar(255) NOT NULL,
manufacturer varchar(255) NOT NULL,
price_per_case float NOT NULL,
case_qty varchar(255),
discontinued bit,
UNIQUE(supplier_id, product_name, manufacturer)
);
```

COPIES OF ALTER TABLE SCRIPTS

```
ALTER TABLE store_meat_qty
ADD FOREIGN KEY (sup_mea_product_id) REFERENCES suppliers_meat_products (sup_mea_product_id);
```

```
ALTER TABLE  store_bread_qty
ADD FOREIGN KEY (sup_bre_product_id) REFERENCES suppliers_bread_products (sup_bre_product_id);
```

```
ALTER TABLE suppliers_meat_products
ADD FOREIGN KEY (supplier_id) REFERENCES suppliers (supplier_id);
```

```
ALTER TABLE suppliers_bread_products
ADD FOREIGN KEY (supplier_id) REFERENCES suppliers (supplier_id);
```

COPIES OF BULK INSERT SCRIPTS

```
BULK INSERT store_meat_qty
FROM 'C:\store_meat_qty.txt'
WITH (
        FIELDTERMINATOR = ',',
        ROWTERMINATOR = '\n',
        KEEPNULLS
);
```

```
BULK INSERT store_bread_qty
FROM 'C:\store_bread_qty.txt'
WITH (
        FIELDTERMINATOR = ',',
        ROWTERMINATOR = '\n',
        KEEPNULLS
```

```
);
```

```
BULK INSERT suppliers_meat_products
FROM 'C:\suppliers_meat_products.txt'
WITH (
        FIELDTERMINATOR = ',',
        ROWTERMINATOR = '\n',
        KEEPNULLS
);
```

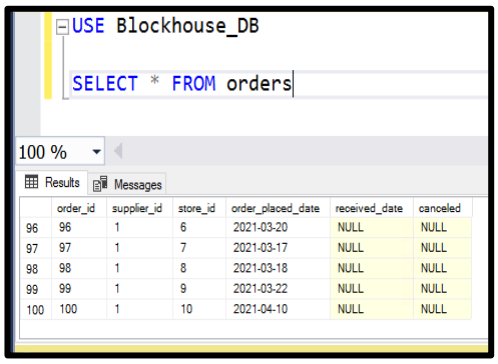
```
BULK INSERT suppliers_bread_products
FROM 'C:\suppliers_bread_products.txt'
WITH (
        FIELDTERMINATOR = ',',
        ROWTERMINATOR = '\n',
        KEEPNULLS
);
```

COPIES OF ALL SQL UPDATE SCRIPTS

INSERT SCRIPT #1 FOR ORDERS

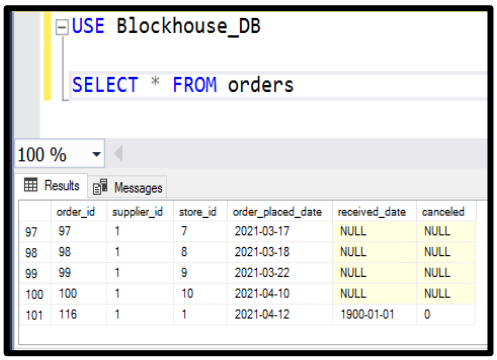
```
USE Blockhouse_DB
INSERT orders
VALUES('1','1','2021-04-12',' ',' ');
```

Before:



	order_id	supplier_id	store_id	order_placed_date	received_date	canceled
96	96	1	6	2021-03-20	NULL	NULL
97	97	1	7	2021-03-17	NULL	NULL
98	98	1	8	2021-03-18	NULL	NULL
99	99	1	9	2021-03-22	NULL	NULL
100	100	1	10	2021-04-10	NULL	NULL

After:

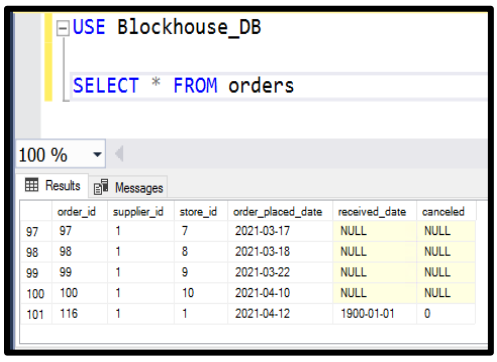


	order_id	supplier_id	store_id	order_placed_date	received_date	canceled
97	97	1	7	2021-03-17	NULL	NULL
98	98	1	8	2021-03-18	NULL	NULL
99	99	1	9	2021-03-22	NULL	NULL
100	100	1	10	2021-04-10	NULL	NULL
101	116	1	1	2021-04-12	1900-01-01	0

UPDATE SCRIPT #1 ORDERS

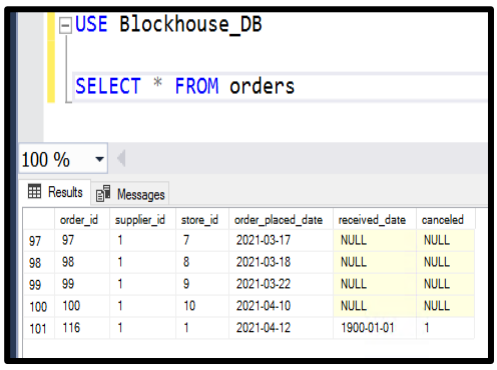
```
USE Blockhouse_DB
UPDATE orders
SET canceled = ’1’
WHERE order_id = 116;
```

Before:



	order_id	supplier_id	store_id	order_placed_date	received_date	canceled
97	97	1	7	2021-03-17	NULL	NULL
98	98	1	8	2021-03-18	NULL	NULL
99	99	1	9	2021-03-22	NULL	NULL
100	100	1	10	2021-04-10	NULL	NULL
101	116	1	1	2021-04-12	1900-01-01	0

After:

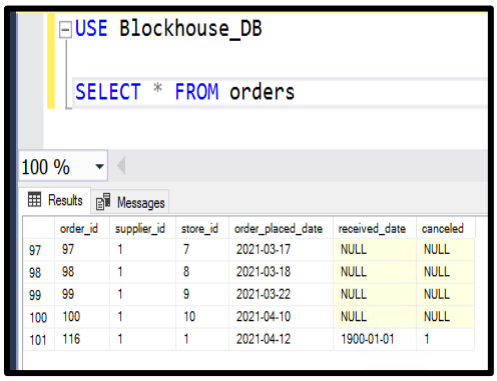


	order_id	supplier_id	store_id	order_placed_date	received_date	canceled
97	97	1	7	2021-03-17	NULL	NULL
98	98	1	8	2021-03-18	NULL	NULL
99	99	1	9	2021-03-22	NULL	NULL
100	100	1	10	2021-04-10	NULL	NULL
101	116	1	1	2021-04-12	1900-01-01	1

DELETE SCRIPT #1 ORDERS

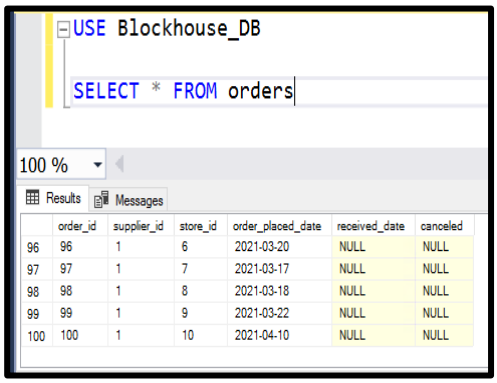
```
USE Blockhouse_DB
DELETE FROM orders
WHERE order_id = 116;
```

Before:



	order_id	supplier_id	store_id	order_placed_date	received_date	canceled
97	97	1	7	2021-03-17	NULL	NULL
98	98	1	8	2021-03-18	NULL	NULL
99	99	1	9	2021-03-22	NULL	NULL
100	100	1	10	2021-04-10	NULL	NULL
101	116	1	1	2021-04-12	1900-01-01	1

After:

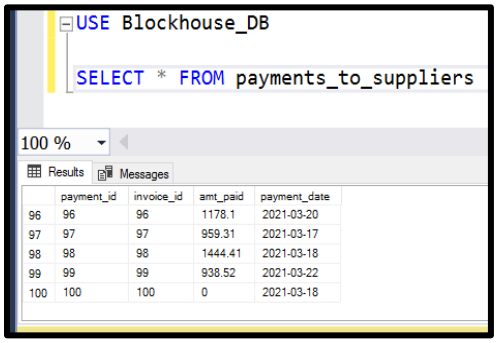


	order_id	supplier_id	store_id	order_placed_date	received_date	canceled
96	96	1	6	2021-03-20	NULL	NULL
97	97	1	7	2021-03-17	NULL	NULL
98	98	1	8	2021-03-18	NULL	NULL
99	99	1	9	2021-03-22	NULL	NULL
100	100	1	10	2021-04-10	NULL	NULL

INSERT SCRIPT #2 FOR PAYMENTS_TO_SUPPLIERS

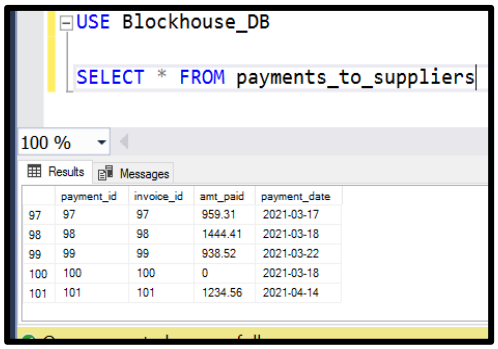
```
USE Blockhouse_DB
INSERT payments_to_suppliers
VALUES('101','1234.56','2021-04-14');
```

Before:



	payment_id	invoice_id	amt_paid	payment_date
96	96	96	1178.1	2021-03-20
97	97	97	959.31	2021-03-17
98	98	98	1444.41	2021-03-18
99	99	99	938.52	2021-03-22
100	100	100	0	2021-03-18

After:

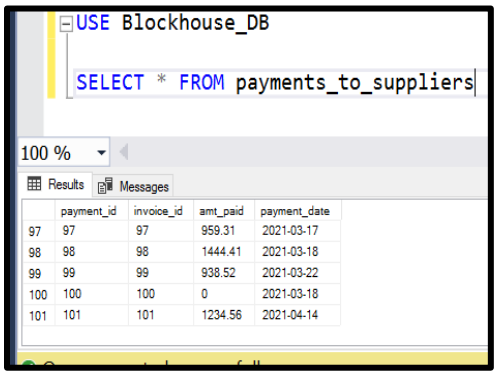


	payment_id	invoice_id	amt_paid	payment_date
97	97	97	959.31	2021-03-17
98	98	98	1444.41	2021-03-18
99	99	99	938.52	2021-03-22
100	100	100	0	2021-03-18
101	101	101	1234.56	2021-04-14

UPDATE SCRIPT #2 FOR PAYMENTS_TO_SUPPLIERS

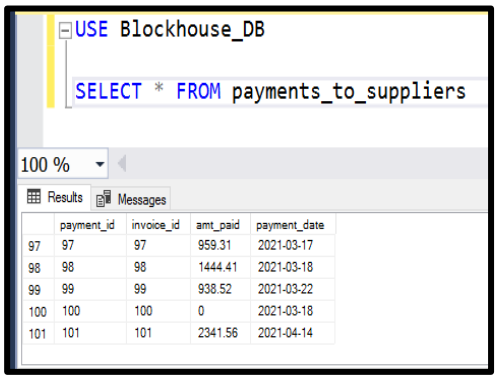
```
USE Blockhouse_DB
UPDATE payments_to_suppliers
SET amt_paid = '2341.56'
WHERE payment_id = 101;
```

Before:



	payment_id	invoice_id	amt_paid	payment_date
97	97	97	959.31	2021-03-17
98	98	98	1444.41	2021-03-18
99	99	99	938.52	2021-03-22
100	100	100	0	2021-03-18
101	101	101	1234.56	2021-04-14

After:

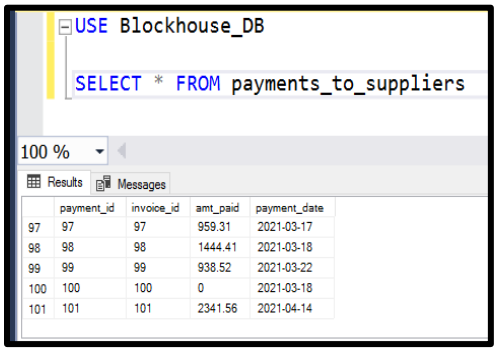


	payment_id	invoice_id	amt_paid	payment_date
97	97	97	959.31	2021-03-17
98	98	98	1444.41	2021-03-18
99	99	99	938.52	2021-03-22
100	100	100	0	2021-03-18
101	101	101	2341.56	2021-04-14

DELETE SCRIPT #2 FOR PAYMENTS_TO_SUPPLIERS

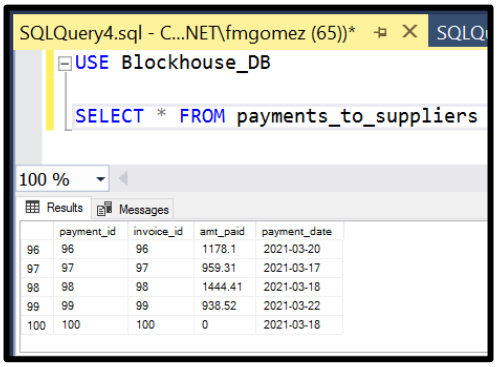
```
USE Blockhouse_DB
DELETE FROM payments_to_suppliers
WHERE payment_id = 101;
```

Before:



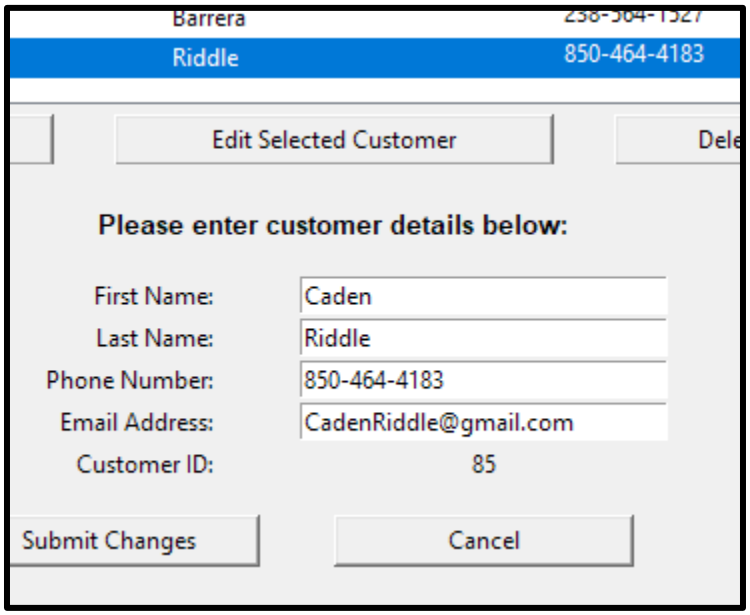
	payment_id	invoice_id	amt_paid	payment_date
97	97	97	959.31	2021-03-17
98	98	98	1444.41	2021-03-18
99	99	99	938.52	2021-03-22
100	100	100	0	2021-03-18
101	101	101	2341.56	2021-04-14

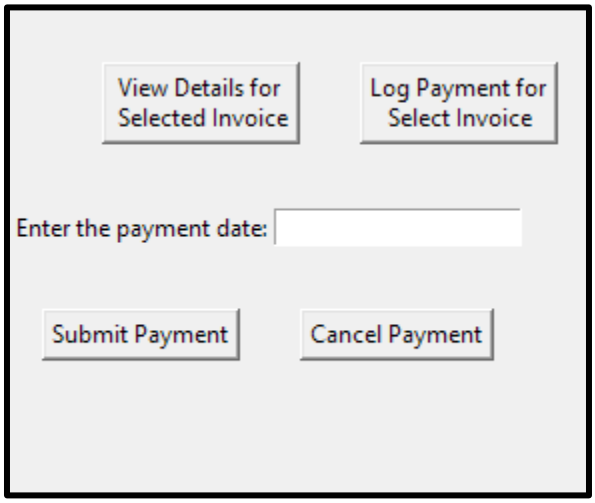
After:

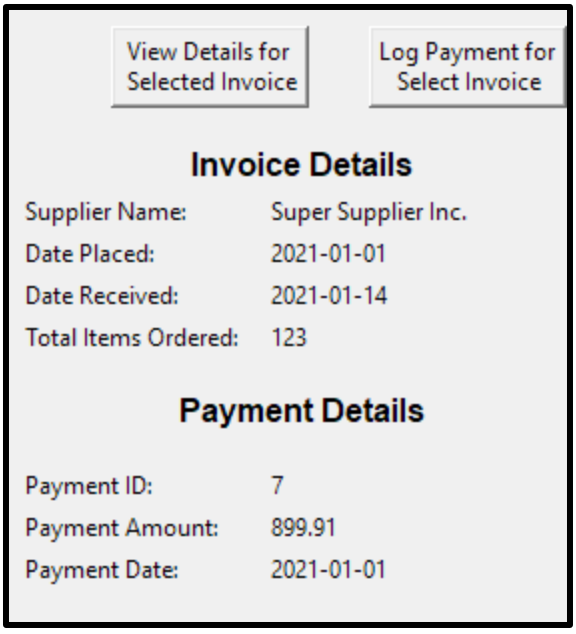


	payment_id	invoice_id	amt_paid	payment_date
96	96	96	1178.1	2021-03-20
97	97	97	959.31	2021-03-17
98	98	98	1444.41	2021-03-18
99	99	99	938.52	2021-03-22
100	100	100	0	2021-03-18

SCREENSHOTS OF FORMS/GUIs







Order ID	Store ID	Supplier ID	Date Placed	Total Due	Amount Paid
1	1	1	2021-01-06	1014.75	1014.75
2	2	1	2021-01-07	1334.52	1334.52
3	3	1	2021-01-02	816.75	816.75
4	4	1	2021-01-06	979.11	979.11
5	5	1	2021-01-03	928.62	928.62
6	6	1	2021-01-04	1188.99	1188.99
7	7	1	2021-01-01	899.91	899.91
8	8	1	2021-01-05	1082.07	1082.07
9	9	1	2021-01-03	1291.95	1291.95
10	10	1	2021-01-05	1142.46	1142.46
11	1	1	2021-01-14	986.04	986.04
12	2	1	2021-01-11	1131.57	1131.57
13	3	1	2021-01-13	1256.31	1256.31
14	4	1	2021-01-13	1462.23	1462.23
15	5	1	2021-01-11	1396.89	1396.89

Francisco Gomez Individual Contributions

QUERY/REPORT 1

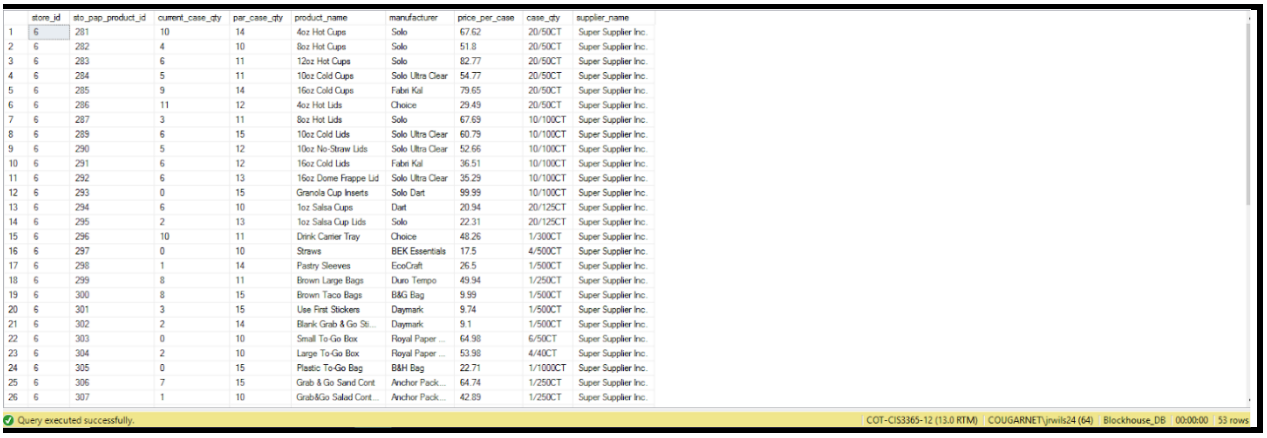
REPORT NAME: Paper To Order Report

BUSINESS PROBLEMS ADDRESSED: This report shows which paper products need to be ordered. A user would want to use this report because it allows them to quickly determine which paper products are not currently at par levels and they can construct an order appropriately.

REPORT DESCRIPTION: Used to show the current paper inventory of a given store if it is less than the corresponding par level. It shows product names and their quantities.

1. For showing current paper quantity (if current less than par) of a particular store.
    a. View name: paper_qty_to_order
    b. WHERE store_id = {} and current_case_qty < par_case_qty

SCREENSHOT OF QUERY RESULTS FOR STORE 6:



	store_id	sto_pap_product_id	current_case_qty	par_case_qty	product_name	manufacturer	price_per_case	case_qty	supplier_name
1	6	281	10	14	4oz Hot Cups	Solo	67.62	20/50CT	Super Supplier Inc.
2	6	282	4	10	8oz Hot Cups	Solo	51.8	20/50CT	Super Supplier Inc.
3	6	283	6	11	12oz Hot Cups	Solo	82.77	20/50CT	Super Supplier Inc.
4	6	284	5	11	10oz Cold Cups	Solo Ultra Clear	54.77	20/50CT	Super Supplier Inc.
5	6	285	9	14	16oz Cold Cups	Fabri Kal	79.65	20/50CT	Super Supplier Inc.
6	6	286	11	12	4oz Hot Lids	Choice	29.49	20/50CT	Super Supplier Inc.
7	6	287	3	11	8oz Hot Lids	Solo	67.69	10/100CT	Super Supplier Inc.
8	6	289	6	15	10oz Cold Lids	Solo Ultra Clear	60.79	10/100CT	Super Supplier Inc.
9	6	290	5	12	10oz No-Straw Lids	Solo Ultra Clear	52.66	10/100CT	Super Supplier Inc.
10	6	291	6	12	16oz Cold Lids	Fabri Kal	36.51	10/100CT	Super Supplier Inc.
11	6	292	6	13	16oz Dome Frappe Lid	Solo Ultra Clear	35.29	10/100CT	Super Supplier Inc.
12	6	293	0	15	Granola Cup Inserts	Solo Dart	99.99	10/100CT	Super Supplier Inc.
13	6	294	6	10	1oz Salsa Cups	Dart	20.94	20/125CT	Super Supplier Inc.
14	6	295	2	13	1oz Salsa Cup Lids	Solo	22.31	20/125CT	Super Supplier Inc.
15	6	296	10	11	Drink Carrier Tray	Choice	48.26	1/300CT	Super Supplier Inc.
16	6	297	0	10	Straws	BEK Essentials	17.5	4/500CT	Super Supplier Inc.
17	6	298	1	14	Pastry Sleeves	EcoCraft	26.5	1/500CT	Super Supplier Inc.
18	6	299	8	11	Brown Large Bags	Duro Tempo	49.94	1/250CT	Super Supplier Inc.
19	6	300	8	15	Brown Taco Bags	B&G Bag	9.99	1/500CT	Super Supplier Inc.
20	6	301	3	15	Use First Stickers	Daymark	9.74	1/500CT	Super Supplier Inc.
21	6	302	2	14	Blank Grab & Go Sti...	Daymark	9.1	1/500CT	Super Supplier Inc.
22	6	303	0	10	Small To-Go Box	Royal Paper ...	64.98	6/50CT	Super Supplier Inc.
23	6	304	2	10	Large To-Go Box	Royal Paper ...	53.98	4/40CT	Super Supplier Inc.
24	6	305	0	15	Plastic To-Go Bag	B&H Bag	22.71	1/1000CT	Super Supplier Inc.
25	6	306	7	15	Grab & Go Sand Cont	Anchor Pack...	64.74	1/250CT	Super Supplier Inc.
26	6	307	1	10	Grab&Go Salad Cont...	Anchor Pack...	42.89	1/250CT	Super Supplier Inc.

Query executed successfully. COT-CIS3365-12 (13.0 RTM) | COUGARNET\jrwils24 (64) | Blockhouse_DB | 00:00:00 | 53 rows

BEGIN (paper_qty_to_order) SCRIPT

