Final Project

David Aquino

Timothy Bibo, Jr.

Rohan Dangi

AIT 735, Section 001

Professor Nicolette Carpenter Boddie

December 8, 2014

Table of Contents

Project Proposal

Team Members

David Aquino
Tim Bibo
Rohan Dangi

Proposed Client

In the year 1988, Dan Gordon and Dean Biersch co-founded Gordon Biersch (GB) Company with the goal of creating most authentic German-style larger. With the experience of 25 years, GB has doubled its annual production and increased its capacity to 4 million gallon of beer which made them the largest craft brewery in San Francisco Bay Area (Pulse, 2014). According to Brewers Association, GB ranks in top 49th Breweries in 2013 (Association, n.d.). Currently, there are 34 GB locations around the States that brews 40 different beers (Advocate, n.d). GB uses Ctuit software (also used by its parent company-Craftwork) that combines the restaurant POS data with inventory, accounting, and other tools to form consistent system throughout its brewery (Biersch, n.d.).

Project Description

Our group would like to apply an existing concept to a new industry. Our idea resembles a stock exchange; however, instead of buying stock in a company, customers would be purchasing products to be consumed. In this particular case, customers would be purchasing alcoholic drinks such as beer or cocktails as they normally do in a bar or restaurant. The twist in this scenario is that the prices of these drinks (and possibly food in the future) will fluctuate according to demand and inventory levels. For example, if a group of people were to order 6 Miller Lites, the result after this purchase would be an increase (~ +5%) in the price of Miller Lites and a decrease in another drink/beer such as Bud Light (~ -5%).

We would be introducing a new POS system which would handle sales and relay them to our system. The system would be smart enough to make decisions such as price changes for the drinks. The business need we are addressing is alcohol sales. We predict that with our system, we can help track sales and introduce methods to increase sales. We intend to build a system which will be incorporated with inventory and track sales. In (almost) real-time we would like to adjust the prices of alcohol based on demand. The system will be smart enough to acknowledge trends as they occur and attempt to capitalize on these trends. It will be able to produce on the fly analysis regarding daily highs/lows compared to original price and current price (based on demand). There will also be periodic (daily/weekly/monthly/yearly) snapshots which can be used to analyze sales trends and affect purchasing decisions.

Project Justification

Throughout the planning, design, and implementation of the project, the group members will demonstrate many of the skills that they acquired in their completed coursework, and they will acquire new skills that they will attain through independent methods.

The overall design and implementation of the project will follow the SDLC, a theory of design that has been reinforced in every course of the AIT-MS program at Towson University. Beginning with the project

proposal and culminating in the operational maintenance of the project, the group members will adhere to the SDLC's pillars of planning, designing, building, testing, and delivering a database system that meets the needs of a hypothetical customer. In addition to following the SDLC, the project will incorporate a number of skills learned in AIT-632 and AIT-732, the two prerequisite courses for AIT -735.

AIT-632 and AIT- 732 introduced the group members to a number of database design elements which the group members will incorporate into the project. For example, data modifications will be driven by stored procedures, while considering transaction controls and methods of reinforcing business rules. Also, data will be validated, and all code will be commented in such a way that the instructor (and hypothetically future developers) will be able to quickly and thoroughly understand it. In addition to the aforementioned topics, the group members will broaden their skill-set by learning and developing other database management techniques.

While the project is nascent, the group members have identified a number of skills that they wish to develop as part of this project. At the most basic level, group members will create a new database on an existing Microsoft SQL Server. (There has also been discussion of learning to install and configure SQL Server as a piece of this project). Also, the group members will learn about and implement sufficiently robust user access and permissions. Additionally, the group members will learn about different methods for backing up and restoring SQL Server databases. The group intends to implement a backup strategy for the project.

Project Plan / Schedule

Our group has broken down the project into following tasks:

1. Project Planning: During this phase, our team will provide a system proposal to the client.
2. Requirement Gathering: This section will analyze and determine the needs and expectation of end users of newly developed system. We have decided to utilize the functions available in the existing system within the industry for our initial requirement gathering phase.
3. Design for System: This phase will define the architectural component of the system to satisfy the requirements gathered in the previous stage. Our team will create ER Diagram, Physical Table Layout, and Data Dictionary during this phase.
4. Implementation: This phase includes writing SQL queries, procedure, and triggers.
5. Data Conversion and Loading: Our team will upload a sample data in the system.
6. Testing: During this phase, our team will test if the queries, procedure, and triggers are working as intended or not.
7. Operational Maintenance: During this phase, our team will examine the available solutions for Backup/recovery option during the system failure.

The following table highlights the above mentioned phase with its estimated timeline. This table differs from the finalized schedule, found later in this document.

TASKS	Estimated Timeline	TOTAL HOURS
Task 1. Project Planning	August 29, 2014 – September 01, 2014	
David Aquino		3
Tim Bibo		3
Rohan Dangi		3
Task 2. Requirement gathering	September 08, 2014– September 16, 2014	
David Aquino		20
Tim Bibo		20
Rohan Dangi		20
Task 3. Design for system	September 18, 2014– October 20, 2014	
David Aquino		85
Tim Bibo		85
Rohan Dangi		85
Task 4. Implementation	October 22, 2014– October 27, 2014	
David Aquino		15
Tim Bibo		15
Rohan Dangi		15
Task 5. Data Conversion & Loading	October 29, 2014– October 31, 2014	
David Aquino		4
Tim Bibo		4
Rohan Dangi		4
Task 6. Testing	November 03, 2014– November 14, 2014	
David Aquino		10
Tim Bibo		10
Rohan Dangi		10
Task 7. Operational Maintenance	November 17, 2014– November 19, 2014	
David Aquino		3
Tim Bibo		3
Rohan Dangi		3
TOTAL HOURS TO COMPLETE PROJECT		**420 Hours**
TOTAL HOURS TO COMPLETE PROJECT PER PARTICIPANT		**140 Hours**

Work Cited

- Advocate. (n.d.). Gordon Biersch Brewery Restaurant | United States | Beers. BeerAdvocate. Retrieved September 1, 2014, from http://www.beeradvocate.com/beer/profile/1551/.
- Association. (n.d.). Brewers Association Lists Top 50 Breweries of 2013.brewersassociation.org. Retrieved August 31, 2014, from http://www.brewersassociation.org/attachments/0001/4525/CBP13_Top_50.pdf.
- Biersch. (n.d.). Careers. Growth. Retrieved August 31, 2014, from http://www.gordonbiersch.com/careers/growth.
- Pulse. (2014, May 20). Gordon Biersch Dunkles release marks brewery's 25th anniversary. BeerPulse. Retrieved August 31, 2014, from http://beerpulse.com/2014/05/gordon-biersch-dunkles-release-marks-brewerys-25th-anniversary-3108/.

Executive Summary

Background

GIBINO, (EIN: 55-55697850) is a United States-based professional software development and consulting firm with annual revenue exceeding $1 million. GIBINO has more than 50 employees that provide software development and consulting services in 10 states, primarily on East Coast. In 2013, GIBINO was listed as among the top 100 software companies in the North-East region.

Client Information

Gordon Biersch (GB) was founded in the year 1988 with the goal to create the most authentic German-style larger. In the year 2014, GB was listed as one of the largest craft brewery in San Francisco Bay Area. There are 34 GB locations around the United States brewing 40 different beers. In the year 2010, GB and Rock Bottom restaurants merged to become a part of Craftworks Restaurants and Breweries, Inc. Currently, GB uses Ctuit software, which combines the restaurant POS data with inventory, accounting, and other tools.

Proposed Project

GIBINO proposes to design an improved POS system for GB The need for an improved POS system stems from various reasons, namely:

- Improve inventory management system
- Reduce inefficiencies through automated operations
- Attract new customers (dynamic pricing system)
-

The proposed system is similar to an existing stock exchange concept, where the price of the beer fluctuates based on its demand. For example, if a group of people were to order 6 Hearty Ales, the result after this purchase would be an increase (~ +5%) in the price of Hearty Ales and a decrease in another drink/beer such as Summer Shandy (~ -5%). At predetermined intervals, the total quantity of each beer sales will determine its price.

In addition to dynamic pricing, this system will track sales and maintain a better inventory management system.

Estimated cost

Our group has estimated the cost of implementing this project to be approximately **$56,058.58** and additional there will cost for data restoration services depending upon the data. Total cost of the project is broken down in the section below:

Description	**Cost**
Servers (3 units)	$ 26,953.68
POS Terminals (4 units)	$6,490.46
POS Software License	$ 99.00
Labor	$ 31,500
Backups	$0. 12
Total Cost	**$ 65,043.14+ (0.12 * x GB)**

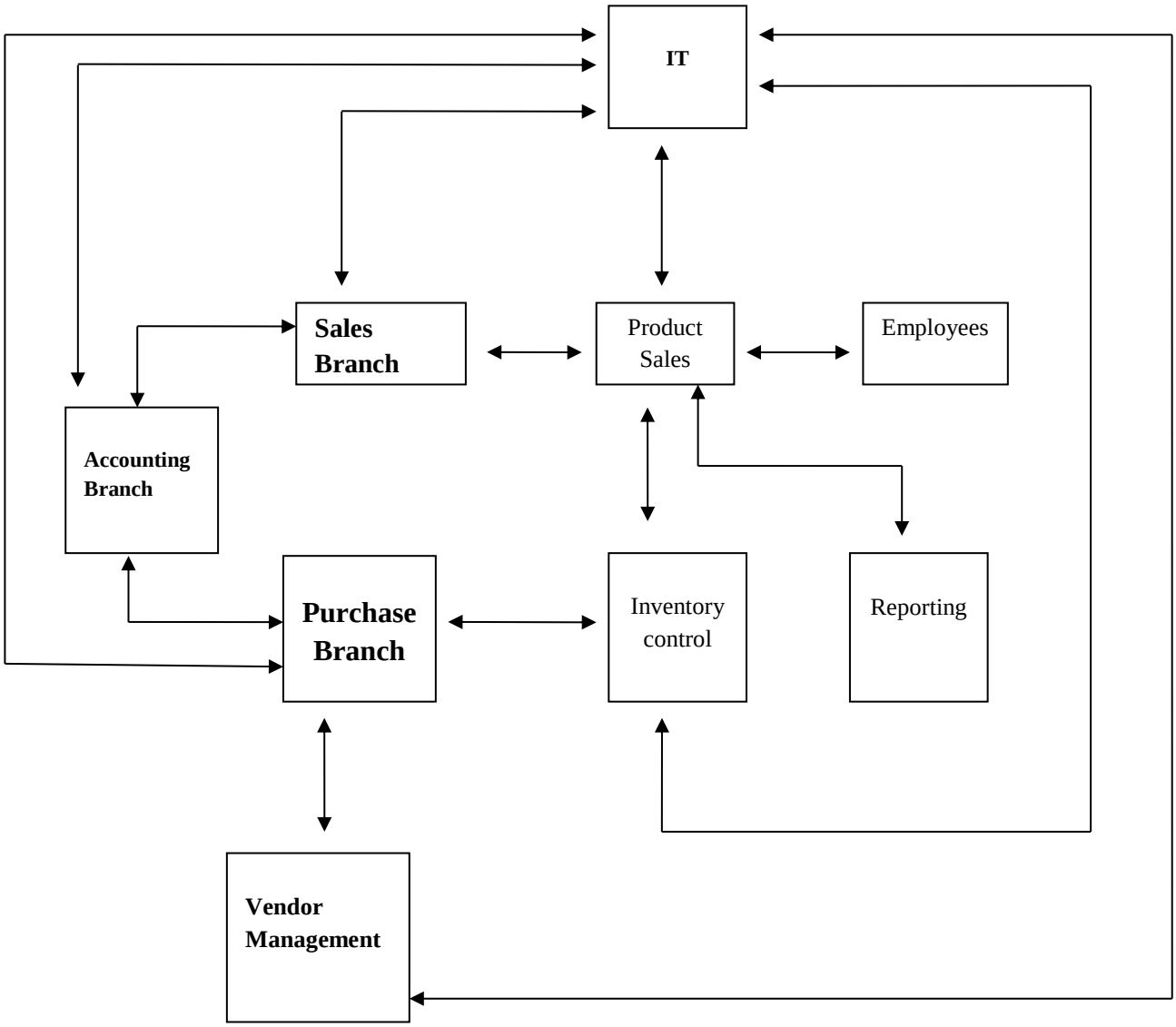
It is acknowledged that if the time frame of the project goes beyond the expected length, there will be significant change in the scope of the project and/or there may be additional requirements added in the later phases of the project. If you have any question, feel free to email us at gibino@ait732.com.

System Definition

Project Scope

This project will consist of creating a database and inventory management system for GB. The project will be completed by December 04, 2014. Our group has recognized the following areas within GB that will be impacted through the implementation of this project.