```
SELECT
                    blockhouse_stores.store_id,
                    store_paper_qty.sto_pap_product_id,
                    store_paper_qty.current_case_qty,
                    store_paper_qty.par_case_qty,
                    suppliers_paper_products.product_name,
                    suppliers_paper_products.manufacturer,
                    suppliers_paper_products.price_per_case,
                    Suppliers_paper_products.case_qty,
                    suppliers.supplier_name
FROM blockhouse_stores
                    INNER JOIN store_paper_qty ON blockhouse_stores.store_id = store_paper_qty.store_id
                    INNER JOIN suppliers_paper_products ON store_paper_qty.sup_pap_product_id = suppliers_paper_products.sup_pap_product_id
                    INNER JOIN suppliers ON suppliers_paper_products.supplier_id = suppliers.supplier_id
```

END (paper_qty_to_order) SCRIPT

QUERY/REPORT 2

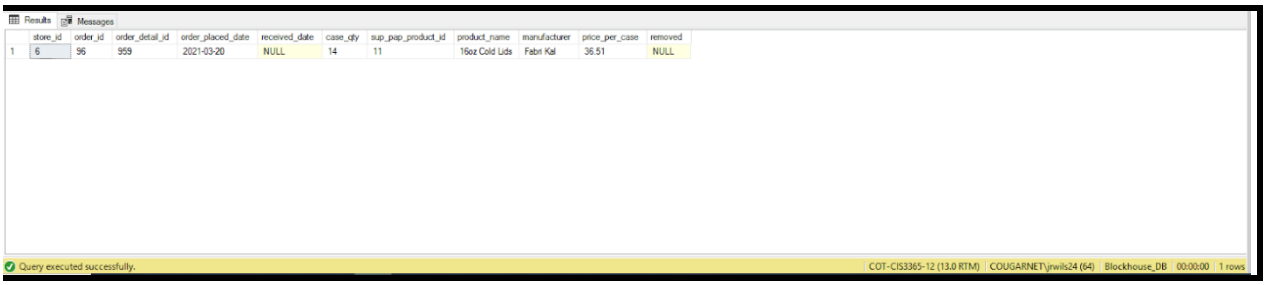
REPORT NAME: Incoming Paper Inventory Report

BUSINESS PROBLEMS ADDRESSED: This report shows incoming paper products. A user would want to use this report to quickly determine whether enough paper products have been ordered to fulfill their par level needs.

REPORT DESCRIPTION: used to show the incoming quantities of paper products. It displays the incoming inventory, their quantity, and cost.

1. For showing the incoming quantity of paper products for a given store.
    a. View name: incoming_paper_products
    b. WHERE store_id = {} and received_date IS NULL

SCREENSHOT OF QUERY RESULTS FOR STORE 6:



	store_id	order_id	order_detail_id	order_placed_date	received_date	case_qty	sup_pap_product_id	product_name	manufacturer	price_per_case	removed
1	6	96	959	2021-03-20	NULL	14	11	16oz Cold Lids	Fabri Kal	36.51	NULL

Query executed successfully. COT-CIS3365-12 (13.0 RTM) COUGARNET\jrwils24 (64) Blockhouse_DB 00:00:00 1 rows

BEGIN (incoming_paper_products) SCRIPT

```
SELECT
                    blockhouse_stores.store_id,
                    orders.order_id,
                    order_details.order_detail_id,
                    orders.order_placed_date,
                    orders.received_date,
                    order_details.case_qty,
                    suppliers_paper_products.sup_pap_product_id,
                    suppliers_paper_products.product_name,
                    suppliers_paper_products.manufacturer,
                    suppliers_paper_products.price_per_case,
                    order_details.removed
FROM order_details
                    INNER JOIN orders ON orders.order_id = order_details.order_id
                    INNER JOIN suppliers_paper_products ON suppliers_paper_products.sup_pap_product_id = order_details.sup_pap_product_id
                    INNER JOIN blockhouse_stores ON blockhouse_stores.store_id = orders.store_id
```

END (incoming_paper_products) SCRIPT

QUERY/REPORT 3

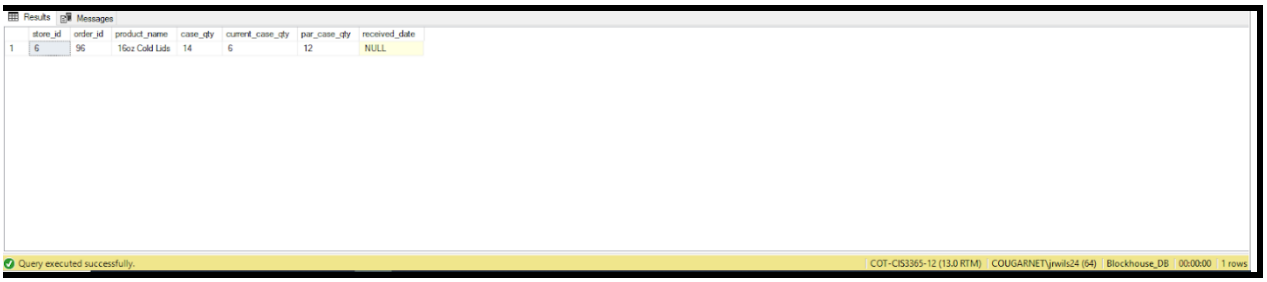
REPORT NAME: Paper Order Errors Report

BUSINESS PROBLEMS ADDRESSED: This report shows any ordered paper products where the amounts ordered + the current quantity don't equal the par quantity. A user would want to use this report to quickly determine whether they should adjust or drop the item from the order.

REPORT DESCRIPTION: used to verify that paper products were ordered correctly for a given store.

1. Shows paper products that were ordered incorrectly for a given store.
    a. View name: incorrectly_ordered_paper_products
    b. WHERE store_id = {} and case_qty + current_case_qty <> par_case_qty and received_date is null

SCREENSHOT OF QUERY RESULTS FOR STORE 6:



BEGIN (incorrectly_ordered_paper_products) SCRIPT

```
SELECT
                orders.store_id,
                orders.order_id,
                suppliers_paper_products.product_name,
                order_details.case_qty,
                store_paper_qty.current_case_qty,
                store_paper_qty.par_case_qty,
                orders.received_date
FROM order_details
                INNER JOIN orders ON order_details.order_id = orders.order_id
                INNER JOIN store_paper_qty ON store_paper_qty.sup_pap_product_id = order_details.sup_pap_product_id and store_paper_qty.store_id =
orders.store_id
                INNER JOIN suppliers_paper_products ON suppliers_paper_products.sup_pap_product_id = order_details.sup_pap_product_id
```

END (incorrectly_ordered_paper_products) SCRIPT

QUERY/REPORT 4

REPORT NAME: Other Order Errors Report

BUSINESS PROBLEMS ADDRESSED: This report shows any ordered other products where the amounts ordered + the current quantity don't equal the par quantity. A user would want to use this report to quickly determine whether they should adjust or drop the item from the order.

REPORT DESCRIPTION: used to verify that other products were ordered correctly for a given store.

1. Shows other products that were ordered incorrectly for a given store.
    a. View name: incorrectly_ordered_other_products
    b. WHERE store_id = {} and case_qty + current_case_qty <> par_case_qty and received_date is null

SCREENSHOT OF QUERY RESULTS FOR STORE 6:

BEGIN (incorrectly_ordered_other_products) SCRIPT

```
SELECT
                orders.store_id,
                orders.order_id,
                suppliers_other_products.product_name,
                order_details.case_qty,
                store_other_qty.current_case_qty,
                store_other_qty.par_case_qty,
                orders.received_date
FROM order_details
                INNER JOIN orders ON order_details.order_id = orders.order_id
                INNER JOIN store_other_qty ON store_other_qty.sup_oth_product_id = order_details.sup_oth_product_id and store_other_qty.store_id =
orders.store_id
                INNER JOIN suppliers_other_products ON suppliers_other_products.sup_oth_product_id = order_details.sup_oth_product_id
```

END (incorrectly_ordered_other_products) SCRIPT

QUERY/REPORT 5

REPORT NAME: Incoming Other Inventory Report

BUSINESS PROBLEMS ADDRESSED: This report shows incoming other products. A user would want to use this report to quickly determine whether enough other products have been ordered to fulfill their par level needs.

REPORT DESCRIPTION: used to show the incoming quantities of other products. It displays the incoming inventory, their quantity, and cost.

1. For showing the incoming quantity of other products for a given store.
    a. View name: incoming_other_products
    b. WHERE store_id = {} and received_date IS NULL

SCREENSHOT OF QUERY RESULTS FOR STORE 6:

	store_id	order_id	order_detail_id	order_placed_date	received_date	case_qty	sup_oth_product_id	product_name	manufacturer	price_per_case	removed
1	6	96	961	2021-03-20	NULL	11	28	Tumeric Ginger	Rishi	21	NULL

Query executed successfully. COT-CIS3365-12 (13.0 RTM) COUGARNET\jrwils24 (64) Blockhouse_DB 00:00:00 1 rows

BEGIN (incoming_other_products) SCRIPT

```
SELECT
                    blockhouse_stores.store_id,
                    orders.order_id,
                    order_details.order_detail_id,
                    orders.order_placed_date,
                    orders.received_date,
                    order_details.case_qty,
                    suppliers_other_products.sup_oth_product_id,
                    suppliers_other_products.product_name,
                    suppliers_other_products.manufacturer,
                    suppliers_other_products.price_per_case,
                    order_details.removed
FROM order_details
                    INNER JOIN orders ON orders.order_id = order_details.order_id
                    INNER JOIN suppliers_other_products ON suppliers_other_products.sup_oth_product_id = order_details.sup_oth_product_id
                    INNER JOIN blockhouse_stores ON blockhouse_stores.store_id = orders.store_id
```

END (incoming_other_products) SCRIPT

COPIES OF CREATE SCRIPTS

```
CREATE TABLE employees (
employee_id INT IDENTITY(1,1) PRIMARY KEY,
store_id INT NOT NULL,
user_id INT NOT NULL,
job_type_id INT NOT NULL,
terminated BIT,
first_name varchar(255),
last_name varchar(255)
);
```

```
CREATE TABLE system_users (
user_id INT IDENTITY(1,1) PRIMARY KEY,
username varchar(255) NOT NULL,
password varchar(255) NOT NULL,
admin BIT,
);
```

```
CREATE TABLE employee_type (
job_type_id int IDENTITY(1,1) PRIMARY KEY,
job_type varchar(255) NOT NULL,
job_description varchar(255) NOT NULL
);
```

```
CREATE TABLE states (
state_id INT IDENTITY(1,1) PRIMARY KEY,
state_code varchar(2) NOT NULL,
full_state_name varchar(255)
);
```

```
CREATE TABLE countries (
country_id int IDENTITY(1,1) PRIMARY KEY,
country varchar(255) NOT NULL
);
```

COPIES OF ALTER TABLE SCRIPTS

```
ALTER TABLE employees
ADD FOREIGN KEY (store_id) REFERENCES blockhosue_stores (store_id),
ADD FOREIGN KEY (user_id) REFERENCES system_users (user_id),
ADD FOREIGN KEY (job_type_id) REFERENCES employee_type (job_type_id);
```

COPIES OF BULK INSERT SCRIPTS

```
BULK INSERT employees
FROM 'C:\employees.txt'
WITH (
        FIELDTERMINATOR = ',',
        ROWTERMINATOR = '\n',
        KEEPNULLS
);
```

```
BULK INSERT system_users
FROM 'C:\system_users.txt'
WITH (
        FIELDTERMINATOR = ',',
        ROWTERMINATOR = '\n',
        KEEPNULLS
);
```

```
BULK INSERT employee_type
FROM 'C:\employee_type.txt'
WITH (
        FIELDTERMINATOR = ',',
        ROWTERMINATOR = '\n'
);
```

```
BULK INSERT system_users
FROM 'C:\states.txt'
WITH (
	FIELDTERMINATOR = ',',
	ROWTERMINATOR = '\n',
	KEEPNULLS
);
```

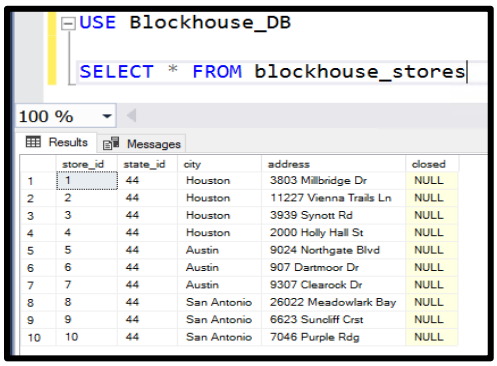
```
BULK INSERT countries
FROM 'C:\countries.txt'
WITH (
	FIELDTERMINATOR = ',',
	ROWTERMINATOR = '\n',
	KEEPNULLS
);
```

COPIES OF ALL SQL UPDATE SCRIPTS

INSERT SCRIPT #1 FOR BLOCKHOUSE_STORES

```
USE Blockhouse_DB
INSERT blockhouse_stores
VALUES('44','Pearland','123 Test st',' ');
```

Before:



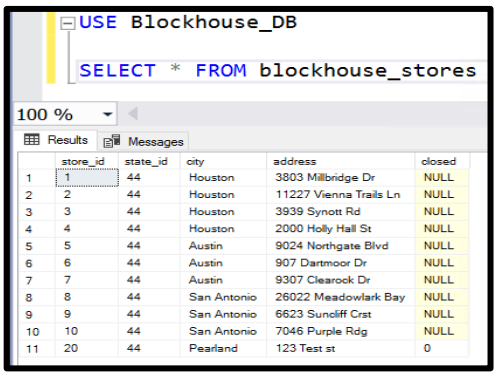
```
USE Blockhouse_DB

SELECT * FROM blockhouse_stores
```

100 %

Results | Messages

	store_id	state_id	city	address	closed
1	1	44	Houston	3803 Millbridge Dr	NULL
2	2	44	Houston	11227 Vienna Trails Ln	NULL
3	3	44	Houston	3939 Synott Rd	NULL
4	4	44	Houston	2000 Holly Hall St	NULL
5	5	44	Austin	9024 Northgate Blvd	NULL
6	6	44	Austin	907 Dartmoor Dr	NULL
7	7	44	Austin	9307 Clearock Dr	NULL
8	8	44	San Antonio	26022 Meadowlark Bay	NULL
9	9	44	San Antonio	6623 Sunciiff Crst	NULL
10	10	44	San Antonio	7046 Purple Rdg	NULL

After:



```
USE Blockhouse_DB

SELECT * FROM blockhouse_stores
```

100 %

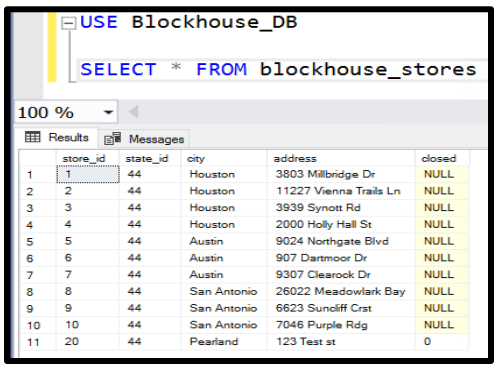
Results | Messages

	store_id	state_id	city	address	closed
1	1	44	Houston	3803 Millbridge Dr	NULL
2	2	44	Houston	11227 Vienna Trails Ln	NULL
3	3	44	Houston	3939 Synott Rd	NULL
4	4	44	Houston	2000 Holly Hall St	NULL
5	5	44	Austin	9024 Northgate Blvd	NULL
6	6	44	Austin	907 Dartmoor Dr	NULL
7	7	44	Austin	9307 Clearock Dr	NULL
8	8	44	San Antonio	26022 Meadowlark Bay	NULL
9	9	44	San Antonio	6623 Sunciiff Crst	NULL
10	10	44	San Antonio	7046 Purple Rdg	NULL
11	20	44	Pearland	123 Test st	0

UPDATE SCRIPT #1 BLOCKHOUSE_STORES

```
USE Blockhouse_DB
UPDATE blockhouse_stores
SET closed = ’1’
WHERE store_id = 20;
```

Before:



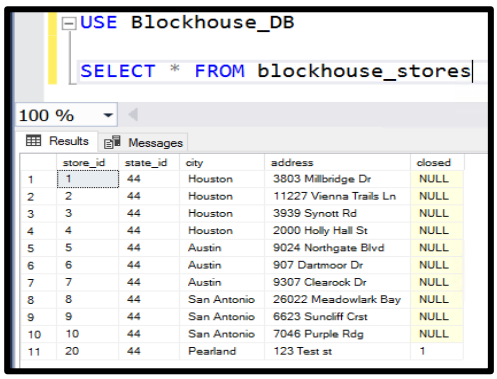
```
USE Blockhouse_DB