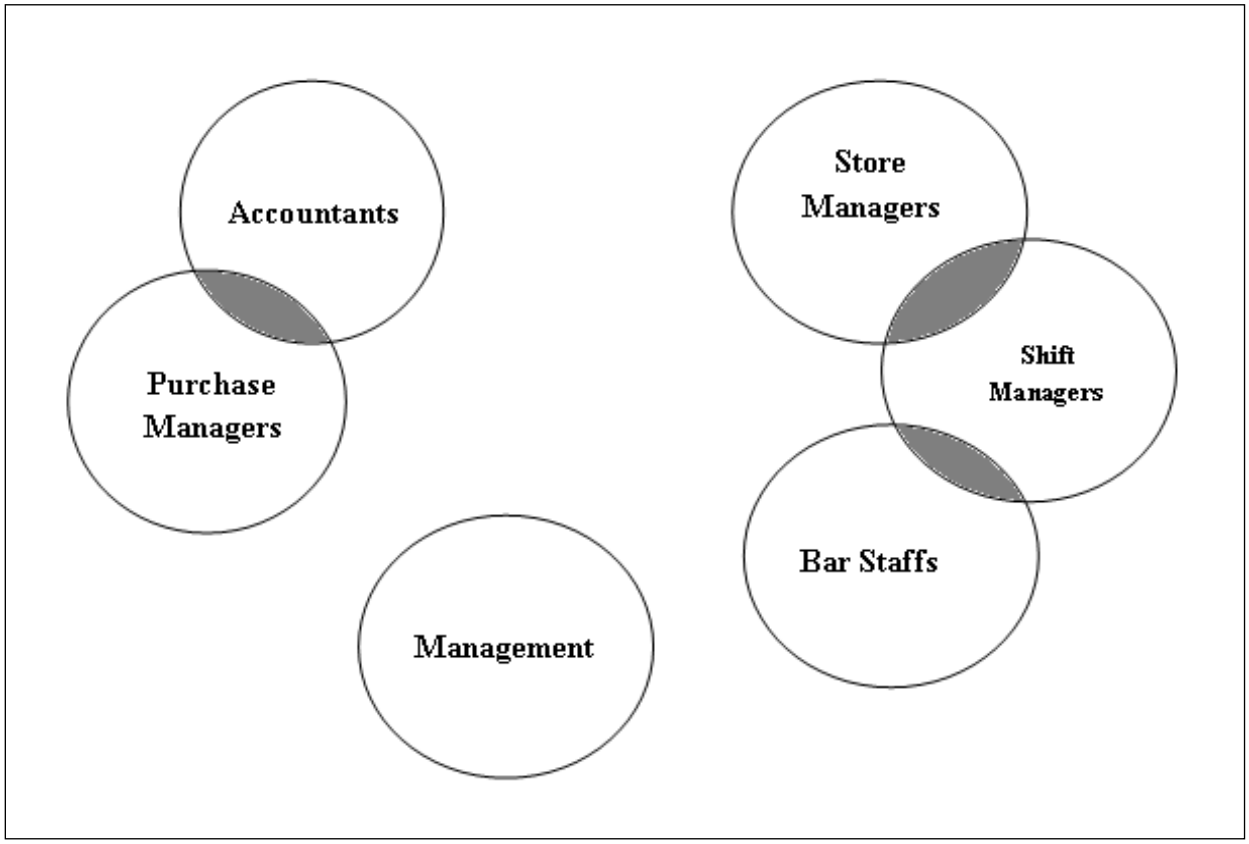
- Sales Branch
- Purchase Branch
- Reporting Group
- IT
- Accounting Department: Accounts Payable/Receivable
- Inventory Management
- Vendor management
- Customer Service



User Views

The primary users of this system include

- bar Staffs
- management
- accountants
- purchasing manager
- shift manager
- store/bar manager



Application Areas

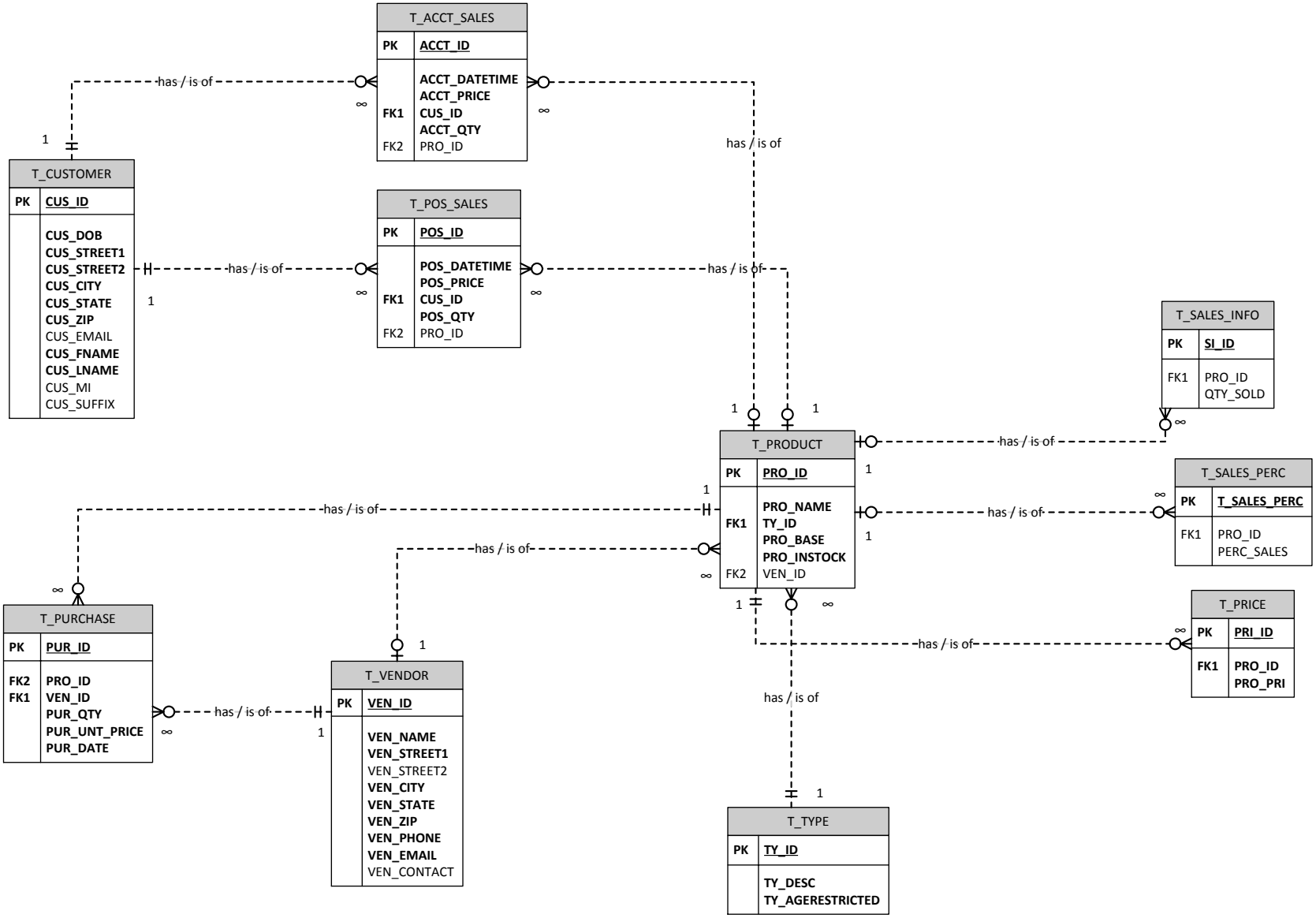
The primary uses of this project include

- backend system to manage purchases from vendors
- frontend POS system to process sales
- visual reporting (via television) to show the current running prices of beers.

System Requirements Specifications

1. The system must encrypt the customers' email addresses..
2. The POS system shall provide a data dump for sales and purchases via ETL process to the accounting software.
3. The system must be able to add new inventory (products) with no duplicate records.
4. The system must be able to add new vendors.
5. The system must be able to update the vendor information.
6. The system must be able to display the entire beverage inventory belonging to a given vendor.
7. The system must produce a formatted report of all products and available quantities for every vendor.
8. The system must display original prices and current prices, % difference in price from current to original, daily high price for products, daily low price for products.
9. The system must display prices for most recently sold products.
10. The system must display top selling products of the day.
11. The system must create a nightly backup, after normal business hours, and differential backups throughout the day.
12. The system must be able to update product prices based on sales.
13. The system must not be able to sell a quantity which exceeds the inventory available, and the system must not be able to sell inventory to a customer below the age of 21.
14. The system must update real-time inventory levels according to sales.
15. The system must be able to display the unpaid items for a customer.
16. The system must be able to predict when to order inventory based on sales and current inventory levels.
17. The system must be able to generate profit report.
18. The system must be able to display total number of inventory sold by type per day.
19. The system must be able to display total revenue for a day.
20. The system must display monthly sales reports.
21. The system must display monthly purchase reports.
22. The system shall keep historical data
23. The system must insert new purchases and update the inventory levels in the product table.
24. The system must reset prices to their base levels.
25. The system must be able to add new product types without duplicates.
26. The system must be able to insert new customers without duplicates.
27. The system must be able to create encrypted ad hoc backups, for transferring across potentially insecure connections.

Entity-Relationship Diagram



Physical Layout

TABLE	VARIABLE	DATA_TYPE	NOT_NULL	PK	FK	IDENTITY	CHECK	UNIQUE	DESCRIPTION	SAMPLE
T_VENDOR	VEN_ID	INT	Y	Y		Y		Y	UNIQUE ID FOR EACH VENDOR	1
	VEN_NAME	VARCHAR(30)	Y					Y	VENDOR NAME	WM BREWERY
	VEN_STREET1	VARCHAR(30)	Y						VENDOR ADDRESS ONE	123 FAKE STREET
	VEN_STREET2	VARCHAR(30)							VENDOR ADDRESS TWO	UNIT X
	VEN_CITY	VARCHAR(30)	Y						CITY	WHITE MARSH
	VEN_STATE	CHAR(2)	Y						STATE	MD
	VEN_ZIP	NUMERIC(5,0)	Y				>=01000<=99999		U.S. ZIP CODE	21245
	VEN_PHONE	VARCHAR(12)	Y						PHONE NUMBER OF VENDOR	410-111-1111
	VEN_EMAIL	VARCHAR(50)	Y						EMAIL OF CONTACT AT VENDOR	JJ@WMBREW.COM
	VEN_CONTACT	VARCHAR(50)	Y						NAME OF CONTACT AT VENDOR	JANET JONES
T_PRODUCT	PRO_ID	INT	Y	Y		Y		Y	UNIQUE ID FOR EACH PRODUCT	1
	PRO_NAME	VARCHAR(30)	Y					Y	DESCRIPTIVE NAME OF PRODUCT	SPICY SUMMER ALE
	TY_ID	INT	Y		Y				REFERENCE TYPE (TY_ID)	5
	PRO_BASE	NUMERIC(5,2)	Y				>0		BASE PRICE OF THE PRODUCT	$5.00
	PRO_INSTOCK	INT	Y				>=0		NUMBER OF PRODUCT IN STOCK	
	VEN_ID	INT	Y		Y				REFERENCE VENDOR (VEN_ID)	546
T_TYPE	TY_ID	INT	Y	Y		Y		Y	UNIQUE ID FOR EACH PRODUCT CATEGORY	1
	TY_DESCRIPTION	VARCHAR(50)	Y					Y	TYPE OF PRODUCT	16 OZ BEER BOTTLE
	TY_AGERESTRICTED	INT	Y				IN(1,2)		MUST CUSTOMER BE 21 TO PURCHASE?	1=YES / 2=NO
T_PURCHASE	PUR_ID	INT	Y	Y		Y		Y	UNIQUE ID FOR EACH PURCHASE	1
	PRO_ID	INT	Y		Y				REFERENCE PRODUCT (PRO_ID)	33
	VEN_ID	INT	Y		Y				REFERENCE VENDOR (VEN_ID)	12
	PUR_QTY	INT	Y				>0		QUANTITY PURCHASED	480
	PUR_UNT_PRICE	NUMERIC(5,2)	Y				>0		PRICE PAID PER UNIT	$1.25
	PUR_DATE	DATETIME	Y						DATE AND TIME OF PURCHASE	8/15/2015:15:32:42

TABLE	VARIABLE	DATA_TYPE	NOT_NULL	PK	FK	IDENTITY	CHECK	UNIQUE	DESCRIPTION	SAMPLE
T_ACCT_SALES	ACCT_ID	INT	Y	Y		Y		Y	UNIQUE ID FOR EACH ITEM IN THE CART	1
	ACCT_DATETIME	DATETIME	Y						DATE AND TIME OF PURCHASE	8/15/2015:15:32:42
	ACCT_PRICE	NUMERIC(5,2)	Y				>0		PRICE AT WHICH PRODUCT WAS SOLD	$1.87
	CUS_ID	INT	Y		Y		>0		REFERENCE CUSTOMER (CUS_ID)	133
	ACCT_QTY	INT	Y				>0		QTY SOLD	
	PRO_ID	INT	Y		Y				REFERENCE PRODUCT (CUS_ID)	2
T_PRICE	PRI_ID	INT	Y	Y		Y		Y	UNIQUE ID FOR EACH PRICE CHANGE	4876
	PRO_ID	INT	Y						REFERENCE PRODUCT (PRO_ID)	33
	PRO_PRI	NUMERIC(5,2)	Y				>0		CURRENT SELLING PRICE OF PRODUCT	$1.87
T_CUSTOMER	CUS_ID	INT	Y	Y		Y			UNIQUE ID FOR EACH CUSTOMER	21
	CUS_DOB	DATE	Y						CUSTOMER DOB	9/12/1990
	CUS_STREET1	VARCHAR(30)							CUSTOMER ADDRESS ONE	123 FAKE STREET
	CUS_STREET2	VARCHAR(30)							CUSTOMER ADDRESS TWO	UNIT X
	CUS_CITY	VARCHAR(30)							CITY	WHITE MARSH
	CUS_STATE	CHAR(2)							STATE	MD
	CUS_ZIP	NUMERIC(5,0)					>=01000<=99999		U.S. ZIP CODE	21245
	CUS_EMAIL	VARCHAR(50)							CUSTOMER EMAIL ADDRESS	JAKE@OLDMAN.COM
	CUS_FNAME	VARCHAR(25)							CUSTOMER FIRST NAME	JAKE
	CUS_LNAME	VARCHAR(30)							CUSTOMER LAST NAME	OLDMAN
	CUS_MI	VARCHAR(1)							CUSTOMER MIDDLE INITIAL	R
	CUS_SUFFIX	VARCHAR(5)							CUSTOMER NAME SUFFIX	III
T_SALES_INFO	SI_ID	INT	Y	Y		Y		Y	UNIQUE ID USED FOR SETTING PRICES	3
	PRO_ID	INT	Y		Y				REFERENCES PRO_ID IN PRODUCT TABLE	9
	QTY_SOLD	INT	Y						QTY SOLD	12
T_SALES_PERC	SP_ID	INT	Y	Y		Y		Y	UNIQUE ID FOR EACH SALES PERCENTAGE	34
	PRO_ID	INT	Y		Y				REFERENCES PRO_ID IN PRODUCT TABLE	9

TABLE	VARIABLE	DATA_TYPE	NOT_NULL	PK	FK	IDENTITY	CHECK	UNIQUE	DESCRIPTION	SAMPLE
	PERC_SALES	INT	Y		Y				PERCENT * 100	44

T_POS_SALES	POS_ID	INT	Y	Y		Y		Y	UNIQUE ID FOR EACH ITEM IN THE CART	1
	POS_DATETIME	DATETIME	Y						DATE AND TIME OF PURCHASE	8/15/2015:15:32:42
	PRO_PRICE	NUMERIC(5,2)	Y				>0		PRICE AT WHICH PRODUCT WAS SOLD	$1.87
	CUS_ID	INT	Y		Y		>0		REFERENCE CUSTOMER (CUS_ID)	133
	POS_QTY	INT	Y				>0		QTY SOLD	
	PRO_ID	INT	Y		Y				REFERENCE PRODUCT (CUS_ID)	2

Data Dictionary

Note: The following format is utilized to illustrate the entities and descriptive information.