SELECT * FROM blockhouse_stores
```

100 %

Results | Messages

	store_id	state_id	city	address	closed
1	1	44	Houston	3803 Millbridge Dr	NULL
2	2	44	Houston	11227 Vienna Trails Ln	NULL
3	3	44	Houston	3939 Synott Rd	NULL
4	4	44	Houston	2000 Holly Hall St	NULL
5	5	44	Austin	9024 Northgate Blvd	NULL
6	6	44	Austin	907 Dartmoor Dr	NULL
7	7	44	Austin	9307 Clearock Dr	NULL
8	8	44	San Antonio	26022 Meadowlark Bay	NULL
9	9	44	San Antonio	6623 Sunciiff Crst	NULL
10	10	44	San Antonio	7046 Purple Rdg	NULL
11	20	44	Pearland	123 Test st	0

After:



```
USE Blockhouse_DB

SELECT * FROM blockhouse_stores
```

100 %

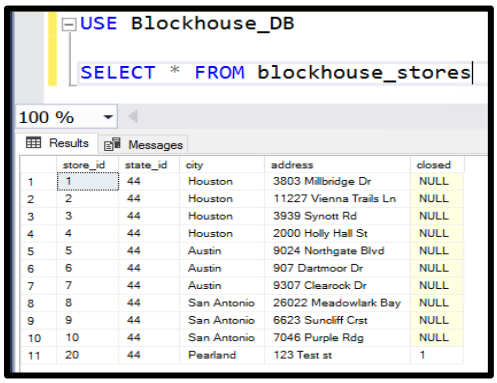
Results | Messages

	store_id	state_id	city	address	closed
1	1	44	Houston	3803 Millbridge Dr	NULL
2	2	44	Houston	11227 Vienna Trails Ln	NULL
3	3	44	Houston	3939 Synott Rd	NULL
4	4	44	Houston	2000 Holly Hall St	NULL
5	5	44	Austin	9024 Northgate Blvd	NULL
6	6	44	Austin	907 Dartmoor Dr	NULL
7	7	44	Austin	9307 Clearock Dr	NULL
8	8	44	San Antonio	26022 Meadowlark Bay	NULL
9	9	44	San Antonio	6623 Sunciiff Crst	NULL
10	10	44	San Antonio	7046 Purple Rdg	NULL
11	20	44	Pearland	123 Test st	1

DELETE SCRIPT #1 BLOCKHOUSE_STORES

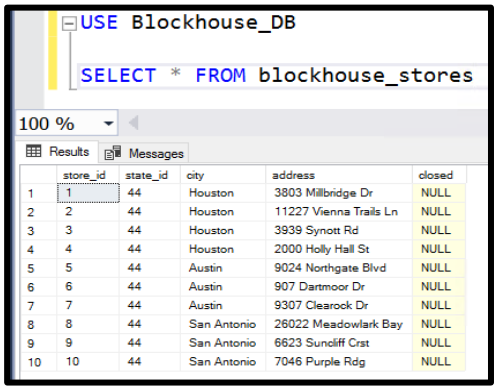
```
USE Blockhouse_DB
DELETE FROM blockhouse_stores
WHERE store_id = 20;
```

Before:



	store_id	state_id	city	address	closed
1	1	44	Houston	3803 Millbridge Dr	NULL
2	2	44	Houston	11227 Vienna Trails Ln	NULL
3	3	44	Houston	3939 Synott Rd	NULL
4	4	44	Houston	2000 Holly Hall St	NULL
5	5	44	Austin	9024 Northgate Blvd	NULL
6	6	44	Austin	907 Dartmoor Dr	NULL
7	7	44	Austin	9307 Clearock Dr	NULL
8	8	44	San Antonio	26022 Meadowlark Bay	NULL
9	9	44	San Antonio	6623 Sunoliff Crst	NULL
10	10	44	San Antonio	7046 Purple Rdg	NULL
11	20	44	Pearland	123 Test st	1

After:

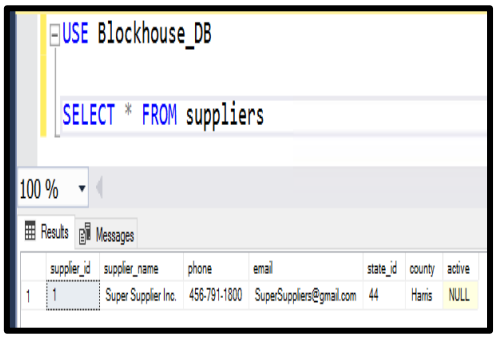


	store_id	state_id	city	address	closed
1	1	44	Houston	3803 Millbridge Dr	NULL
2	2	44	Houston	11227 Vienna Trails Ln	NULL
3	3	44	Houston	3939 Synott Rd	NULL
4	4	44	Houston	2000 Holly Hall St	NULL
5	5	44	Austin	9024 Northgate Blvd	NULL
6	6	44	Austin	907 Dartmoor Dr	NULL
7	7	44	Austin	9307 Clearock Dr	NULL
8	8	44	San Antonio	26022 Meadowlark Bay	NULL
9	9	44	San Antonio	6623 Sunoliff Crst	NULL
10	10	44	San Antonio	7046 Purple Rdg	NULL

INSERT SCRIPT #2 FOR SUPPLIERS

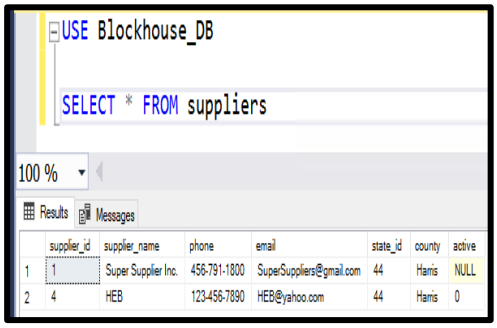
```
USE Blockhouse_DB
INSERT suppliers
VALUES('HEB','123-456-7890','HEB@yahoo.com','44','Harris','');
```

Before:



	supplier_id	supplier_name	phone	email	state_id	county	active
1	1	Super Supplier Inc.	456-791-1800	SuperSuppliers@gmail.com	44	Harris	NULL

After:

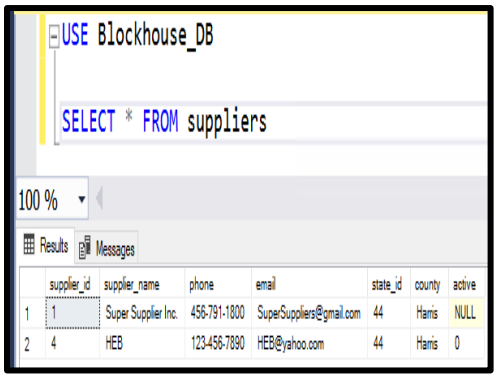


	supplier_id	supplier_name	phone	email	state_id	county	active
1	1	Super Supplier Inc.	456-791-1800	SuperSuppliers@gmail.com	44	Harris	NULL
2	4	HEB	123-456-7890	HEB@yahoo.com	44	Harris	0

UPDATE SCRIPT #2 FOR SUPPLIERS

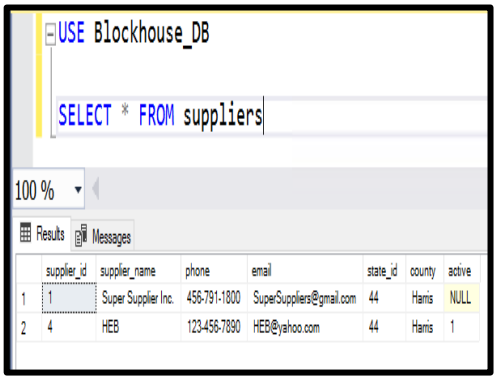
```
USE Blockhouse_DB
UPDATE suppliers
SET active = '1'
WHERE supplier_id = 4;
```

Before:



	supplier_id	supplier_name	phone	email	state_id	county	active
1	1	Super Supplier Inc.	456-791-1800	SuperSuppliers@gmail.com	44	Harris	NULL
2	4	HEB	123-456-7890	HEB@yahoo.com	44	Harris	0

After:

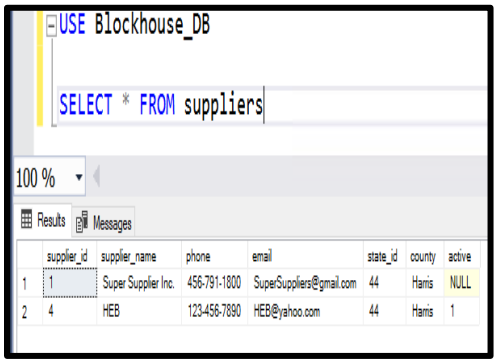


	supplier_id	supplier_name	phone	email	state_id	county	active
1	1	Super Supplier Inc.	456-791-1800	SuperSuppliers@gmail.com	44	Harris	NULL
2	4	HEB	123-456-7890	HEB@yahoo.com	44	Harris	1

DELETE SCRIPT #2 FOR SUPPLIERS

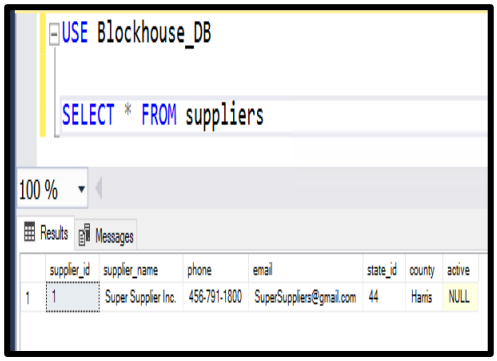
```
USE Blockhouse_DB
DELETE FROM suppliers
WHERE supplier_id = 4;
```

Before:



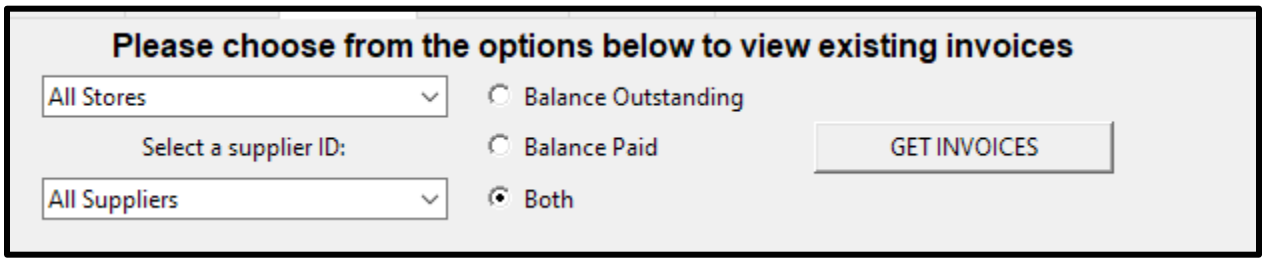
	supplier_id	supplier_name	phone	email	state_id	county	active
1	1	Super Supplier Inc.	456-791-1800	SuperSuppliers@gmail.com	44	Harris	NULL
2	4	HEB	123-456-7890	HEB@yahoo.com	44	Harris	1

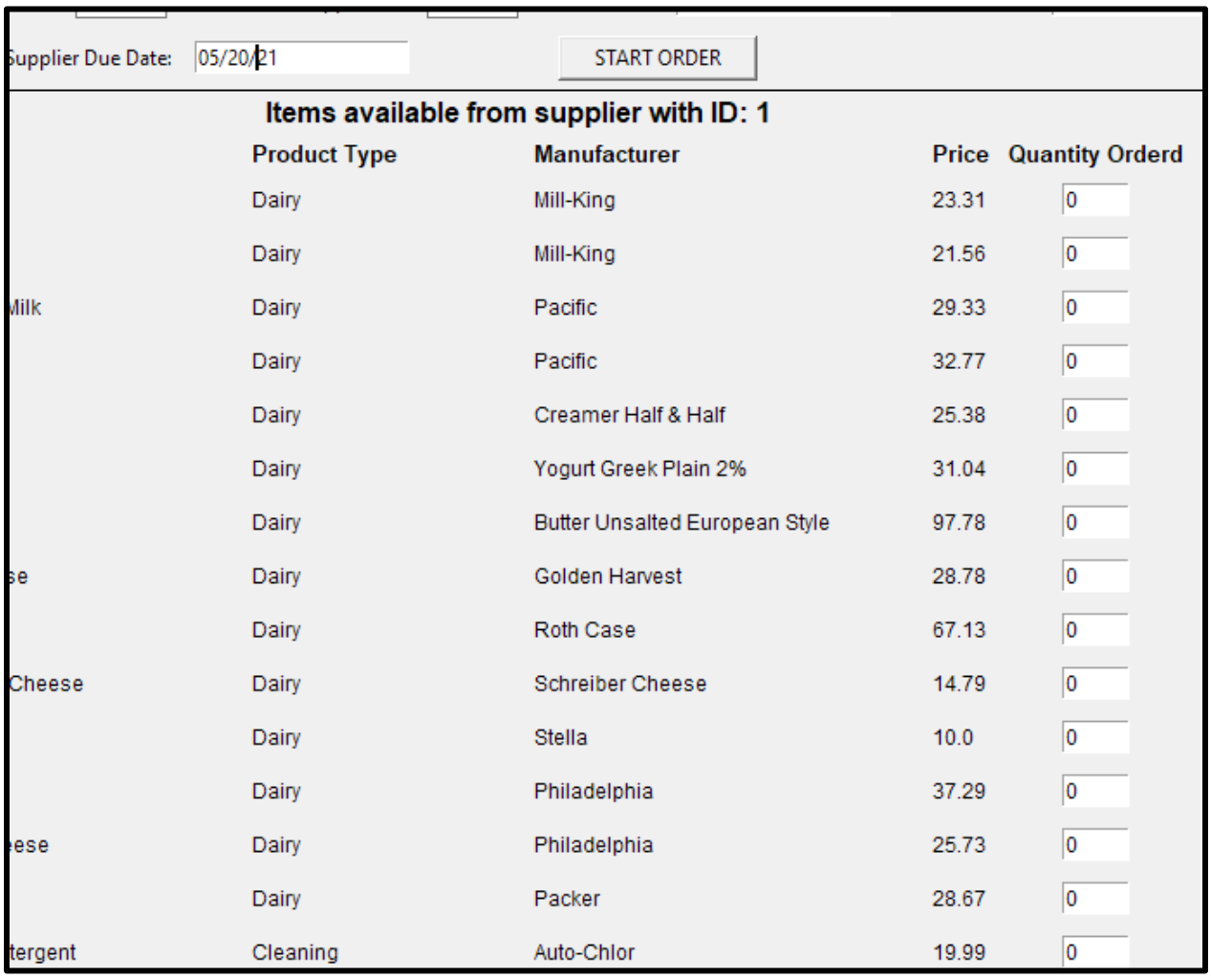
After:

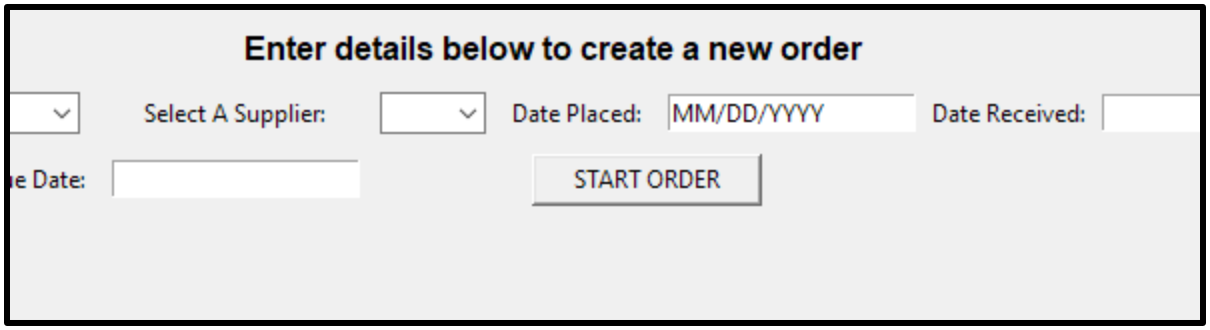


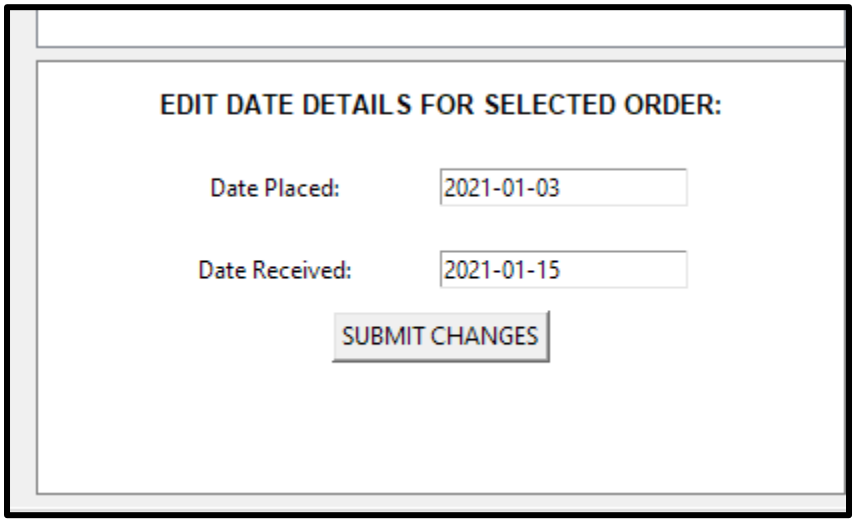
	supplier_id	supplier_name	phone	email	state_id	county	active
1	1	Super Supplier Inc.	456-791-1800	SuperSuppliers@gmail.com	44	Harris	NULL

SCREENSHOTS OF FORMS/GUIs






Enter details below to create a new order
Select A Supplier:
Date Placed:
MM/DD/YYYY
Date Received:
e Date:
START ORDER


EDIT DATE DETAILS FOR SELECTED ORDER:
Date Placed:
2021-01-03
Date Received:
2021-01-15
SUBMIT CHANGES

Andy Luong Individual Contributions

QUERY/REPORT 1

REPORT NAME: Meat To Order Report

BUSINESS PROBLEMS ADDRESSED: This query selects meat products and quantity from the store and suppliers. It then joins these attributes based on a specific id. The results show information of meat if it is less than a certain level for a given store. This would be useful for the business to use because it would allow them to see if a store has an insufficient amount of meat products available. The business would want to use this query because if they see a store with meat products below the part level, they can place an order to restock that item.

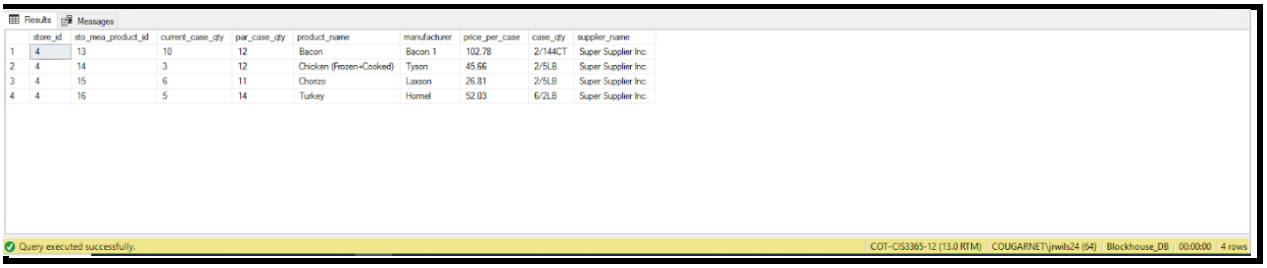
REPORT DESCRIPTION: Used to show the current meat inventory of a given store if it is less than the corresponding par level. It shows product names and their quantities.

1. For showing current meat quantity (if current less than par) of a particular store.

a. View name: meat_qty_to_order

b. WHERE store_id = {} and current_case_qty < par_case_qty

SCREENSHOT OF QUERY RESULTS FOR STORE 4:



Results | Messages

	store_id	sto_mea_product_id	current_case_qty	par_case_qty	product_name	manufacturer	price_per_case	case_qty	supplier_name
1	4	13	10	12	Bacon	Bacon 1	102.78	2/144CT	Super Supplier Inc.
2	4	14	3	12	Chicken (Frozen+Cooked)	Tyson	45.66	2/5LB	Super Supplier Inc.
3	4	15	6	11	Chorizo	Luxson	26.81	2/5LB	Super Supplier Inc.
4	4	16	5	14	Turkey	Hormel	52.03	6/2LB	Super Supplier Inc.

Query executed successfully. | COT-CIS3365-12 (13.0 RTM) | COUGARNET\jrwils24 (64) | Blockhouse_DB | 00:00:00 | 4 rows

BEGIN (meat_qty_to_order) SCRIPT