Entity Name		
Field Designator	Type	Size
Required/Optional – Definition		

T_VENDOR

Vendor ID (Primary Key)		
ven_id	INT (Auto-Generated)	
Required – The ven_id is a unique number that all the vendors should have which is assigned during the first time transaction between the vendor and the organization.		

Vendor Name		
ven_name	VARCHAR	30
Required – Name of the vendor. For example, Towson Brewery		

Street1 Name		
ven_street1	VARCHAR	30
Required – The street name of the Vendor's address.		

Street2 Name		
ven_street2	VARCHAR	30
Optional – If the Vendor's address requires a second line.		

City		
ven_city	VARCHAR	30
Required – The city of the Vendor's address.		

State		
ven_state	CHAR	2

Required – The two character state code of the vendor's address

Zip Code		
ven_zip	NUMERIC	5,0
Required – The zip code of the vendor.		

Vendor's Phone		
ven_phone	VARCHAR	12
Required – Phone number of the Vendor including dash delimiters. For example: "410-458-8774".		

Vendor's Email Address		
ven_email	VARCHAR	50
Required – Email Address of the Vendor		

Vendor's Contact Person		
ven_contact	VARCHAR	50
Required – First name & last name of contact person at Vendor		

T_PRODUCT

Product ID (Primary Key)
pro_id INT(Auto-Generated)
Required- The pro_id is a unique number for products.

Name of Product		
pro_name	VARCHAR	30
Required – Name given to the product		

Type ID of the Product (Foreign Key-References T_TYPE Table)
ty_id INT
Required- This is a type id from the T_Type table.

Price of the Product		
pro_base	NUMERIC	5,2
Required – This is the base price of the product which has to be >0		

Vendor ID (Foreign Key- References T_VENDOR Table)		
ven_id	INT	10
Required – The ven_id is a unique number that all the vendors should have which is assigned during the first time transaction between the vendor and the organization. See T_Vendor Table.		

Products in Stock	
pro_instock	INT
Required – Total number of products in stock must be >=0	

T_TYPE

Type ID (Primary Key)		
type_id	INT	
Required – This is a unique ID given to a product based on its type		

Type Description		
ty_description	VARCHAR	50
Required – Type description of the product such as 12/16 Oz Beer can, 8.4/20 Oz Energy Drink, etc.		
Age Restriction		
ty_restricted	Int	1
Required – Ty_Restricted can be either “0” or “1” where “1” denotes as Alcohol drink requiring to be 21 to purchase whereas, “0” denotes as non-alcoholic drink.		

T_PURCHASE

Purchase ID (Primary key)
pur_id INT (Auto-Generated)
Required – This is a unique ID given for each purchase from a vendor.

Product ID (Foreign Key- References T_PRODUCT Table)
pro_id INT
Required- The pro_id is a unique number for products. See T_PRODUCT Table.

Vendor ID (Foreign Key- References T_VENDOR Table)
ven_id INT
Required – The ven_id is a unique number that all the vendors should have which is assigned during the first time transaction between the vendor and the organization. See T_Vendor Table.

Purchased Quantity
pur_qty INT
Required – Total number of quantities of product purchased which has to be >0.

Unit Price
Pur_Unt_Price Numeric 5,2
Required – This is price paid per unit of product.

Date of Purchase
Pur_Date Datetime 8
Required – This is date and time of the day when purchase occurred

T_ACCT_SALES

Accounting ID (Primary key)
acct_ID INT(Auto-Generated)
Required – This is a unique ID for sales. The Accounting Sales table is permanent and is used to track historical data.

Date of Sales
acct_datetime DATETIME
Required – This is date and time of the day when sales occurred.

Price of Sold Price		
acct_price	NUMERIC	5,2
Required – This is the price at which the product was sold.		

Customer ID (Foreign key- Reference T_CUSTOMER Table)
cus_ID INT
Required – This is a unique ID given for each customer. See T_CUSTOMER table

Sold Quantity	
acct_qty	INT
Required – Total number of quantities of product sold which has to be >0.	

Product ID (Foreign Key- References T_PRODUCT Table)
pro_id INT
Required- The pro_id is a unique number for products. See T_PRODUCT Table

T_PRICE

Price ID (Primary key)
pri_id INT (Auto-Generated)
Required – This is a unique ID for the price table.

Product ID (Foreign Key- References T_PRODUCT Table)
pro_id INT
Required- The pro_id is a unique number for products. See T_PRODUCT Table

Current Price		
pro_price	NUMERIC	5,2
Required – This is the current price of the product.		

T_CUSTOMER

Customer ID (Primary Key)

cus_id	INT	
Required – This is a unique ID given for each customer.		

Customer's Date of Birth		
cus_dob	DATETIME	
Required- Date of Birth is needed to calculate customer age and determine eligibility to purchase products.		

Customer's Address Street 1		
cus_street1	VARCHAR	30
Optional – Customer's street address line 1.		

Customer's Address Street2		
cus_street2	VARCHAR	30
Optional – Customer's street address line 2.		

Customer's Address City		
cus_city	VARCHAR	30
Optional – The city of the Customers's address.		

Customer's State		
cus_state	CHAR	2
Optional – The two character state code of the Customer's address.		

Zip Code		
cus_zip	NUMERIC	5,0
Optional – The zip code of the Customer's Address.		

Customer's Email Address		
cus_email	VARCHAR	50

Optional – Email Address of the Customer

Customer's First Name		
cus_fname	VARCHAR	25
Optional – First name of the customer		

Customer's Last Name		
cus_lname	VARCHAR	30
Optional – Last name of the customer		

Middle Initial		
cus_mi	VARCHAR	1
Optional – Middle initial of the customer		

Customer Name Suffix		
cus_suffix	VARCHAR	5
Optional – Customer Name Suffix such Jr., Sr. Ph.D., II, III, IV, V, etc.		

T_SALES_INFO

Sales ID (Primary Key)	
si_id	INT (Auto-generated)
Required – This is a unique ID for this helper table.	

Product ID (Foreign Key- References T_PRODUCT Table)	
pro_id	INT
Required- The pro_id is a unique number for products. See T_PRODUCT Table.	

.

Quantity	
qty_sold	INT
Required – Total number of product sold.	

T_SALES_PERC

Sales Percentage ID (Primary Key)	
sp_id	INT (Auto-Generated)
Required – This is a unique ID for this helper table.	

Product ID (Foreign Key- Reference T_PRODUCT Table)	
pro_id	INT
Required- The pro_id is a unique number for products. See T_PRODUCT Table.	

Percentage Sold	
pct_of_sales	INT
Required- Percentage of sales for products sold.	

T_POS_SALES

POS ID (Primary Key)	
pos_id	INT (Auto-generated)
Required – This is a unique ID given for sales at the pos terminal.	

Date/Time of Sale	
pos_datetime	DATETIME
Required – This is date and time of the sale at the pos terminal.	

Price		
pro_price	NUMERIC	5,2
Required – This is the price at which the product was sold.		

Customer ID (Foreign key- References T_CUSTOMER Table)	
cus_id	INT
Required – This is a unique ID given for each customer. See T_CUSTOMER Table	

POS Paid	
pos_paid	INT

Required – 0 for unpaid, 1 for paid

Quantity		
pos_qty	INT	
Required – Number products sold which has to be >0.		

Product ID (Foreign Key- References T_PRODUCT Table)	
pro_id	INT
Required- The pro_id is a unique number for products. See T_PRODUCT Table	

T_PRICE_DIFF

Price Differential ID	
diff_id	INT
Required- The diff_id is a unique number for this helper table	

Product ID (Foreign Key- References T_PRODUCT Table)	
pro_id	INT
Required- The pro_id is a unique number for products. See T_PRODUCT Table	

Percentage Difference		
diff_perc	DECIMAL	5,1
Required- The percentage difference between current price of a product and it's base price.		

T_CUS_AGE

Customer Age ID	
cage_id	INT
Required- The cage_id is a unique number this helper table.	

Customer ID (Foreign Key- References T_Customer Table)	
pro_id	INT

Required- The pro_id is a unique number for products. See T_PRODUCT Table

Age
age INT
Required- Age is determined by a date diff from the customer date of birth.

System Architecture

We have divided the hardware & software requirements for the system into two parts. One part will be used for maintaining POS activities and the other part will be used to support the database functionality.

The requirements POS activities include:

- Computer workstation
- Touch screen monitor
- Credit card stripe reader
- Printer (for receipts and reports)
- POS (PHP Point of Sale) software for each workstation
- Network routers
- Battery backup
- Automated cash drawer
- Supported Operating Systems: Windows POS Ready 7 Windows 7 Professional, and Windows 8 Pro
- CPU: 1.5 GHz or Better Intel Based
- RAM: 1 GB or More
- Hard Drive: 16 GB or More Free Space
- RAID 1
- NIC: 100 Mbit or 1 gigabit

Requirements for the database activities include:

- 2 x Dell Poweredge T420 Servers
- Microsoft Windows Server 2012
- Microsoft SQL Server 2012 5-User + 5-Pack of Device CAL
- Dual Intel Xeon E5-2420 Processors
- 8GB RDIMM RAM
- RAID 1 with PERC H310 Controller
    - 1TB 7200 RPM, Hot-Swap Drives
- Powervault RD100 Backup System
- 5 Years on-site support

GB will have a powerful, stable and flexible system that will allow performing point of sale, cash management, customer and resource management. The overall technical IT architecture is divided into in-store and enterprise architecture. The in-store infrastructure includes POS systems running 64-bit Windows POS Ready 7 operating system with 16 GB RAM, Intel core i7-2630QM CPU @ 2.00GHz, and 500 GB HDD on each workstation. These systems are used for processing sales transactions such as placing and processing orders, viewing customer orders, managing daily operations and inventory, or viewing role-based reports like shift report etc.

To carry out these activities POS systems also have application/software, touch screen monitor, card reader, receipt/report printer, and automated cash drawer. These systems do not do the data processing; however, they transmit data to the local database server, which does the heavy data processing activities.

There are two database servers -Windows Server 2012, configured with RAID 1 in each store. One is the primary database server (POS1), and the other is a mirrored server (POS2). A third server (WIT1) acts as a witness server. In the event of POS1 failing, WIT1 will initiative an automatic failover. POS1 will be marked as disconnected, and all clients will be connected from the database. POS2 will come online and will then roll forward any logs in the database and will rollback any uncommitted transactions.

Every morning, at 05:00, the system uploads the full database backup to the remote storage system Amazon Web Services (AWS). The backup compression options keep the uploaded files manageable. The system uploads incremental backups to AWS at 20:00 and 01:00.

The in-store network uses both Ethernet cables and Wireless network technology and is connected to a computer on the Headquarters Intranet.

There are daily full-backups, with incremental backups occurring throughout the day. These backups are stored locally and are transmitted to a remote backup system (Amazon's S3 service).

GB accepts cash, debit cards and credit card as a payment method. To process the card payments, GB has decided to use MegaPath's services. MegaPath Corporation is a business telecommunication company that provides voice, private network, data security, cloud services to the retail industry. This external payment processor is complied with Payment Card Industry (PCI) requirement for data security which allows GB a private network connection to a wide variety of payment processors such as debit & credit cards.

The payment process follows this sequence:

- Megapath's Payment Processor Extranet connects with GB's MPLS VPN
- It securely transfers the card payment (debit & credit card information) from GB store location to payment processors
- All the back-end functions of routing the card information to banks for validating customer's available fund are done at MegaPath's end
- Depending on the message provided from Bank to MegaPath Payment processor, the transaction is either approved or declined

Project Plan / Schedule

TASKS	Estimated Timeline	TOTAL HOURS
Task 1. Project Planning	August 29, 2014 – September 01, 2014	
David Aquino		3
Tim Bibo		3
Rohan Dangi		3
Task 2. Requirement gathering	September 08, 2014– September 16, 2014	
David Aquino		20
Tim Bibo		20
Rohan Dangi		20
Task 3. Design for system	September 18, 2014– October 20, 2014	
David Aquino		35
Tim Bibo		35
Rohan Dangi		35
Task 4. Implementation	October 22, 2014– October 27, 2014	
David Aquino		25
Tim Bibo		25
Rohan Dangi		25
Task 5. Testing	October 28, 2014– November 28, 2014	
David Aquino		54
Tim Bibo		54
Rohan Dangi		54
Task 6. Operational Maintenance	November 29, 2014– December 11, 2014	
David Aquino		3
Tim Bibo		3
Rohan Dangi		3
TOTAL HOURS TO COMPLETE PROJECT		**420 Hours**
TOTAL HOURS TO COMPLETE PROJECT PER PARTICIPANT		**140 Hours**

Test Cases

1 – The system must encrypt the customers' email addresses.

This code encrypts customer emails using the triple DES algorithm.

```
--Select * from T_customer
IF NOT EXISTS
    (SELECT * FROM sys.symmetric_keys WHERE symmetric_key_id = 101)
    CREATE MASTER KEY ENCRYPTION BY
    PASSWORD = 'password123'
GO

--create certificate
CREATE CERTIFICATE Customer_email
   WITH SUBJECT = 'Protect Customer Email';
GO

CREATE SYMMETRIC KEY CustemailKey1
    WITH ALGORITHM = TRIPLE_DES_3KEY
    ENCRYPTION BY CERTIFICATE Customer_email;
GO

-- Create a column in which to store the encrypted data.
ALTER TABLE T_customer
    ADD EncryptedEmail varbinary(max);
GO


-- Open the symmetric key with which to encrypt the data.
OPEN SYMMETRIC KEY CustemailKey1
   DECRYPTION BY CERTIFICATE  Customer_email;

-- update the column with Encrypted value

UPDATE T_customer
SET EncryptedEmail = EncryptByKey(Key_GUID('CustemailKey1'), cus_email);
GO

-- To verify the encryption - open the symmetric key with which to decrypt the data.
OPEN SYMMETRIC KEY CustemailKey1
   DECRYPTION BY CERTIFICATE Customer_email;
GO


--To verify if decryption works, select original data, encrypted data,and Decrypted data
SELECT cus_email, EncryptedEmail
    AS 'Encrypted Cus_email',
    CONVERT(varchar, DecryptByKey( EncryptedEmail))
    AS 'Decrypted Cus_email'
    FROM t_customer;
GO
```