```
SELECT
                    blockhouse_stores.store_id,
                    store_meat_qty.sto_mea_product_id,
                    store_meat_qty.current_case_qty,
                    store_meat_qty.par_case_qty,
                    suppliers_meat_products.product_name,
                    suppliers_meat_products.manufacturer,
                    suppliers_meat_products.price_per_case,
                    Suppliers_meat_products.case_qty,
                    suppliers.supplier_name
FROM blockhouse_stores
                    INNER JOIN store_meat_qty ON blockhouse_stores.store_id = store_meat_qty.store_id
                    INNER JOIN suppliers_meat_products ON store_meat_qty.sup_mea_product_id = suppliers_meat_products.sup_mea_product_id
                    INNER JOIN suppliers ON suppliers_meat_products.supplier_id = suppliers.supplier_id
```

END (meat_qty_to_order) SCRIPT

QUERY/REPORT 2

REPORT NAME: Incoming meat Inventory Report

BUSINESS PROBLEMS ADDRESSED: This query selects order details and meat products and joins them together based on a specific store id. The results show incoming meat products that have been ordered for a specific store. The business would want to use this query to see an organized view of an order that has been placed for a specific item such as meat products.

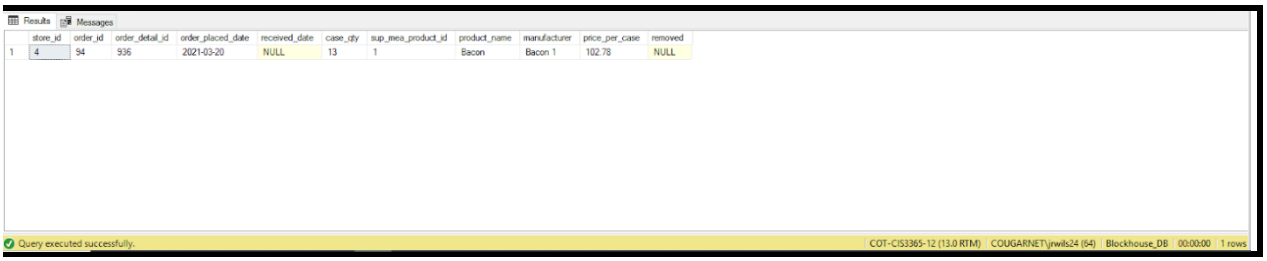
REPORT DESCRIPTION: used to show the incoming quantities of meat products. It displays the incoming inventory, their quantity, and cost.

1. For showing the incoming quantity of meat products for a given store.

a. View name: incoming_meat_products

b. WHERE store_id = {} and received_date IS NULL

SCREENSHOT OF QUERY RESULTS FOR STORE 4:



Results | Messages

	store_id	order_id	order_detail_id	order_placed_date	received_date	case_qty	sup_mea_product_id	product_name	manufacturer	price_per_case	removed
1	4	94	936	2021-03-20	NULL	13	1	Bacon	Bacon 1	102.78	NULL

Query executed successfully. COT-CIS3365-12 (13.0 RTM) COUGARNET\jrwils24 (64) Blockhouse_DB 00:00:00 1 rows

BEGIN (incoming_meat_products) SCRIPT

```
SELECT
                    blockhouse_stores.store_id,
                    orders.order_id,
                    order_details.order_detail_id,
                    orders.order_placed_date,
                    orders.received_date,
                    order_details.case_qty,
                    suppliers_meat_products.sup_mea_product_id,
                    suppliers_meat_products.product_name,
                    suppliers_meat_products.manufacturer,
                    suppliers_meat_products.price_per_case,
                    order_details.removed
FROM order_details
                    INNER JOIN orders ON orders.order_id = order_details.order_id
                    INNER JOIN suppliers_meat_products ON suppliers_meat_products.sup_mea_product_id = order_details.sup_mea_product_id
                    INNER JOIN blockhouse_stores ON blockhouse_stores.store_id = orders.store_id
```

END (incoming_meat_products) SCRIPT

QUERY/REPORT 3

REPORT NAME: Meat Order Errors Report

BUSINESS PROBLEMS ADDRESSED: This query selects order details and meat products and joins them based on store id. The results show information of incorrectly ordered meat products. This is useful for the business because they can verify if the ordered meat products ended up in the correct store with the correct quantity. The business would want to use this query because they can locate where the incorrect order was delivered so they can quickly fix that mistake.

REPORT DESCRIPTION: used to verify that meat products were ordered correctly for a given store.

1. Shows meat products that were ordered incorrectly for a given store.

a. View name: incorrectly_ordered_meat_products

b. WHERE store_id = {} and case_qty + current_case_qty <> par_case_qty and received_date is null

SCREENSHOT OF QUERY RESULTS FOR STORE 4:

	store_id	order_id	product_name	case_qty	current_case_qty	par_case_qty	received_date
1	4	94	Bacon	13	10	12	NULL

Query executed successfully. COT-CIS3365-12 (13.0 RTM) COUGARNET\jrwils24 (64) Blockhouse_DB 00:00:00 1 rows

BEGIN (incorrectly_ordered_meat_products) SCRIPT

```
SELECT
                orders.store_id,
                orders.order_id,
                suppliers_meat_products.product_name,
                order_details.case_qty,
                store_meat_qty.current_case_qty,
                store_meat_qty.par_case_qty,
                orders.received_date
FROM order_details
                INNER JOIN orders ON order_details.order_id = orders.order_id
                INNER JOIN store_meat_qty ON store_meat_qty.sup_mea_product_id = order_details.sup_mea_product_id and store_meat_qty.store_id =
orders.store_id
                INNER JOIN suppliers_meat_products ON suppliers_meat_products.sup_mea_product_id = order_details.sup_mea_product_id
```

END (incorrectly_ordered_meat_products) SCRIPT

QUERY/REPORT 4

REPORT NAME: Retail Order Errors Report

BUSINESS PROBLEMS ADDRESSED: This query selects order details and retail products and joins them based on store id. The results show information of incorrectly ordered retail products. This is useful for the business because they can verify if the ordered retail products ended up in the correct store with the correct quantity. The business would want to use this query because they can locate where the incorrect order was delivered so they can quickly fix that mistake.

REPORT DESCRIPTION: used to verify that retail products were ordered correctly for a given store.

1. Shows retail products that were ordered incorrectly for a given store.

a. View name: incorrectly_ordered_retail_products

b. WHERE store_id = {} and case_qty + current_case_qty <> par_case_qty and received_date is null

SCREENSHOT OF QUERY RESULTS FOR STORE 4:

	store_id	order_id	product_name	case_qty	current_case_qty	par_case_qty	received_date
1	4	94	THTC - Honey - Jar	13	5	13	NULL

Query executed successfully. COT-CIS3365-12 (13.0 RTM) COUGARNET\jrwils24 (64) Blockhouse_DB 00:00:00 1 rows

BEGIN (incorrectly_ordered_retail_products) SCRIPT

```
SELECT
                orders.store_id,
                orders.order_id,
                suppliers_retail_products.product_name,
                order_details.case_qty,
                store_retail_qty.current_case_qty,
                store_retail_qty.par_case_qty,
                orders.received_date
FROM order_details
                INNER JOIN orders ON order_details.order_id = orders.order_id
                INNER JOIN store_retail_qty ON store_retail_qty.sup_ret_product_id = order_details.sup_ret_product_id and store_retail_qty.store_id =
orders.store_id
                INNER JOIN suppliers_retail_products ON suppliers_retail_products.sup_ret_product_id = order_details.sup_ret_product_id
```

END (incorrectly_ordered_retail_products) SCRIPT

COPIES OF CREATE SCRIPTS

```
CREATE TABLE store_sss_qty (
sto_sss_product_id INT IDENTITY(1,1) PRIMARY KEY,
sup_sss_product_id INT NOT NULL,
store_id INT NOT NULL,
current_case_qty INT NOT NULL,
par_case_qty INT NOT NULL,
remove BIT,
UNIQUE(sup_sss_product_id, store_id)
);
```

```
CREATE TABLE store_other_qty (
sto_oth_product_id INT IDENTITY(1,1) PRIMARY KEY,
sup_oth_product_id INT NOT NULL,
store_id INT NOT NULL,
current_case_qty INT NOT NULL,
par_case_qty INT NOT NULL,
remove BIT,
UNIQUE(sup_oth_product_id, store_id)
);
```

```
CREATE TABLE suppliers_sss_products (
sup_sss_product_id int IDENTITY(1,1) PRIMARY KEY,
supplier_id int NOT NULL,
product_name varchar(255) NOT NULL,
manufacturer varchar(255) NOT NULL,
price_per_case float NOT NULL,
case_qty int varchar(255),
discontinued bit,
UNIQUE(supplier_id, product_name, manufacturer));
```

```
CREATE TABLE suppliers_other_products (
sup_oth_product_id int IDENTITY(1,1) PRIMARY KEY,
supplier_id int NOT NULL,
product_name varchar(255) NOT NULL,
manufacturer varchar(255) NOT NULL,
price_per_case float NOT NULL,
case_qty varchar(255),
discontinued bit,
UNIQUE(supplier_id, product_name, manufacturer)
);
```

COPIES OF ALTER TABLE SCRIPTS

```
ALTER TABLE store_sss_qty
ADD FOREIGN KEY (sup_sss_product_id) REFERENCES suppliers_sss_products (sup_sss_product_id);
```

```
ALTER TABLE  store_other_qty
ADD FOREIGN KEY (sup_oth_product_id) REFERENCES suppliers_other_products (sup_oth_product_id);
```

```
ALTER TABLE suppliers_sss_products
ADD FOREIGN KEY (supplier_id) REFERENCES suppliers (supplier_id);
```

```
ALTER TABLE suppliers_other_products
ADD FOREIGN KEY (supplier_id) REFERENCES suppliers (supplier_id);
```

COPIES OF BULK INSERT SCRIPTS

```
BULK INSERT store_sss_qty
FROM 'C:\store_sss_qty.txt'
WITH (
        FIELDTERMINATOR = ',',
        ROWTERMINATOR = '\n',
        KEEPNULLS
);
```

```
BULK INSERT store_other_qty
FROM 'C:\store_other_qty.txt'
```

```
WITH (
        FIELDTERMINATOR = ',',
        ROWTERMINATOR = '\n',
        KEEPNULLS
);
```

```
BULK INSERT suppliers_sss_products
FROM 'C:\suppliers_sss_products.txt'
WITH (
        FIELDTERMINATOR = ',',
        ROWTERMINATOR = '\n',
        KEEPNULLS
);
```

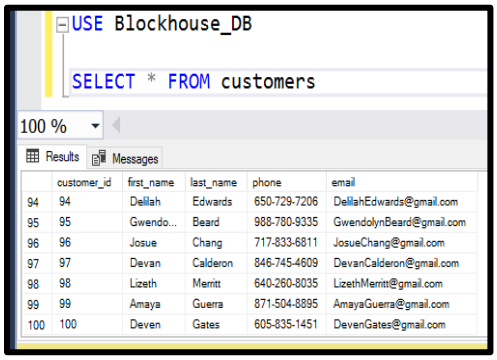
```
BULK INSERT suppliers_other_products
FROM 'C:\suppliers_other_products.txt'
WITH (
        FIELDTERMINATOR = ',',
        ROWTERMINATOR = '\n',
        KEEPNULLS
);
```

COPIES OF ALL SQL UPDATE SCRIPTS

INSERT SCRIPT #1 FOR CUSTOMERS

```
USE Blockhouse_DB
INSERT customers
VALUES('Ariel','Gonzalez','123-456-7890','ArielGonzalez@gmail.com');
```

Before:



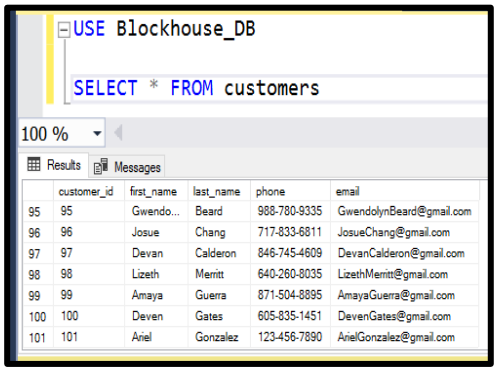
```
USE Blockhouse_DB

SELECT * FROM customers
```

100 %

Results | Messages

	customer_id	first_name	last_name	phone	email
94	94	Delilah	Edwards	650-729-7206	DelilahEdwards@gmail.com
95	95	Gwendo...	Beard	988-780-9335	GwendolynBeard@gmail.com
96	96	Josue	Chang	717-833-6811	JosueChang@gmail.com
97	97	Devan	Calderon	846-745-4609	DevanCalderon@gmail.com
98	98	Lizeth	Merritt	640-260-8035	LizethMerritt@gmail.com
99	99	Amaya	Guerra	871-504-8895	AmayaGuerra@gmail.com
100	100	Deven	Gates	605-835-1451	DevenGates@gmail.com

After:



```
USE Blockhouse_DB

SELECT * FROM customers
```

100 %

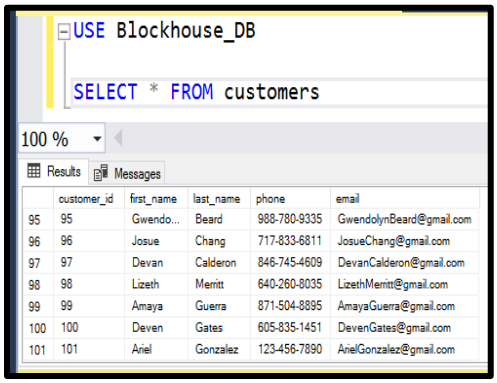
Results | Messages

	customer_id	first_name	last_name	phone	email
95	95	Gwendo...	Beard	988-780-9335	GwendolynBeard@gmail.com
96	96	Josue	Chang	717-833-6811	JosueChang@gmail.com
97	97	Devan	Calderon	846-745-4609	DevanCalderon@gmail.com
98	98	Lizeth	Merritt	640-260-8035	LizethMerritt@gmail.com
99	99	Amaya	Guerra	871-504-8895	AmayaGuerra@gmail.com
100	100	Deven	Gates	605-835-1451	DevenGates@gmail.com
101	101	Ariel	Gonzalez	123-456-7890	ArielGonzalez@gmail.com

UPDATE SCRIPT #1 CUSTOMERS

```
USE Blockhouse_DB
UPDATE customers
SET phone =  '789-456-1230'
WHERE customer_id = 101;
```

Before:



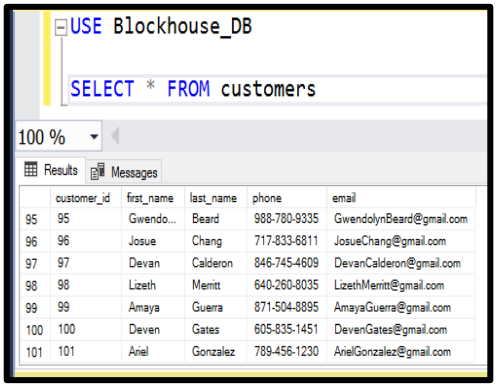
```
USE Blockhouse_DB

SELECT * FROM customers
```

100 %

Results | Messages

	customer_id	first_name	last_name	phone	email
95	95	Gwendo...	Beard	988-780-9335	GwendolynBeard@gmail.com
96	96	Josue	Chang	717-833-6811	JosueChang@gmail.com
97	97	Devan	Calderon	846-745-4609	DevanCalderon@gmail.com
98	98	Lizeth	Merritt	640-260-8035	LizethMerritt@gmail.com
99	99	Amaya	Guerra	871-504-8895	AmayaGuerra@gmail.com
100	100	Deven	Gates	605-835-1451	DevenGates@gmail.com
101	101	Ariel	Gonzalez	123-456-7890	ArielGonzalez@gmail.com

After:



```
USE Blockhouse_DB

SELECT * FROM customers
```

100 %

Results | Messages

	customer_id	first_name	last_name	phone	email
95	95	Gwendo...	Beard	988-780-9335	GwendolynBeard@gmail.com
96	96	Josue	Chang	717-833-6811	JosueChang@gmail.com
97	97	Devan	Calderon	846-745-4609	DevanCalderon@gmail.com
98	98	Lizeth	Merritt	640-260-8035	LizethMerritt@gmail.com
99	99	Amaya	Guerra	871-504-8895	AmayaGuerra@gmail.com
100	100	Deven	Gates	605-835-1451	DevenGates@gmail.com
101	101	Ariel	Gonzalez	789-456-1230	ArielGonzalez@gmail.com

DELETE SCRIPT #1 CUSTOMERS

```
USE Blockhouse_DB
DELETE FROM customers
WHERE customer_id = 101;
```

Before:

```
USE Blockhouse_DB

SELECT * FROM customers
```

100 %

Results | Messages

	customer_id	first_name	last_name	phone	email
95	95	Gwendo...	Beard	988-780-9335	GwendolynBeard@gmail.com
96	96	Josue	Chang	717-833-6811	JosueChang@gmail.com
97	97	Devan	Calderon	846-745-4609	DevanCalderon@gmail.com
98	98	Lizeth	Merritt	640-260-8035	LizethMerritt@gmail.com
99	99	Amaya	Guerra	871-504-8895	AmayaGuerra@gmail.com
100	100	Deven	Gates	605-835-1451	DevenGates@gmail.com
101	101	Ariel	Gonzalez	789-456-1230	ArielGonzalez@gmail.com

After:

```
USE Blockhouse_DB

SELECT * FROM customers
```

100 %

Results | Messages

	customer_id	first_name	last_name	phone	email
94	94	Delilah	Edwards	650-729-7206	DelilahEdwards@gmail.com
95	95	Gwendo...	Beard	988-780-9335	GwendolynBeard@gmail.com
96	96	Josue	Chang	717-833-6811	JosueChang@gmail.com
97	97	Devan	Calderon	846-745-4609	DevanCalderon@gmail.com
98	98	Lizeth	Merritt	640-260-8035	LizethMerritt@gmail.com
99	99	Amaya	Guerra	871-504-8895	AmayaGuerra@gmail.com
100	100	Deven	Gates	605-835-1451	DevenGates@gmail.com

INSERT SCRIPT #2 FOR CUSTOMER_PAYMENTS

```
USE Blockhouse_DB
INSERT customer_payments
VALUES('101','123.45','2021-04-10');
```

Before:

```
USE Blockhouse_DB

SELECT * FROM customer_payments
```

100 %

Results | Messages

	payment_id	invoice_id	amt_paid	payment_date
94	94	94	195.03	2021-02-19
95	95	95	322.74	2021-02-21
96	96	96	191.07	2021-03-06
97	97	97	302.94	2021-03-11
98	98	98	257.4	2021-03-18
99	99	99	232.65	2021-03-23
100	100	100	221.76	2021-03-24

After:

```
USE Blockhouse_DB

SELECT * FROM customer_payments
```

100 %

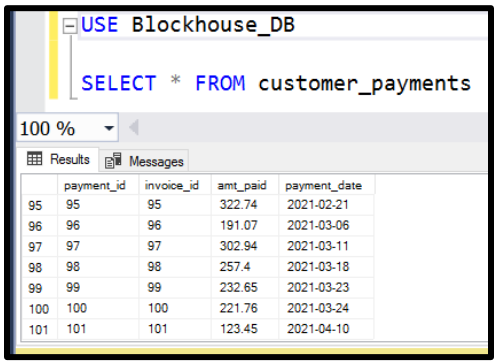
Results | Messages

	payment_id	invoice_id	amt_paid	payment_date
95	95	95	322.74	2021-02-21
96	96	96	191.07	2021-03-06
97	97	97	302.94	2021-03-11
98	98	98	257.4	2021-03-18
99	99	99	232.65	2021-03-23
100	100	100	221.76	2021-03-24
101	101	101	123.45	2021-04-10

UPDATE SCRIPT #2 FOR CUSTOMER_PAYMENTS

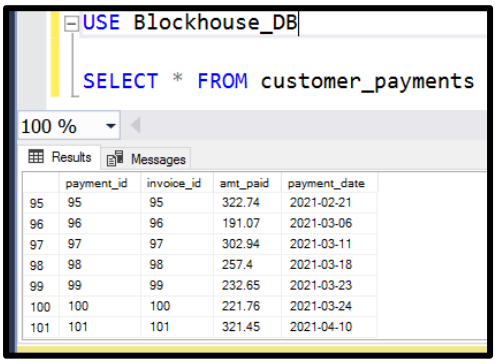
```
USE Blockhouse_DB
UPDATE customer_payments
SET amt_paid = '321.45'
WHERE payment_id = 101;
```

Before:



	payment_id	invoice_id	amt_paid	payment_date
95	95	95	322.74	2021-02-21
96	96	96	191.07	2021-03-06
97	97	97	302.94	2021-03-11
98	98	98	257.4	2021-03-18
99	99	99	232.65	2021-03-23
100	100	100	221.76	2021-03-24
101	101	101	123.45	2021-04-10

After:

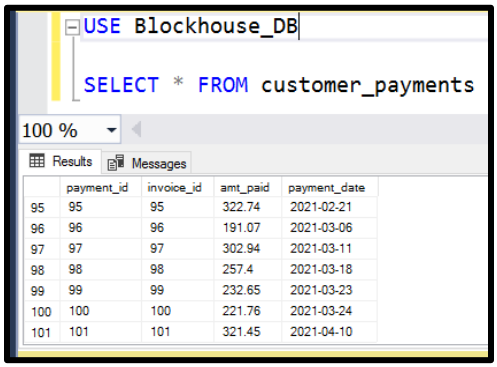


	payment_id	invoice_id	amt_paid	payment_date
95	95	95	322.74	2021-02-21
96	96	96	191.07	2021-03-06
97	97	97	302.94	2021-03-11
98	98	98	257.4	2021-03-18
99	99	99	232.65	2021-03-23
100	100	100	221.76	2021-03-24
101	101	101	321.45	2021-04-10

DELETE SCRIPT #2 FOR CUSTOMER_PAYMENTS

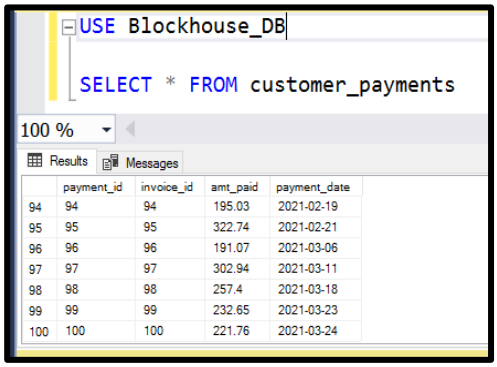
```
USE Blockhouse_DB
DELETE FROM customer_payments
WHERE payment_id = 101;
```

Before:



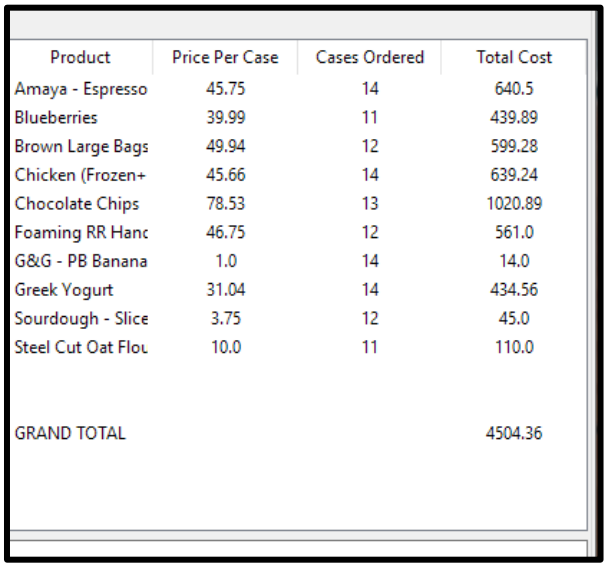
	payment_id	invoice_id	amt_paid	payment_date
95	95	95	322.74	2021-02-21
96	96	96	191.07	2021-03-06
97	97	97	302.94	2021-03-11
98	98	98	257.4	2021-03-18
99	99	99	232.65	2021-03-23
100	100	100	221.76	2021-03-24
101	101	101	321.45	2021-04-10

After:

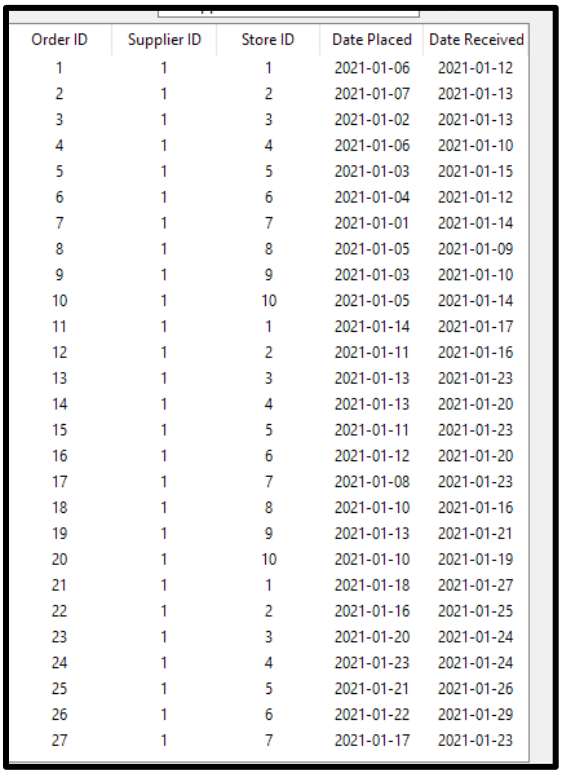


	payment_id	invoice_id	amt_paid	payment_date
94	94	94	195.03	2021-02-19
95	95	95	322.74	2021-02-21
96	96	96	191.07	2021-03-06
97	97	97	302.94	2021-03-11
98	98	98	257.4	2021-03-18
99	99	99	232.65	2021-03-23
100	100	100	221.76	2021-03-24

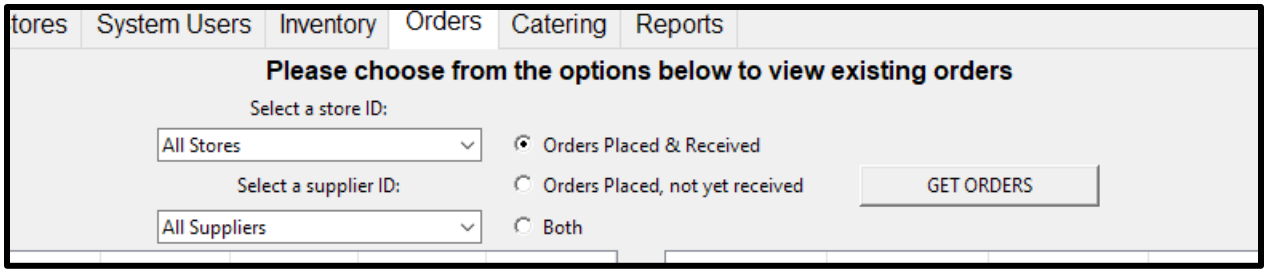
SCREENSHOTS OF FORMS/GUIs

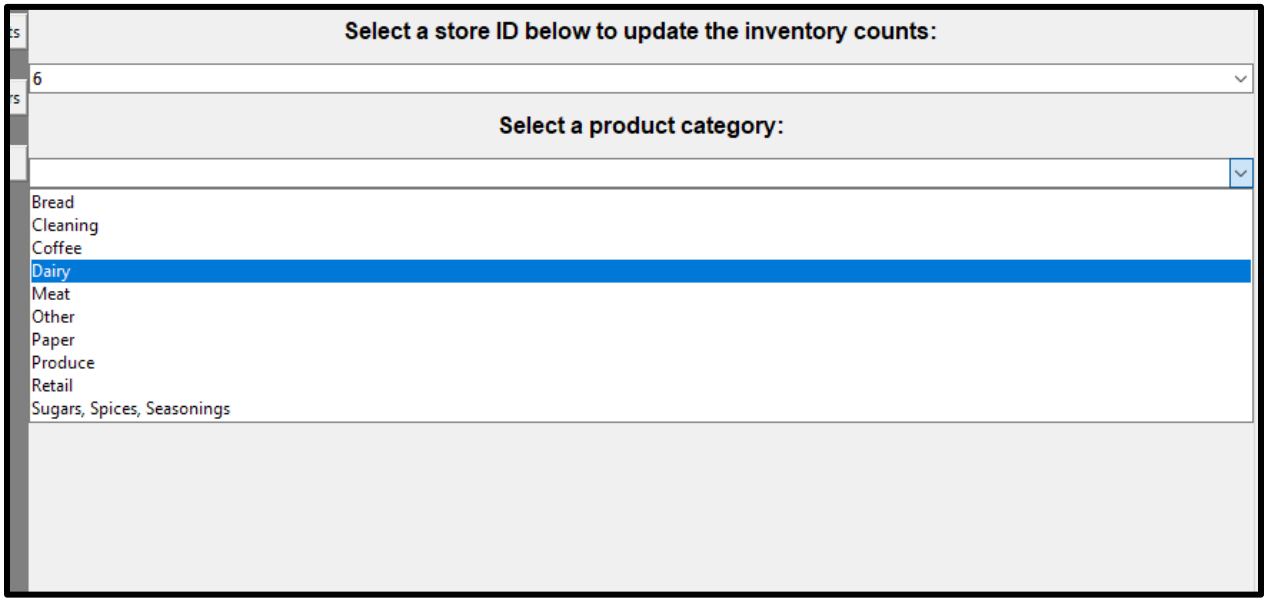


Product	Price Per Case	Cases Ordered	Total Cost
Amaya - Espresso	45.75	14	640.5
Blueberries	39.99	11	439.89
Brown Large Bags	49.94	12	599.28
Chicken (Frozen+	45.66	14	639.24
Chocolate Chips	78.53	13	1020.89
Foaming RR Hanc	46.75	12	561.0
G&G - PB Banana	1.0	14	14.0
Greek Yogurt	31.04	14	434.56
Sourdough - Slice	3.75	12	45.0
Steel Cut Oat Flou	10.0	11	110.0
GRAND TOTAL			4504.36



Order ID	Supplier ID	Store ID	Date Placed	Date Received
1	1	1	2021-01-06	2021-01-12
2	1	2	2021-01-07	2021-01-13
3	1	3	2021-01-02	2021-01-13
4	1	4	2021-01-06	2021-01-10
5	1	5	2021-01-03	2021-01-15
6	1	6	2021-01-04	2021-01-12
7	1	7	2021-01-01	2021-01-14
8	1	8	2021-01-05	2021-01-09
9	1	9	2021-01-03	2021-01-10
10	1	10	2021-01-05	2021-01-14
11	1	1	2021-01-14	2021-01-17
12	1	2	2021-01-11	2021-01-16
13	1	3	2021-01-13	2021-01-23
14	1	4	2021-01-13	2021-01-20
15	1	5	2021-01-11	2021-01-23
16	1	6	2021-01-12	2021-01-20
17	1	7	2021-01-08	2021-01-23
18	1	8	2021-01-10	2021-01-16
19	1	9	2021-01-13	2021-01-21
20	1	10	2021-01-10	2021-01-19
21	1	1	2021-01-18	2021-01-27
22	1	2	2021-01-16	2021-01-25
23	1	3	2021-01-20	2021-01-24
24	1	4	2021-01-23	2021-01-24
25	1	5	2021-01-21	2021-01-26
26	1	6	2021-01-22	2021-01-29
27	1	7	2021-01-17	2021-01-23


tores
System Users
Inventory
Orders
Catering
Reports
Please choose from the options below to view existing orders
Select a store ID:
All Stores
Select a supplier ID:
All Suppliers
Orders Placed & Received
Orders Placed, not yet received
Both
GET ORDERS


Select a store ID below to update the inventory counts:
6
Select a product category:
Bread
Cleaning
Coffee
Dairy
Meat
Other
Paper
Produce
Retail
Sugars, Spices, Seasonings

Andres Pirela Individual Contributions

QUERY/REPORT 1

REPORT NAME: Bread To Order Report

BUSINESS PROBLEMS ADDRESSED: This report displays the current stock of bread products at a particular store that is below the defined par level. This allows the store to know which bread products they will need to order to maintain their stock above par level. The client would use this query to view the current bread product stock, if it is below par level, the query would display these products, this would then allow the client to place an order to restock those items.

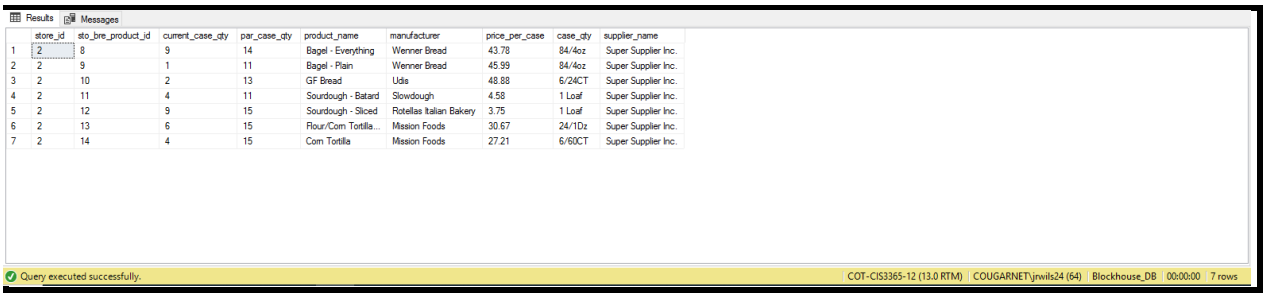
REPORT DESCRIPTION: Used to show the current bread inventory of a given store if it is less than the corresponding par level. It shows product names and their quantities.

1. For showing current bread quantity (if current less than par) of a particular store.

a. View name: bread_qty_to_order

b. WHERE store_id = {} and current_case_qty < par_case_qty

SCREENSHOT OF QUERY RESULTS FOR STORE 2:



Results | Messages

	store_id	sto_bre_product_id	current_case_qty	par_case_qty	product_name	manufacturer	price_per_case	case_qty	supplier_name
1	2	8	9	14	Bagel - Everything	Wenner Bread	43.78	84/4oz	Super Supplier Inc.
2	2	9	1	11	Bagel - Plain	Wenner Bread	45.99	84/4oz	Super Supplier Inc.
3	2	10	2	13	GF Bread	Udis	48.88	6/24CT	Super Supplier Inc.
4	2	11	4	11	Sourdough - Batard	Slowdough	4.58	1 Loaf	Super Supplier Inc.
5	2	12	9	15	Sourdough - Sliced	Rotellas Italian Bakery	3.75	1 Loaf	Super Supplier Inc.
6	2	13	6	15	Flour/Corn Tortilla...	Mission Foods	30.67	24/1Dz	Super Supplier Inc.
7	2	14	4	15	Corn Tortilla	Mission Foods	27.21	6/60CT	Super Supplier Inc.

Query executed successfully. | COT-CIS3365-12 (13.0 RTM) | COUGARNET\jrwils24 (64) | Blockhouse_DB | 00:00:00 | 7 rows

BEGIN (bread_qty_to_order) SCRIPT

```
SELECT
                    blockhouse_stores.store_id,
                    store_bread_qty.sto_bre_product_id,
                    store_bread_qty.current_case_qty,
                    store_bread_qty.par_case_qty,
                    suppliers_bread_products.product_name,
                    suppliers_bread_products.manufacturer,
                    suppliers_bread_products.price_per_case,
                    Suppliers_bread_products.case_qty,
                    suppliers.supplier_name
FROM blockhouse_stores
                    INNER JOIN store_bread_qty ON blockhouse_stores.store_id = store_bread_qty.store_id
                    INNER JOIN suppliers_bread_products ON store_bread_qty.sup_bre_product_id = suppliers_bread_products.sup_bre_product_id
                    INNER JOIN suppliers ON suppliers_bread_products.supplier_id = suppliers.supplier_id
```

END (bread_qty_to_order) SCRIPT

QUERY/REPORT 2

REPORT NAME: Incoming Bread Inventory Report

BUSINESS PROBLEMS ADDRESSED: This report allows the client to view the incoming bread product orders for a particular store. The purpose of this report is to display incoming stock for the client in such a way that they do not accidentally place another restocking order when there is current stock on the way. Thus, providing an organized view of bread products that have been ordered.

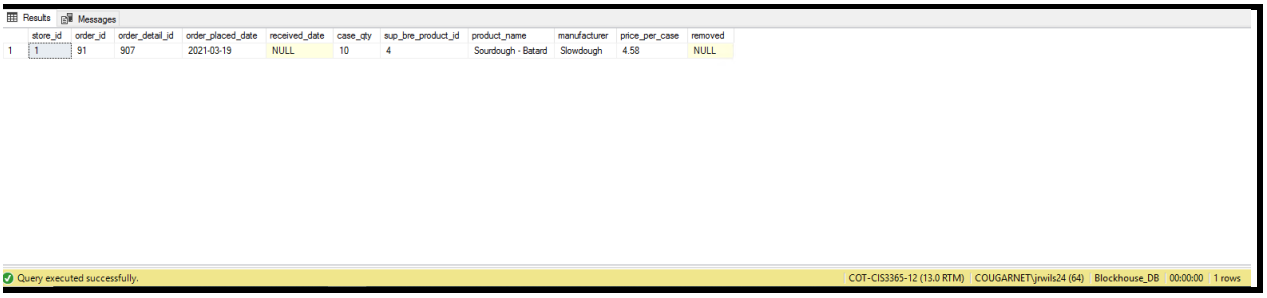
REPORT DESCRIPTION: Used to show the incoming quantities of bread products. It displays the incoming inventory, their quantity, and cost.

1. For showing the incoming quantity of bread products for a given store.

a. View name: **incoming_bread_products**

b. WHERE store_id = {} and received_date IS NULL

SCREENSHOT OF QUERY RESULTS FOR STORE 1:



Results Messages

	store_id	order_id	order_detail_id	order_placed_date	received_date	case_qty	sup_bre_product_id	product_name	manufacturer	price_per_case	removed
1	1	91	907	2021-03-19	NULL	10	4	Sourdough - Batard	Slowdough	4.58	NULL

Query executed successfully. COT-CIS3365-12 (13.0 RTM) COUGARNET\jrwils24 (64) Blockhouse_DB 00:00:00 1 rows

BEGIN (incoming_bread_products) SCRIPT

```
SELECT
                    blockhouse_stores.store_id,
                    orders.order_id,
                    order_details.order_detail_id,
                    orders.order_placed_date,
                    orders.received_date,
                    order_details.case_qty,
                    suppliers_bread_products.sup_bre_product_id,
                    suppliers_bread_products.product_name,
                    suppliers_bread_products.manufacturer,
                    suppliers_bread_products.price_per_case,
                    order_details.removed
FROM order_details
                    INNER JOIN orders ON orders.order_id = order_details.order_id
                    INNER JOIN suppliers_bread_products ON suppliers_bread_products.sup_bre_product_id = order_details.sup_bre_product_id
                    INNER JOIN blockhouse_stores ON blockhouse_stores.store_id = orders.store_id
```

END (incoming_bread_products) SCRIPT

QUERY/REPORT 3

REPORT NAME: Bread Order Errors Report

BUSINESS PROBLEMS ADDRESSED: This report allows the client to verify if bread products were ordered correctly to a specific store. The purpose of this query is to allow the client to verify if the order placed was for the correct amount of product needed. This allows the client to quickly view any incorrect or subpar orders and fix them as needed.

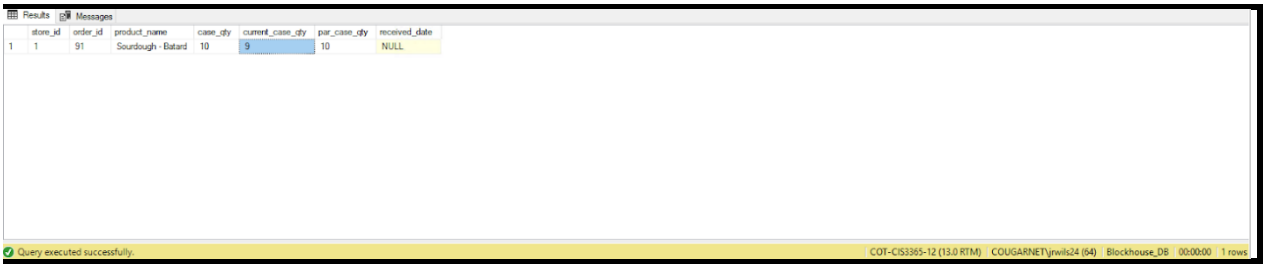
REPORT DESCRIPTION: used to verify that bread products were ordered correctly for a given store.

1. Shows bread products that were ordered incorrectly for a given store.

a. View name: incorrectly_ordered_bread_products

b. WHERE store_id = {} and case_qty + current_case_qty <> par_case_qty and received_date is null

SCREENSHOT OF QUERY RESULTS FOR STORE 1:



BEGIN (incorrectly_ordered_bread_products) SCRIPT