This screenshot shows both the encrypted and decrypted versions of the customers' email addresses.

```
37      DECRYPTION BY CERTIFICATE Customer_email;
38  GO
39
40
41  --To verify if decryption works, select original data, encrypted data,and Decrypted data
42  SELECT cus_email, EncryptedEmail
43      AS 'Encrypted Cus_email',
44      CONVERT(varchar, DecryptByKey( EncryptedEmail))
45      AS 'Decrypted Cus_email'
46      FROM t_customer;
47  GO
```

100 %

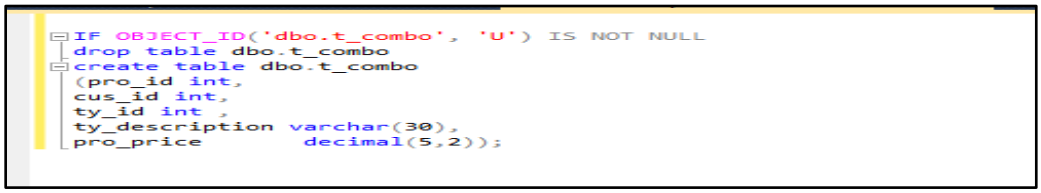
Results | Messages

	cus_email	Encrypted Cus_email	Decrypted Cus_email
1	joesmith@gmail.com	0x0058511E9F60AA4190CEF5CDF3010D3B01000000E7F9F1A...	joesmith@gmail.com
2	samdan@yahoo.com	0x0058511E9F60AA4190CEF5CDF3010D3B01000000F930450...	samdan@yahoo.com
3	Mking@gmail.com	0x0058511E9F60AA4190CEF5CDF3010D3B010000006B3AD6...	Mking@gmail.com
4	amyclinton@gmail.com	0x0058511E9F60AA4190CEF5CDF3010D3B01000000718666F...	amyclinton@gmail.com
5	Clintonb@ymail.com	0x0058511E9F60AA4190CEF5CDF3010D3B01000000D9E97C...	Clintonb@ymail.com
6	Mike	0x0058511E9F60AA4190CEF5CDF3010D3B010000001EBB468...	Mike
7	m.posmer@gmail.com	0x0058511E9F60AA4190CEF5CDF3010D3B01000000D27B069...	m.posmer@gmail.com
8	DillSmith@ymail.com	0x0058511E9F60AA4190CEF5CDF3010D3B01000000A78E0B0...	DillSmith@ymail.com

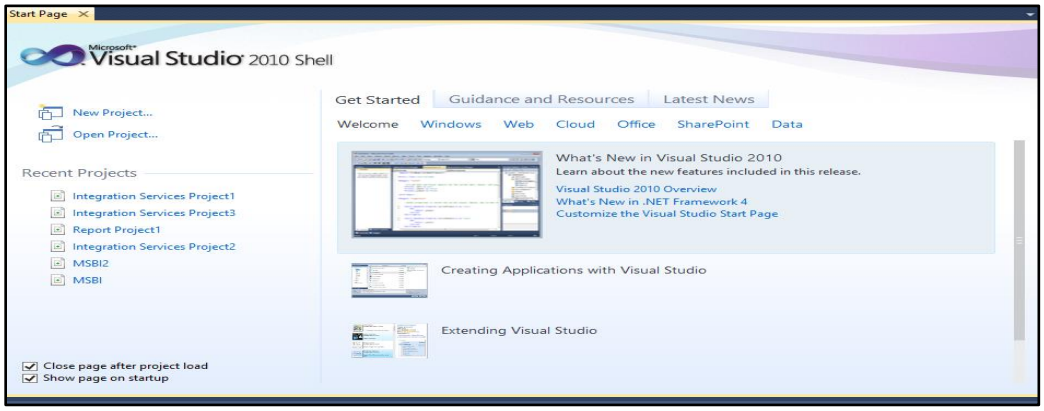
2 - The POS system shall provide a data dump for sales and purchases via ETL process to the accounting software.

This code and the following steps establish an ETL to extract sales data and prepare it for use with accounting software.

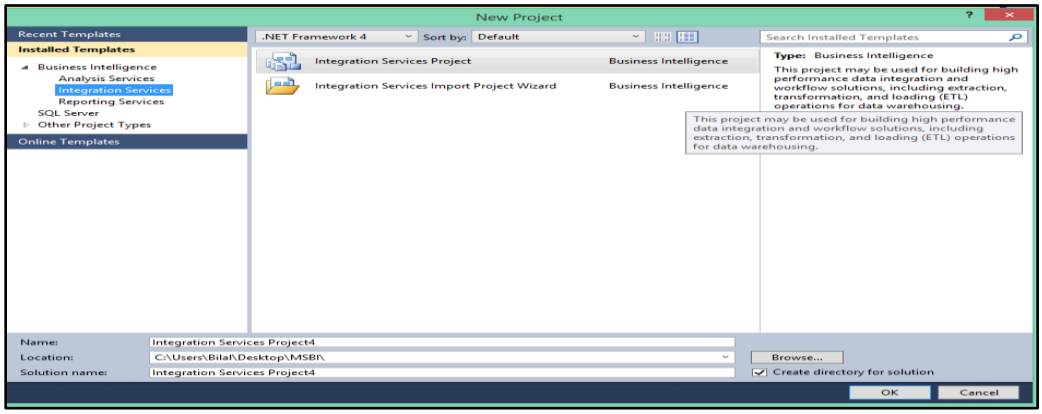
```
--SSIS Integration Service
--Create a table to transfer the data
IF OBJECT_ID('dbo.t_combo', 'U') IS NOT NULL
drop table dbo.t_combo
create table dbo.t_combo
(pro_id int,
cus_id int,
ty_id int ,
ty_description varchar(50),
pro_price           decimal(5,2));

--Following data needs to be transferred
Select ts.pro_id,ts.cus_id,tp.ty_id,tt.ty_description,ts.pro_price from
T_pos_sales ts join T_product tp on ts.pro_id=tp.pro_id join t_type tt on tp.ty_id=tt.ty_id
```