```
SELECT
                orders.store_id,
                orders.order_id,
                suppliers_bread_products.product_name,
                order_details.case_qty,
                store_bread_qty.current_case_qty,
                store_bread_qty.par_case_qty,
                orders.received_date
FROM order_details
                INNER JOIN orders ON order_details.order_id = orders.order_id
                INNER JOIN store_bread_qty ON store_bread_qty.sup_bre_product_id = order_details.sup_bre_product_id and store_bread_qty.store_id =
orders.store_id
                INNER JOIN suppliers_bread_products ON suppliers_bread_products.sup_bre_product_id = order_details.sup_bre_product_id
```

END (incorrectly_ordered_bread_products) SCRIPT

QUERY/REPORT 4

REPORT NAME: Cleaning Order Errors Report

BUSINESS PROBLEMS ADDRESSED: This report allows the client to verify if cleaning products were ordered correctly to a specific store. The purpose of this query is to allow the client to verify if the order placed was for the correct amount of product needed. This allows the client to quickly view any incorrect or subpar orders and fix them as needed.

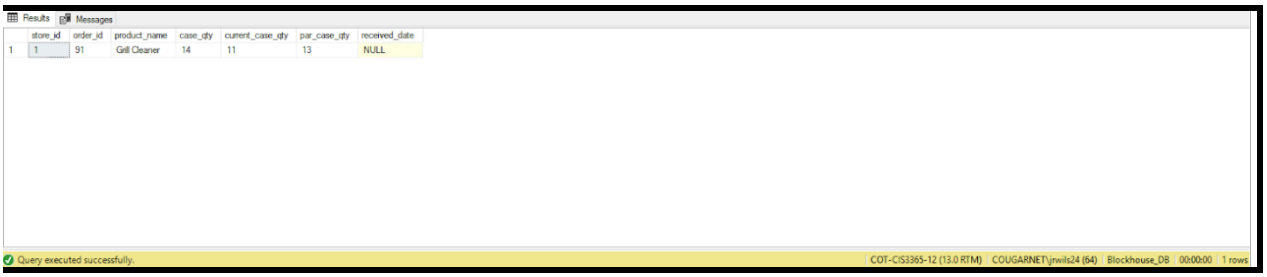
REPORT DESCRIPTION: used to verify that Cleaning products were ordered correctly for a given store.

1. For showing the incoming quantity of cleaning products for a given store.

a. View name: incoming_cleaning_products

b. WHERE store_id = {} and received_date IS NULL

SCREENSHOT OF QUERY RESULTS FOR STORE 1:



BEGIN (incorrectly_ordered_cleaning_products) SCRIPT

```
SELECT
                orders.store_id,
                orders.order_id,
                suppliers_cleaning_products.product_name,
                order_details.case_qty,
                store_cleaning_qty.current_case_qty,
                store_cleaning_qty.par_case_qty,
                orders.received_date
FROM order_details
                INNER JOIN orders ON order_details.order_id = orders.order_id
                INNER JOIN store_cleaning_qty ON store_cleaning_qty.sup_cle_product_id = order_details.sup_cle_product_id and store_cleaning_qty.store_id =
orders.store_id
                INNER JOIN suppliers_cleaning_products ON suppliers_cleaning_products.sup_cle_product_id = order_details.sup_cle_product_id
```

END (incorrectly_ordered_cleaning_products) SCRIPT

COPIES OF CREATE SCRIPTS

```
CREATE TABLE store_produce_qty (
sto_pro_product_id INT IDENTITY(1,1) PRIMARY KEY,
sup_pro_product_id INT NOT NULL,
store_id INT NOT NULL,
current_case_qty INT NOT NULL,
par_case_qty INT NOT NULL,
remove BIT,
UNIQUE(sup_pro_product_id, store_id)
);
```

```
CREATE TABLE store_paper_qty (
sto_pap_product_id INT IDENTITY(1,1) PRIMARY KEY,
sup_pap_product_id INT NOT NULL,
store_id INT NOT NULL,
current_case_qty INT NOT NULL,
par_case_qty INT NOT NULL,
remove BIT,
UNIQUE(sup_pap_product_id, store_id)
);
```

```
CREATE TABLE suppliers_produce_products (
sup_pro_product_id int IDENTITY(1,1) PRIMARY KEY,
supplier_id int NOT NULL,
product_name varchar(255) NOT NULL,
manufacturer varchar(255) NOT NULL,
price_per_case float NOT NULL,
case_qty varchar(255),
discontinued bit,
UNIQUE(supplier_id, product_name, manufacturer));
```

```
CREATE TABLE suppliers_paper_products (
sup_pap_product_id int IDENTITY(1,1) PRIMARY KEY,
supplier_id int NOT NULL,
product_name varchar(255) NOT NULL,
manufacturer varchar(255) NOT NULL,
price_per_case float NOT NULL,
case_qty varchar(255),
discontinued bit,
UNIQUE(supplier_id, product_name, manufacturer)
);
```

COPIES OF ALTER TABLE SCRIPTS

```
ALTER TABLE store_produce_qty
ADD FOREIGN KEY (sup_pro_product_id) REFERENCES suppliers_produce_products (sup_pro_product_id);
```

```
ALTER TABLE  store_paper_qty
ADD FOREIGN KEY (sup_pap_product_id) REFERENCES suppliers_paper_products (sup_pap_product_id);
```

```
ALTER TABLE suppliers_produce_products
ADD FOREIGN KEY (supplier_id) REFERENCES suppliers (supplier_id);
```

```
ALTER TABLE suppliers_paper_products
ADD FOREIGN KEY (supplier_id) REFERENCES suppliers (supplier_id);
```

COPIES OF BULK INSERT SCRIPTS

```
BULK INSERT store_produce_qty
FROM 'C:\ store_produce_qty.txt'
WITH (
        FIELDTERMINATOR = ',',
        ROWTERMINATOR = '\n',
        KEEPNULLS
);
```

```
BULK INSERT store_paper_qty
FROM 'C:\ store_paper_qty.txt'
WITH (
        FIELDTERMINATOR = ',',
```

```
        ROWTERMINATOR = '\n',
        KEEPNULLS
);
```

```
BULK INSERT suppliers_produce_products
FROM 'C:\ suppliers_produce_products.txt'
WITH (
        FIELDTERMINATOR = ',',
        ROWTERMINATOR = '\n',
        KEEPNULLS
);
```

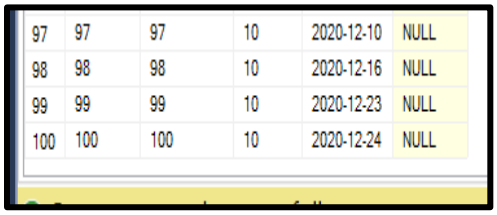
```
BULK INSERT suppliers_paper_products
FROM 'C:\ suppliers_paper_products.txt'
WITH (
        FIELDTERMINATOR = ',',
        ROWTERMINATOR = '\n',
        KEEPNULLS
);
```

COPIES OF ALL SQL UPDATE SCRIPTS

INSERT SCRIPT #1 FOR CATERING_EVENTS

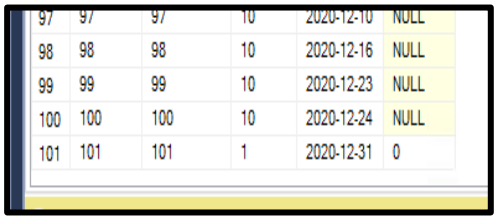
```
USE Blockhouse_DB
INSERT catering_events
VALUES('101','1','2020-12-31',' ');
```

Before:



97	97	97	10	2020-12-10	NULL
98	98	98	10	2020-12-16	NULL
99	99	99	10	2020-12-23	NULL
100	100	100	10	2020-12-24	NULL

After:

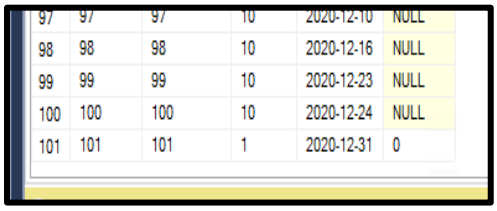


97	97	97	10	2020-12-10	NULL
98	98	98	10	2020-12-16	NULL
99	99	99	10	2020-12-23	NULL
100	100	100	10	2020-12-24	NULL
101	101	101	1	2020-12-31	0

UPDATE SCRIPT #1 CATERING_EVENTS

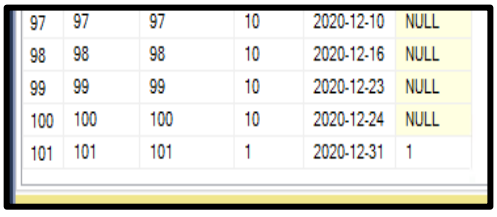
```
USE Blockhouse_DB
UPDATE catering_events
SET canceled =  ‘1'
WHERE event_id = 101;
```

Before:



97	97	97	10	2020-12-10	NULL
98	98	98	10	2020-12-16	NULL
99	99	99	10	2020-12-23	NULL
100	100	100	10	2020-12-24	NULL
101	101	101	1	2020-12-31	0

After:

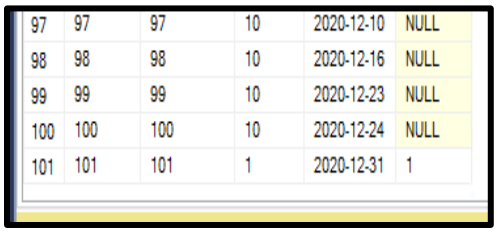


97	97	97	10	2020-12-10	NULL
98	98	98	10	2020-12-16	NULL
99	99	99	10	2020-12-23	NULL
100	100	100	10	2020-12-24	NULL
101	101	101	1	2020-12-31	1

DELETE SCRIPT #1 CATERING_EVENTS

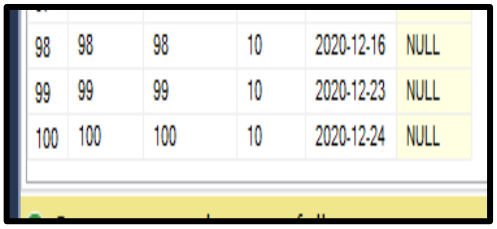
```
USE Blockhouse_DB
DELETE FROM catering_events
WHERE event_id = 101;
```

Before:



97	97	97	10	2020-12-10	NULL
98	98	98	10	2020-12-16	NULL
99	99	99	10	2020-12-23	NULL
100	100	100	10	2020-12-24	NULL
101	101	101	1	2020-12-31	1

After:

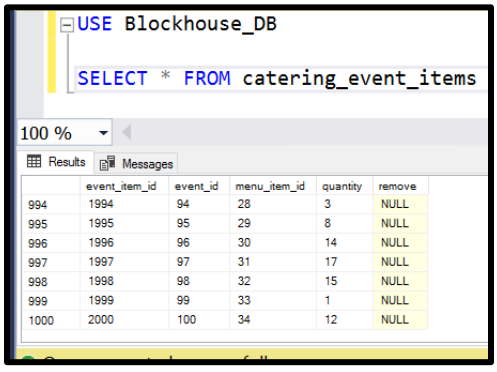


98	98	98	10	2020-12-16	NULL
99	99	99	10	2020-12-23	NULL
100	100	100	10	2020-12-24	NULL

INSERT SCRIPT #2 FOR CATERING_EVENT_ITEMS

```
USE Blockhouse_DB
INSERT catering_event_items
VALUES('101','30','21',' ');
```

Before:



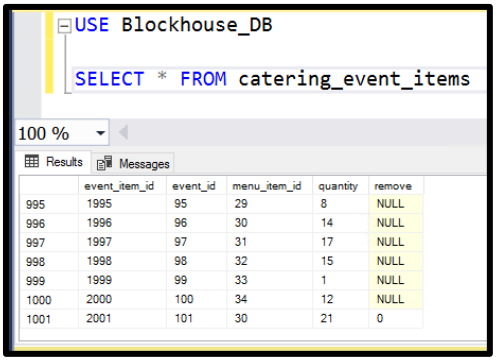
```
USE Blockhouse_DB

SELECT * FROM catering_event_items
```

100 %

Results | Messages

	event_item_id	event_id	menu_item_id	quantity	remove
994	1994	94	28	3	NULL
995	1995	95	29	8	NULL
996	1996	96	30	14	NULL
997	1997	97	31	17	NULL
998	1998	98	32	15	NULL
999	1999	99	33	1	NULL
1000	2000	100	34	12	NULL

After:



```
USE Blockhouse_DB

SELECT * FROM catering_event_items
```

100 %

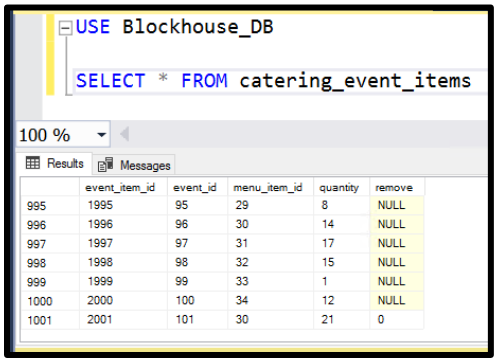
Results | Messages

	event_item_id	event_id	menu_item_id	quantity	remove
995	1995	95	29	8	NULL
996	1996	96	30	14	NULL
997	1997	97	31	17	NULL
998	1998	98	32	15	NULL
999	1999	99	33	1	NULL
1000	2000	100	34	12	NULL
1001	2001	101	30	21	0

UPDATE SCRIPT #2 FOR CATERING_EVENT_ITEMS

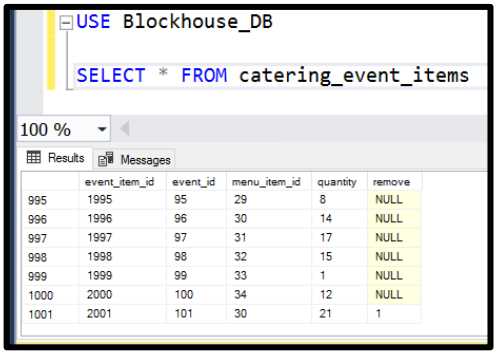
```
USE Blockhouse_DB
UPDATE catering_event_items
SET remove = '1'
WHERE event_item_id = 2001;
```

Before:



	event_item_id	event_id	menu_item_id	quantity	remove
995	1995	95	29	8	NULL
996	1996	96	30	14	NULL
997	1997	97	31	17	NULL
998	1998	98	32	15	NULL
999	1999	99	33	1	NULL
1000	2000	100	34	12	NULL
1001	2001	101	30	21	0

After:

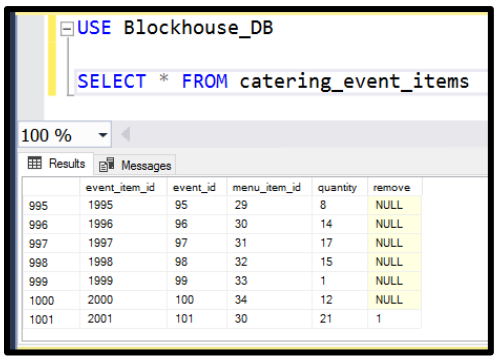


	event_item_id	event_id	menu_item_id	quantity	remove
995	1995	95	29	8	NULL
996	1996	96	30	14	NULL
997	1997	97	31	17	NULL
998	1998	98	32	15	NULL
999	1999	99	33	1	NULL
1000	2000	100	34	12	NULL
1001	2001	101	30	21	1

DELETE SCRIPT #2 FOR CATERING_EVENT_ITEMS

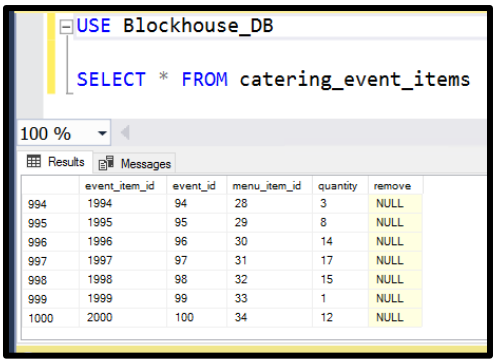
```
USE Blockhouse_DB
DELETE FROM catering_event_items
WHERE event_item_id = 2001;
```

Before:



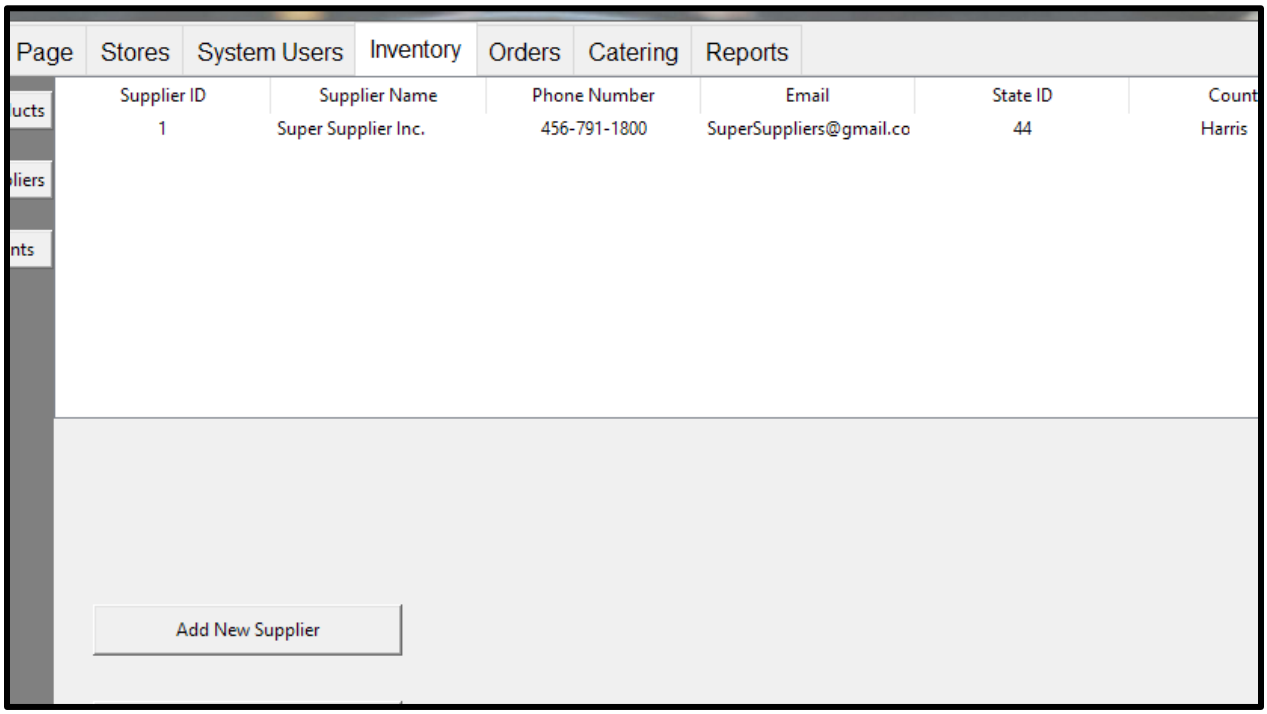
	event_item_id	event_id	menu_item_id	quantity	remove
995	1995	95	29	8	NULL
996	1996	96	30	14	NULL
997	1997	97	31	17	NULL
998	1998	98	32	15	NULL
999	1999	99	33	1	NULL
1000	2000	100	34	12	NULL
1001	2001	101	30	21	1

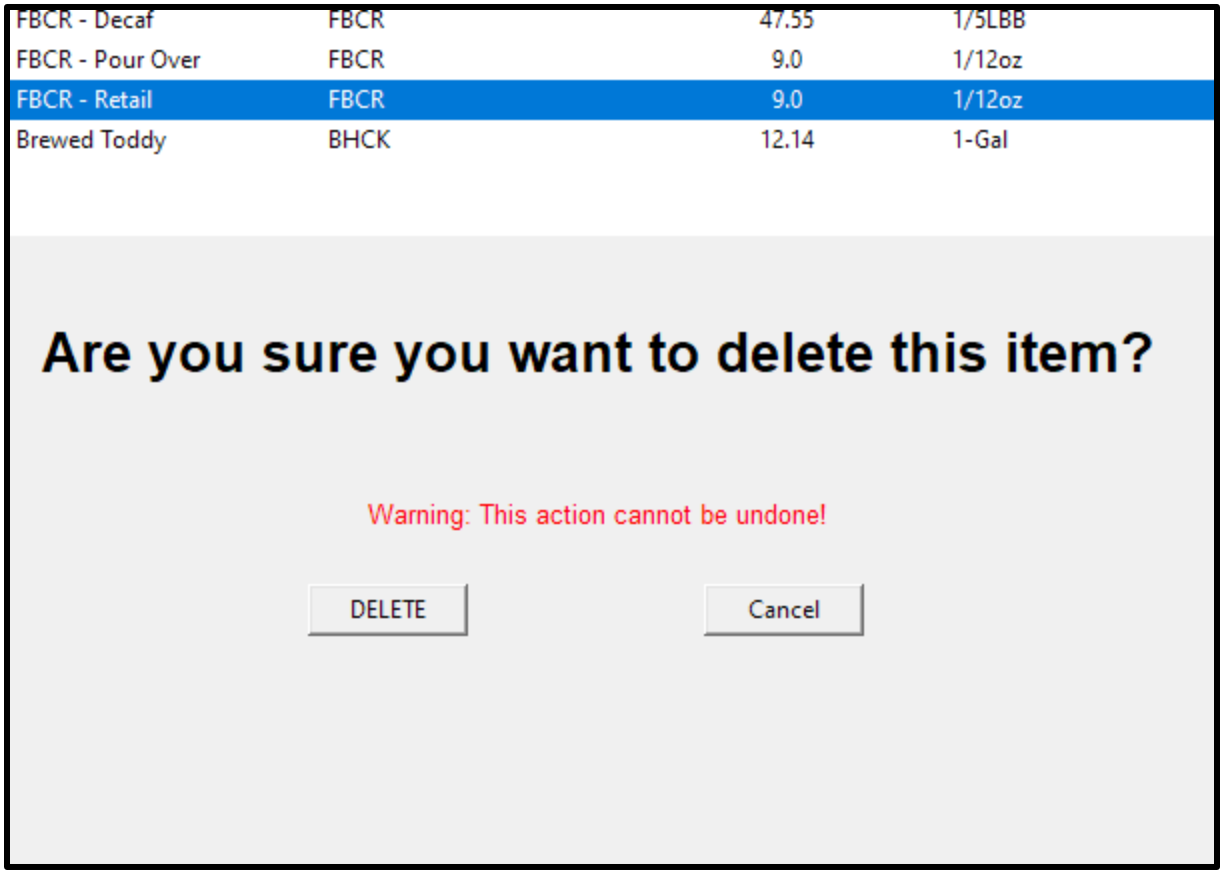
After:

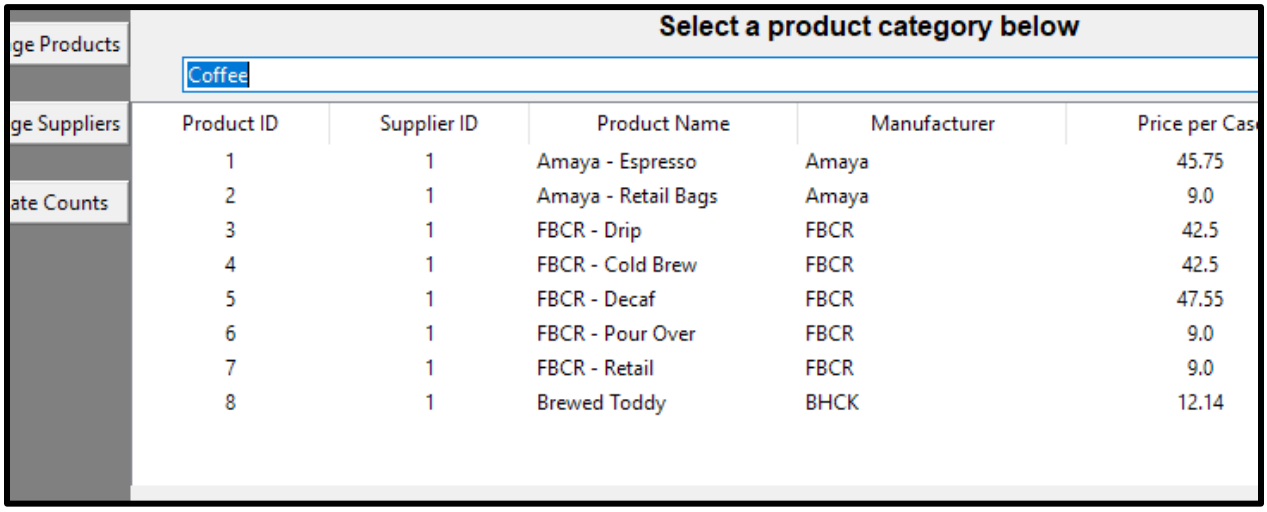


	event_item_id	event_id	menu_item_id	quantity	remove
994	1994	94	28	3	NULL
995	1995	95	29	8	NULL
996	1996	96	30	14	NULL
997	1997	97	31	17	NULL
998	1998	98	32	15	NULL
999	1999	99	33	1	NULL
1000	2000	100	34	12	NULL

SCREENSHOTS OF FORMS/GUIs




FBCR - Decaf FBCR 47.55 1/5LBB
FBCR - Pour Over FBCR 9.0 1/12oz
FBCR - Retail FBCR 9.0 1/12oz
Brewed Toddy BHCK 12.14 1-Gal
Are you sure you want to delete this item?
Warning: This action cannot be undone!
DELETE
Cancel


ge Products
ge Suppliers
ate Counts
Select a product category below
Coffee
Product ID Supplier ID Product Name Manufacturer Price per Cas
1 1 Amaya - Espresso Amaya 45.75
2 1 Amaya - Retail Bags Amaya 9.0
3 1 FBCR - Drip FBCR 42.5
4 1 FBCR - Cold Brew FBCR 42.5
5 1 FBCR - Decaf FBCR 47.55
6 1 FBCR - Pour Over FBCR 9.0
7 1 FBCR - Retail FBCR 9.0
8 1 Brewed Toddy BHCK 12.14

Dan Thomas Individual Contributions

QUERY/REPORT 1

REPORT NAME: Dairy To Order Report

BUSINESS PROBLEMS ADDRESSED: This report shows which dairy products need to be ordered. A user would want to use this report because it allows them to quickly determine which dairy products are not currently at par levels and they can construct an order appropriately.

REPORT DESCRIPTION: Used to show the current Dairy inventory of a given store if it is less than the corresponding par level. It shows product names and their quantities.

1. For showing current dairy quantity (if current less than par) of a particular store.
    a. View name: **dairy_qty_to_order**
    b. WHERE store_id = {} and current_case_qty < par_case_qty

SCREENSHOT OF QUERY RESULTS FOR STORE 1:

Results | Messages

	store_id	sto_dai_product_id	current_case_qty	par_case_qty	product_name	manufacturer	price_per_case	case_qty	supplier_name
1	1	141	2	10	Whole Milk	Mill-King	23.31	4-1 Gal	Super Supplier Inc.
2	1	142	5	13	1% Milk	Mill-King	21.56	4-1 Gal	Super Supplier Inc.
3	1	143	1	10	Barista Almond Milk	Pacific	29.33	12/Qt	Super Supplier Inc.
4	1	144	2	15	Barista Oat Milk	Pacific	32.77	12/QT	Super Supplier Inc.
5	1	145	11	14	1/2 & 1/2	Creamer Half & Half	25.38	12/QT	Super Supplier Inc.
6	1	146	6	14	Greek Yogurt	Yogurt Greek Plain 2%	31.04	6/35.3oz	Super Supplier Inc.
7	1	147	3	10	Butter	Butter Unsalted European Style	97.78	36/1LB	Super Supplier Inc.
8	1	148	9	11	Shredded Cheese	Golden Harvest	28.78	2/5LB	Super Supplier Inc.
9	1	149	9	15	Gouda Cheese	Roth Case	67.13	4/2.5LB	Super Supplier Inc.
10	1	150	4	12	White American Cheese	Schreiber Cheese	14.79	4/5LB	Super Supplier Inc.
11	1	151	8	12	Feta Crumbles	Stella	10	4/2.5LB	Super Supplier Inc.
12	1	152	6	11	Cream Cheese	Philadelphia	37.29	12/8oz	Super Supplier Inc.
13	1	153	11	14	Indiv Cream Cheese	Philadelphia	25.73	100/1oz	Super Supplier Inc.
14	1	154	1	14	Eggs	Packer	28.67	1/15DZ	Super Supplier Inc.

Query executed successfully. | COT-CIS3365-12 (13.0 RTM) | COUGARNET\jrwils24 (64) | Blockhouse_DB | 00:00:00 | 14 rows

BEGIN (dairy_qty_to_order) SCRIPT

```
SELECT
                    blockhouse_stores.store_id,
                    store_dairy_qty.sto_dai_product_id,
                    store_dairy_qty.current_case_qty,
                    store_dairy_qty.par_case_qty,
                    suppliers_dairy_products.product_name,
                    suppliers_dairy_products.manufacturer,
                    suppliers_dairy_products.price_per_case,
                    Suppliers_dairy_products.case_qty,
                    suppliers.supplier_name
FROM blockhouse_stores
                    INNER JOIN store_dairy_qty ON blockhouse_stores.store_id = store_dairy_qty.store_id
                    INNER JOIN suppliers_dairy_products ON store_dairy_qty.sup_dai_product_id = suppliers_dairy_products.sup_dai_product_id
                    INNER JOIN suppliers ON suppliers_dairy_products.supplier_id = suppliers.supplier_id
```

END (dairy_qty_to_order) SCRIPT

QUERY/REPORT 2

REPORT NAME: Incoming Dairy Inventory Report

BUSINESS PROBLEMS ADDRESSED: This report shows incoming dairy products. A user would want to use this report to quickly determine whether enough dairy products have been ordered to fulfill their par level needs.

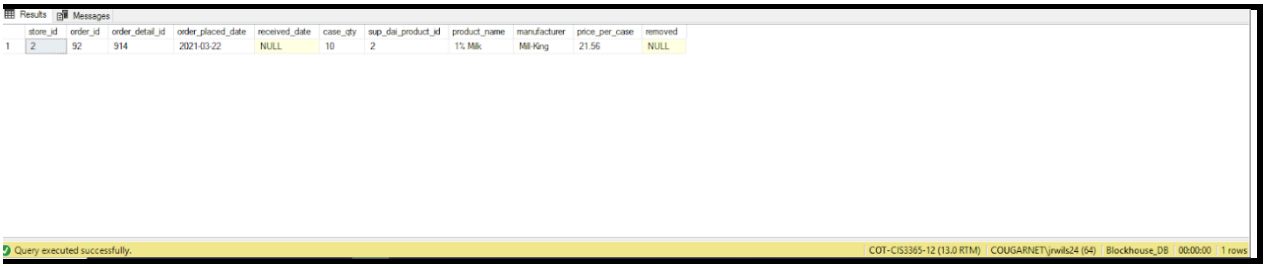
REPORT DESCRIPTION: used to show the incoming quantities of dairy products. It displays the incoming inventory, their quantity, and cost.

1. For showing the incoming quantity of dairy products for a given store.

a. View name: incoming_dairy_products

b. WHERE store_id = {} and received_date IS NULL

SCREENSHOT OF QUERY RESULTS FOR STORE 2:



BEGIN (incoming_dairy_products) SCRIPT

```
SELECT
                    blockhouse_stores.store_id,
                    orders.order_id,
                    order_details.order_detail_id,
                    orders.order_placed_date,
                    orders.received_date,
                    order_details.case_qty,
                    suppliers_dairy_products.sup_dai_product_id,
                    suppliers_dairy_products.product_name,
                    suppliers_dairy_products.manufacturer,
                    suppliers_dairy_products.price_per_case,
                    order_details.removed
FROM order_details
                    INNER JOIN orders ON orders.order_id = order_details.order_id
                    INNER JOIN suppliers_dairy_products ON suppliers_dairy_products.sup_dai_product_id = order_details.sup_dai_product_id
                    INNER JOIN blockhouse_stores ON blockhouse_stores.store_id = orders.store_id
```

END (incoming_dairy_products) SCRIPT

QUERY/REPORT 3

REPORT NAME: Dairy Order Errors Report

BUSINESS PROBLEMS ADDRESSED: This report shows any ordered dairy products where the amounts ordered + the current quantity don't equal the par quantity. A user would want to use this report to quickly determine whether they should adjust or drop the item from the order.

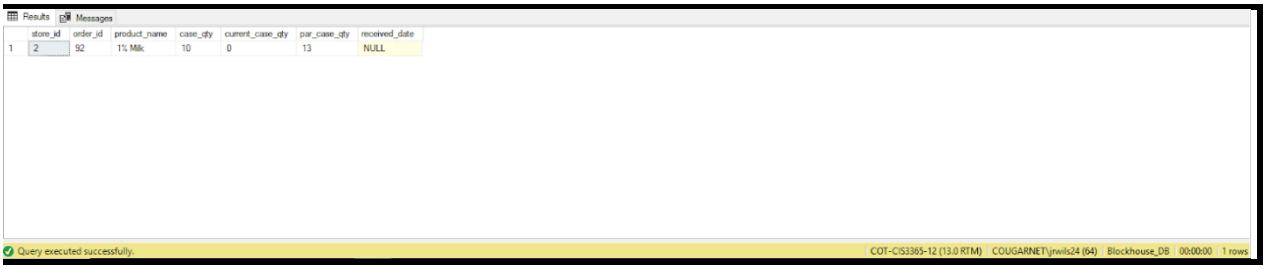
REPORT DESCRIPTION: used to verify that Dairy products were ordered correctly for a given store.

1. Shows dairy products that were ordered incorrectly for a given store.

a. View name: incorrectly_ordered_dairy_products

b. WHERE store_id = {} and case_qty + current_case_qty <> par_case_qty and received_date is null

SCREENSHOT OF QUERY RESULTS FOR STORE 2:



BEGIN (incorrectly_ordered_bread_products) SCRIPT

```
SELECT
                orders.store_id,
                orders.order_id,
                suppliers_dairy_products.product_name,
                order_details.case_qty,
                store_dairy_qty.current_case_qty,
                store_dairy_qty.par_case_qty,
                orders.received_date
FROM order_details
                INNER JOIN orders ON order_details.order_id = orders.order_id
                INNER JOIN store_dairy_qty ON store_dairy_qty.sup_dai_product_id = order_details.sup_dai_product_id and store_dairy_qty.store_id =
orders.store_id
                INNER JOIN suppliers_dairy_products ON suppliers_dairy_products.sup_dai_product_id = order_details.sup_dai_product_id
```

END (incorrectly_ordered_bread_products) SCRIPT

QUERY/REPORT 4

REPORT NAME: Incoming Cleaning Inventory Report

BUSINESS PROBLEMS ADDRESSED: This report shows incoming cleaning products. A user would want to use this report to quickly determine whether enough cleaning products have been ordered to fulfill their par level needs.

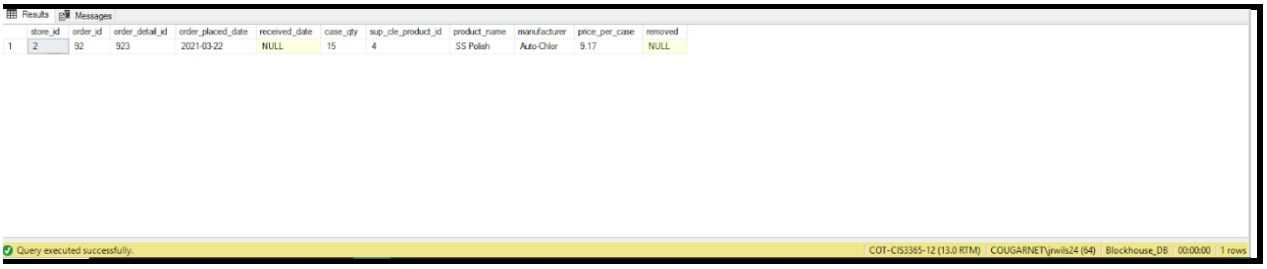
REPORT DESCRIPTION: used to show the incoming quantities of Cleaning products. It displays the incoming inventory, their quantity, and cost.

1. For showing the incoming quantity of cleaning products for a given store.

a. View name: incoming_cleaning_products

b. WHERE store_id = {} and received_date IS NULL

SCREENSHOT OF QUERY RESULTS FOR STORE 2:



BEGIN (incoming_cleaning_products) SCRIPT

```
SELECT
                blockhouse_stores.store_id,
                orders.order_id,
                order_details.order_detail_id,
                orders.order_placed_date,
                orders.received_date,
                order_details.case_qty,
                suppliers_cleaning_products.sup_cle_product_id,
                suppliers_cleaning_products.product_name,
                suppliers_cleaning_products.manufacturer,
                suppliers_cleaning_products.price_per_case,
                order_details.removed
FROM order_details
                INNER JOIN orders ON orders.order_id = order_details.order_id
                INNER JOIN suppliers_cleaning_products ON suppliers_cleaning_products.sup_cle_product_id = order_details.sup_cle_product_id
                INNER JOIN blockhouse_stores ON blockhouse_stores.store_id = orders.store_id
```

END (incoming_cleaning_products) SCRIPT

COPIES OF CREATE SCRIPTS

```
CREATE TABLE store_retail_qty (
sto_ret_product_id INT IDENTITY(1,1) PRIMARY KEY,
sup_ret_product_id INT NOT NULL,
store_id INT NOT NULL,
current_case_qty INT NOT NULL,
par_case_qty INT NOT NULL,
remove BIT,
UNIQUE(sup_ret_product_id, store_id)
);
```

```
CREATE TABLE store_cleaning_qty (
sto_cle_product_id INT IDENTITY(1,1) PRIMARY KEY,
sup_cle_product_id INT NOT NULL,
store_id INT NOT NULL,
current_case_qty INT NOT NULL,
par_case_qty INT NOT NULL,
remove BIT,
UNIQUE(sup_cle_product_id, store_id)
);
```

```
CREATE TABLE suppliers_retail_products (
sup_ret_product_id int IDENTITY(1,1) PRIMARY KEY,
supplier_id int NOT NULL,
product_name varchar(255) NOT NULL,
manufacturer varchar(255) NOT NULL,
price_per_case float NOT NULL,
case_qty varchar(255),
discontinued bit,
UNIQUE(supplier_id, product_name, manufacturer));
```

```
CREATE TABLE suppliers_cleaning_products (
sup_cle_product_id int IDENTITY(1,1) PRIMARY KEY,
supplier_id int NOT NULL,
product_name varchar(255) NOT NULL,
manufacturer varchar(255) NOT NULL,
price_per_case float NOT NULL,
case_qty varchar(255),
discontinued bit,
UNIQUE(supplier_id, product_name, manufacturer)
);
```

COPIES OF ALTER TABLE SCRIPTS

```
ALTER TABLE store_retail_qty
ADD FOREIGN KEY (sup_ret_product_id) REFERENCES suppliers_retail_products (sup_ret_product_id);
```

```
ALTER TABLE  store_cleaning_qty
ADD FOREIGN KEY (sup_cle_product_id) REFERENCES suppliers_cleaning_products (sup_cle_product_id);
```

```
ALTER TABLE suppliers_retail_products
ADD FOREIGN KEY (supplier_id) REFERENCES suppliers (supplier_id);
```

```
ALTER TABLE suppliers_cleaning_products
ADD FOREIGN KEY (supplier_id) REFERENCES suppliers (supplier_id);
```

COPIES OF BULK INSERT SCRIPTS

```
BULK INSERT store_retail_qty
FROM 'C:\store_retail_qty.txt'
WITH (
        FIELDTERMINATOR = ',',
        ROWTERMINATOR = '\n',
        KEEPNULLS
);
```

```
BULK INSERT store_cleaning_qty
FROM 'C:\store_cleaning_qty.txt'
WITH (
        FIELDTERMINATOR = ',',
        ROWTERMINATOR = '\n',
        KEEPNULLS
);
```

```
BULK INSERT suppliers_retail_products
FROM 'C:\suppliers_retail_products.txt'
WITH (
        FIELDTERMINATOR = ',',
        ROWTERMINATOR = '\n',
        KEEPNULLS
);
```

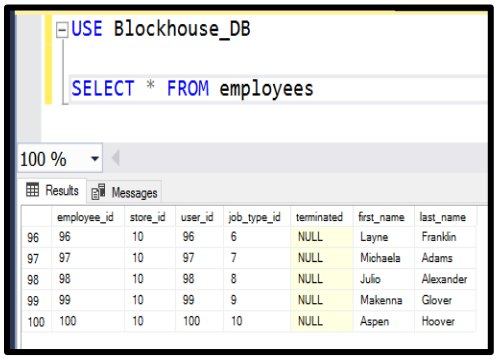
```
BULK INSERT suppliers_cleaning_products
FROM 'C:\suppliers_cleaning_products.txt'
WITH (
        FIELDTERMINATOR = ',',
        ROWTERMINATOR = '\n',
        KEEPNULLS
);
```

COPIES OF ALL SQL UPDATE SCRIPTS

INSERT SCRIPT #1 FOR EMPLOYEES

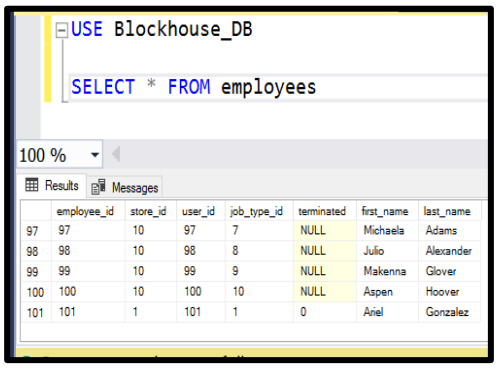
```
USE Blockhouse_DB
INSERT employees
VALUES('1','101','1',' ','Ariel','Gonzalez');
```

Before:



	employee_id	store_id	user_id	job_type_id	terminated	first_name	last_name
96	96	10	96	6	NULL	Layne	Franklin
97	97	10	97	7	NULL	Michaela	Adams
98	98	10	98	8	NULL	Julio	Alexander
99	99	10	99	9	NULL	Makenna	Glover
100	100	10	100	10	NULL	Aspen	Hoover

After:

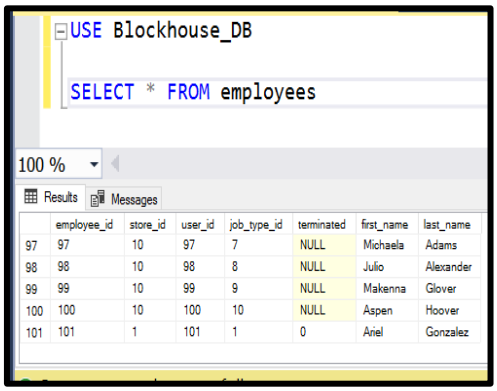


	employee_id	store_id	user_id	job_type_id	terminated	first_name	last_name
97	97	10	97	7	NULL	Michaela	Adams
98	98	10	98	8	NULL	Julio	Alexander
99	99	10	99	9	NULL	Makenna	Glover
100	100	10	100	10	NULL	Aspen	Hoover
101	101	1	101	1	0	Ariel	Gonzalez

UPDATE SCRIPT #1 EMPLOYEES

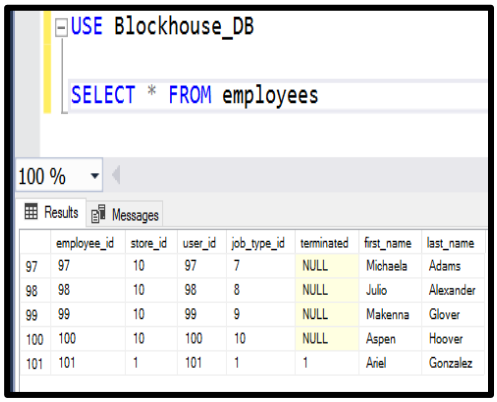
```
USE Blockhouse_DB
UPDATE employees
SET terminated =  '1'
WHERE employee_id = 101;
```

Before:



	employee_id	store_id	user_id	job_type_id	terminated	first_name	last_name
97	97	10	97	7	NULL	Michaela	Adams
98	98	10	98	8	NULL	Julio	Alexander
99	99	10	99	9	NULL	Makenna	Glover
100	100	10	100	10	NULL	Aspen	Hoover
101	101	1	101	1	0	Ariel	Gonzalez

After:

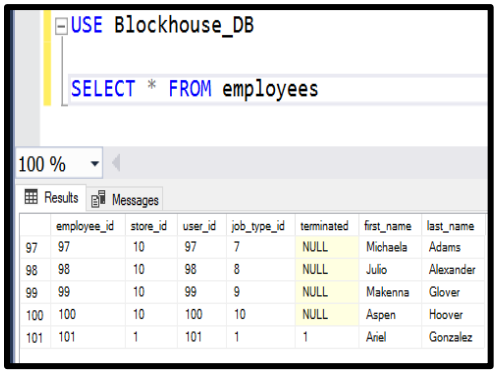


	employee_id	store_id	user_id	job_type_id	terminated	first_name	last_name
97	97	10	97	7	NULL	Michaela	Adams
98	98	10	98	8	NULL	Julio	Alexander
99	99	10	99	9	NULL	Makenna	Glover
100	100	10	100	10	NULL	Aspen	Hoover
101	101	1	101	1	1	Ariel	Gonzalez

DELETE SCRIPT #1 EMPLOYEES

```
USE Blockhouse_DB
DELETE FROM employees
WHERE employee_id = 101;
```

Before:



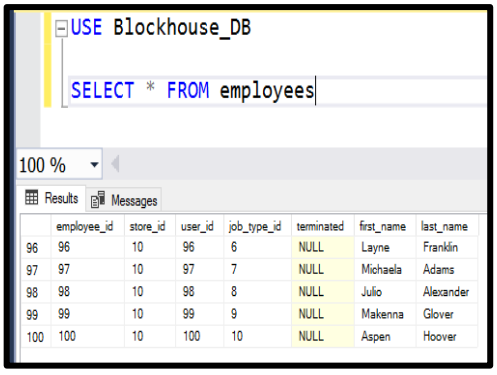
```
USE Blockhouse_DB

SELECT * FROM employees
```

100 %

Results | Messages

	employee_id	store_id	user_id	job_type_id	terminated	first_name	last_name
97	97	10	97	7	NULL	Michaela	Adams
98	98	10	98	8	NULL	Julio	Alexander
99	99	10	99	9	NULL	Makenna	Glover
100	100	10	100	10	NULL	Aspen	Hoover
101	101	1	101	1	1	Ariel	Gonzalez

After:



```
USE Blockhouse_DB

SELECT * FROM employees
```

100 %

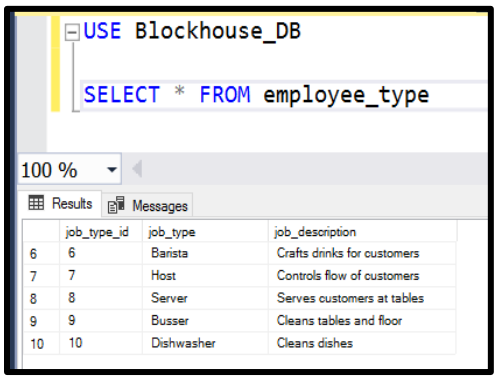
Results | Messages

	employee_id	store_id	user_id	job_type_id	terminated	first_name	last_name
96	96	10	96	6	NULL	Layne	Franklin
97	97	10	97	7	NULL	Michaela	Adams
98	98	10	98	8	NULL	Julio	Alexander
99	99	10	99	9	NULL	Makenna	Glover
100	100	10	100	10	NULL	Aspen	Hoover

INSERT SCRIPT #2 FOR EMPLOYEE_TYPE

```
USE Blockhouse_DB
INSERT employee_type
VALUES('House Holder','Owner of store');
```

Before:



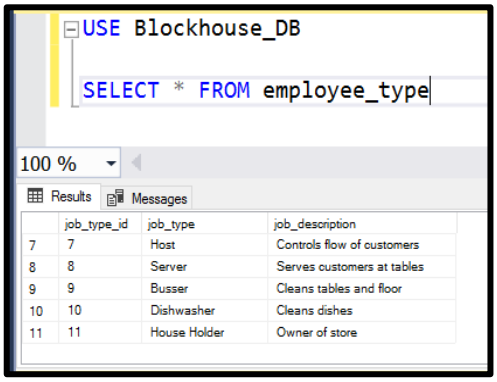
```
USE Blockhouse_DB

SELECT * FROM employee_type
```

100 %

Results | Messages

	job_type_id	job_type	job_description
6	6	Barista	Crafts drinks for customers
7	7	Host	Controls flow of customers
8	8	Server	Serves customers at tables
9	9	Busser	Cleans tables and floor
10	10	Dishwasher	Cleans dishes

After:



```
USE Blockhouse_DB

SELECT * FROM employee_type
```

100 %

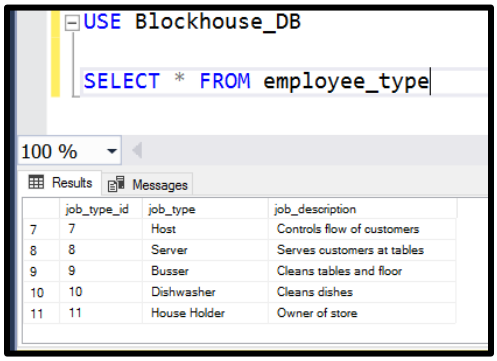
Results | Messages

	job_type_id	job_type	job_description
7	7	Host	Controls flow of customers
8	8	Server	Serves customers at tables
9	9	Busser	Cleans tables and floor
10	10	Dishwasher	Cleans dishes
11	11	House Holder	Owner of store

UPDATE SCRIPT #2 FOR EMPLOYEE_TYPE

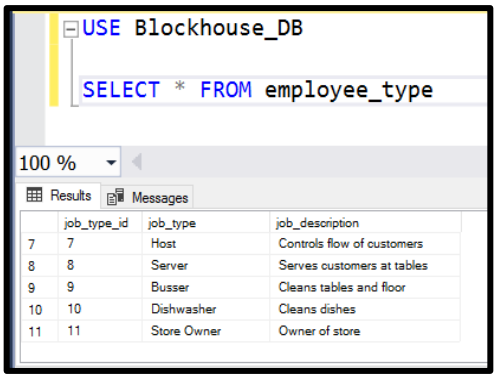
```
USE Blockhouse_DB
UPDATE employee_type
SET job_type = 'Store Owner'
WHERE job_type_id = 11;
```

Before:



	job_type_id	job_type	job_description
7	7	Host	Controls flow of customers
8	8	Server	Serves customers at tables
9	9	Busser	Cleans tables and floor
10	10	Dishwasher	Cleans dishes
11	11	House Holder	Owner of store

After:

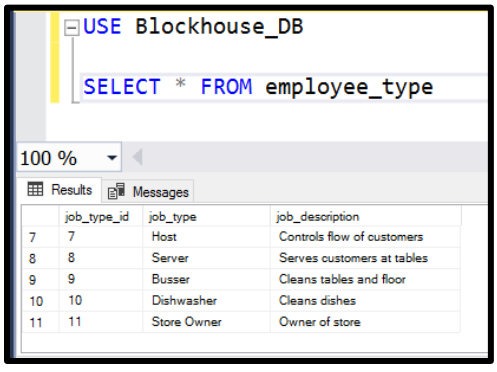


	job_type_id	job_type	job_description
7	7	Host	Controls flow of customers
8	8	Server	Serves customers at tables
9	9	Busser	Cleans tables and floor
10	10	Dishwasher	Cleans dishes
11	11	Store Owner	Owner of store

DELETE SCRIPT #2 FOR EMPLOYEE_TYPE

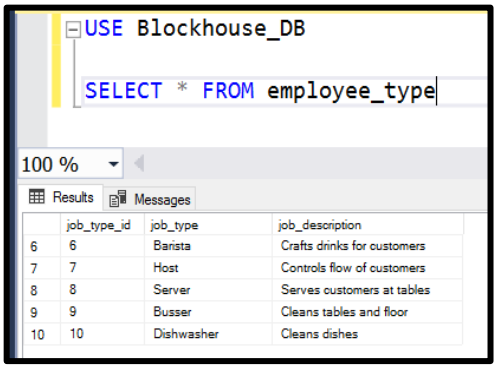
```
USE Blockhouse_DB
DELETE FROM employee_type
WHERE job_type_id = 11;
```

Before:



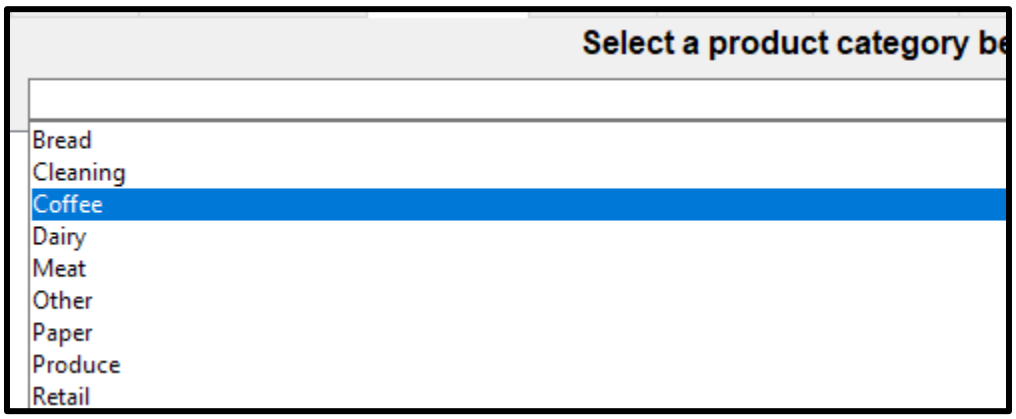
	job_type_id	job_type	job_description
7	7	Host	Controls flow of customers
8	8	Server	Serves customers at tables
9	9	Busser	Cleans tables and floor
10	10	Dishwasher	Cleans dishes
11	11	Store Owner	Owner of store

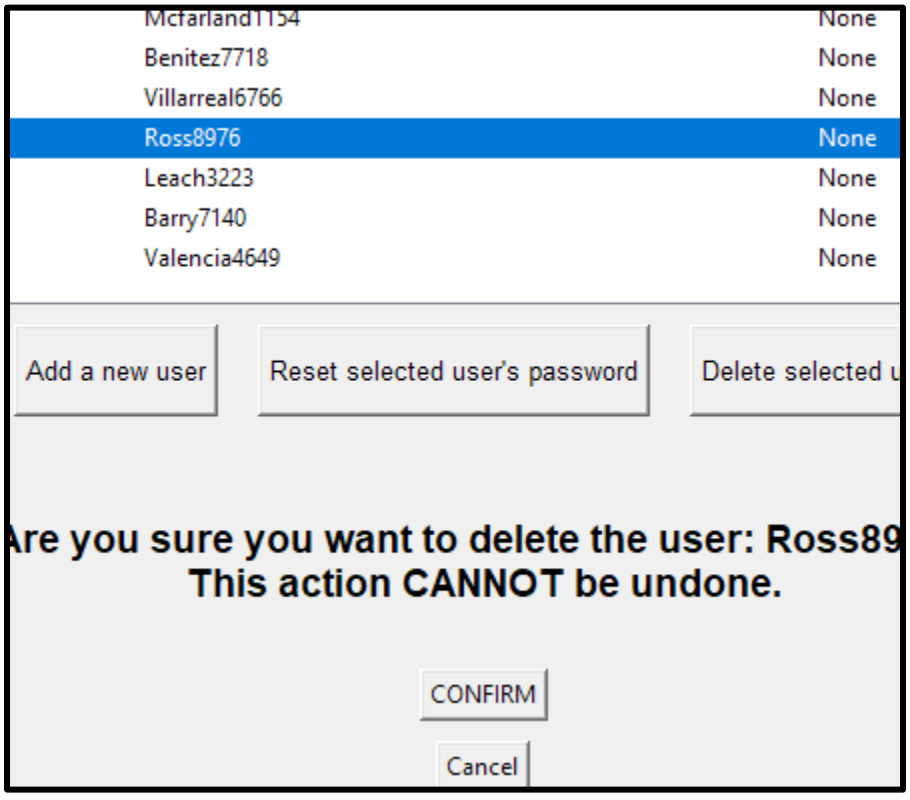
After:

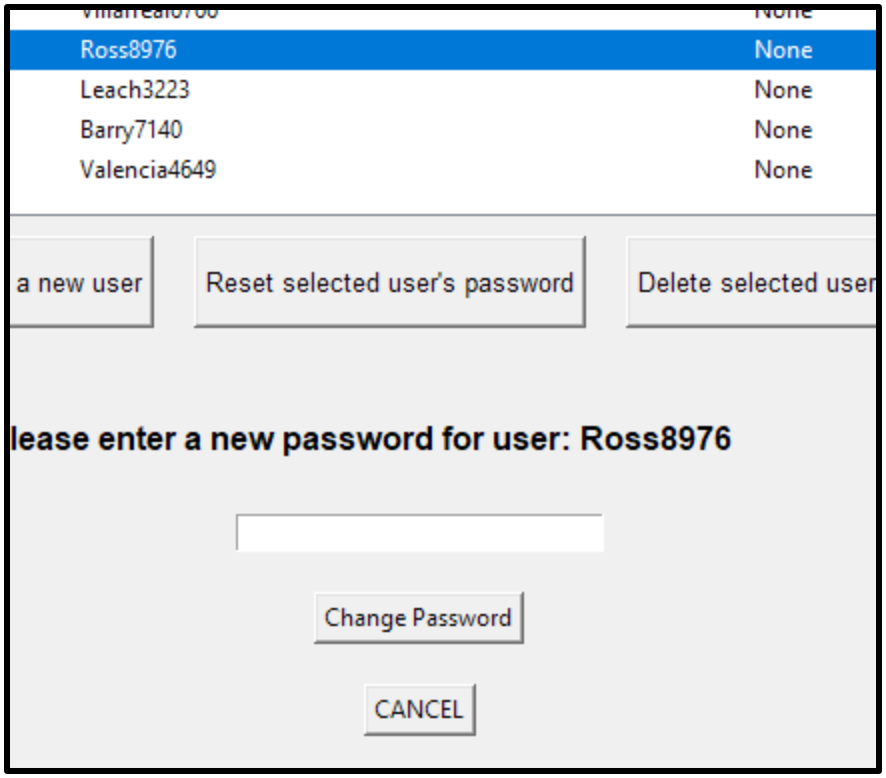


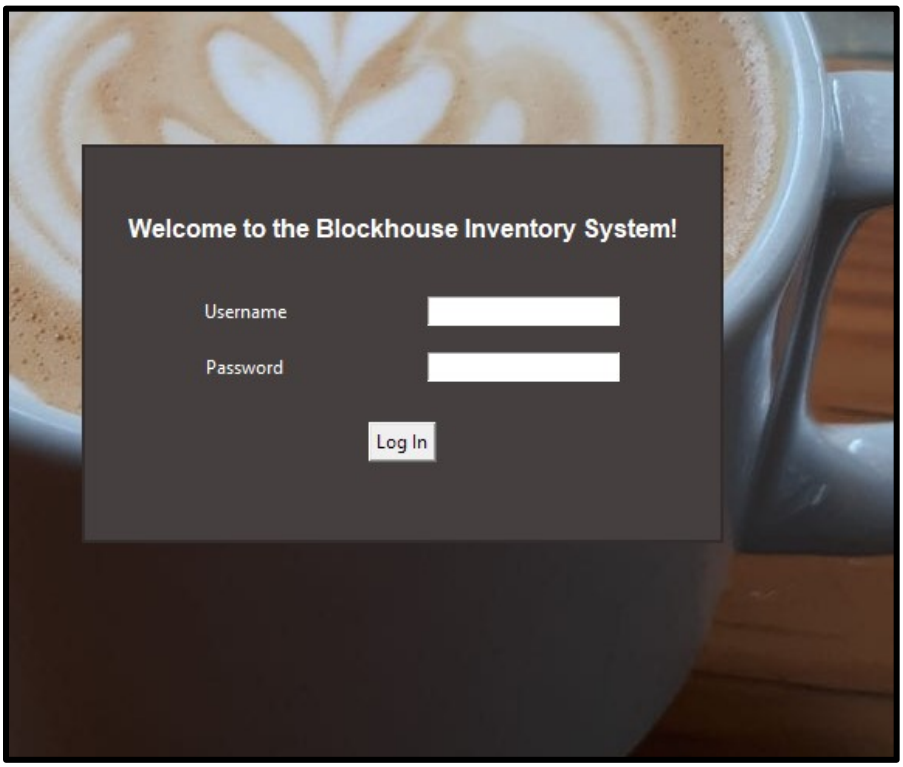
	job_type_id	job_type	job_description
6	6	Barista	Crafts drinks for customers
7	7	Host	Controls flow of customers
8	8	Server	Serves customers at tables
9	9	Busser	Cleans tables and floor
10	10	Dishwasher	Cleans dishes

SCREENSHOTS OF FORMS/GUIs






Ross8976 None
Leach3223 None
Barry7140 None
Valencia4649 None
a new user
Reset selected user's password
Delete selected user
lease enter a new password for user: Ross8976
Change Password
CANCEL


Welcome to the Blockhouse Inventory System!
Username
Password
Log In