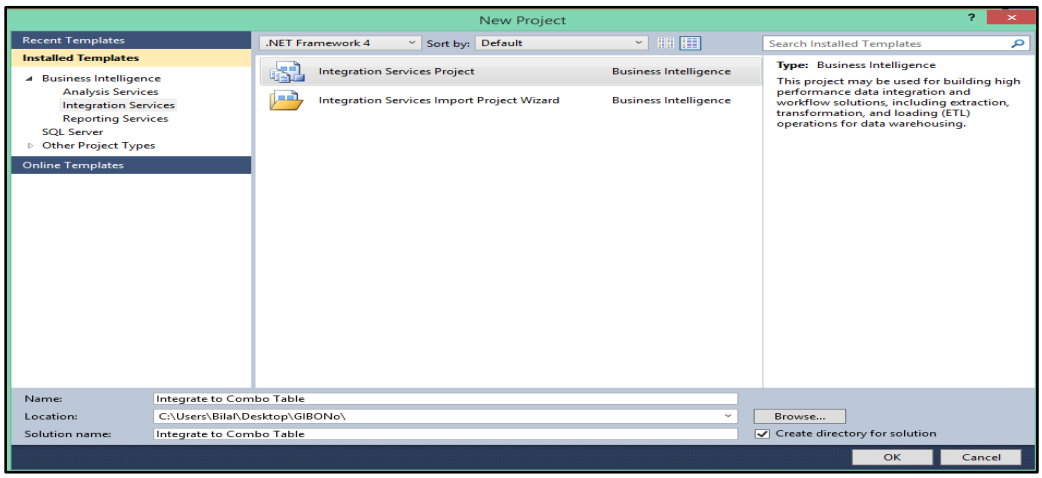
a. Select new project



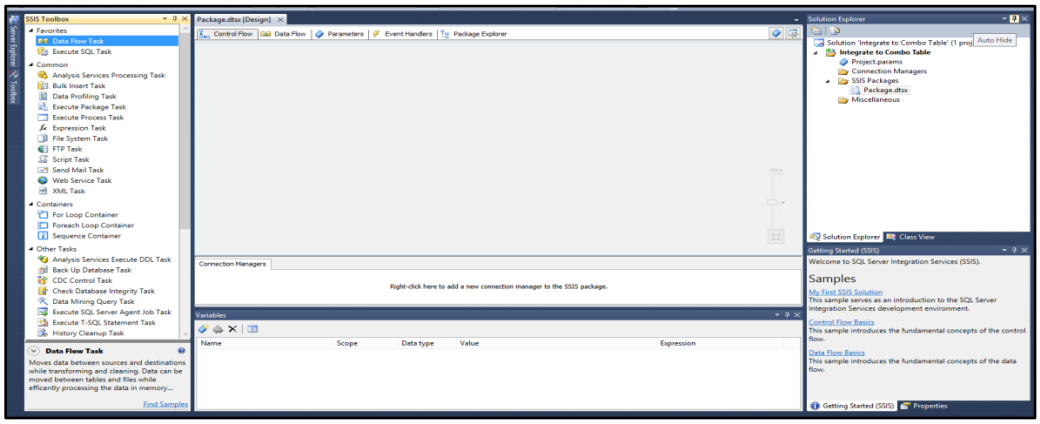
b. Integration service



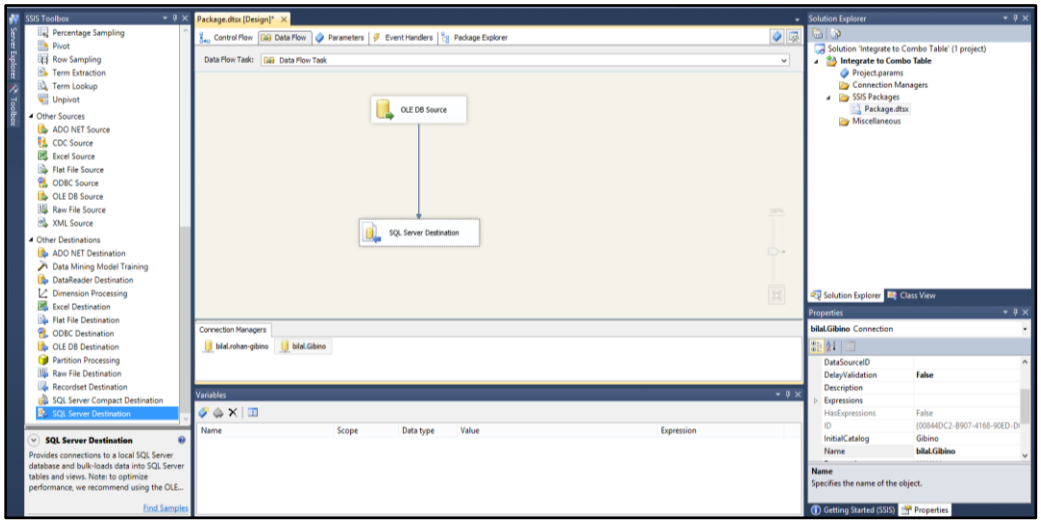
c. Give it a name and save



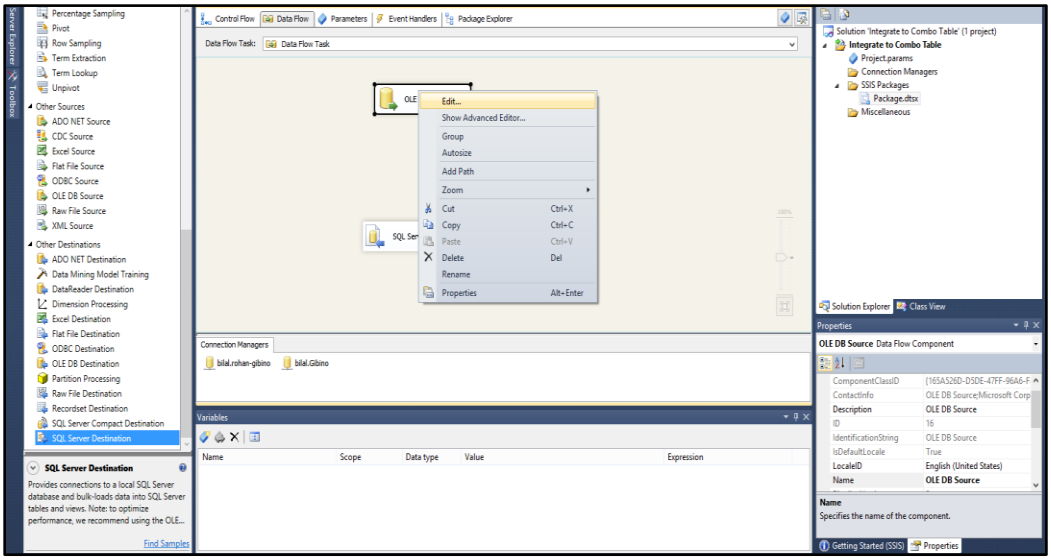
d. Drag data flow in control flow window:



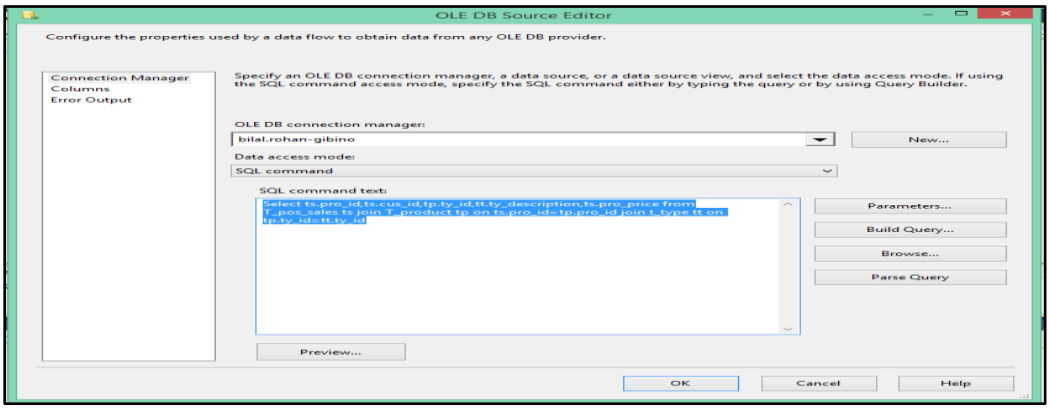
e. Connecting sources database with source assistant:



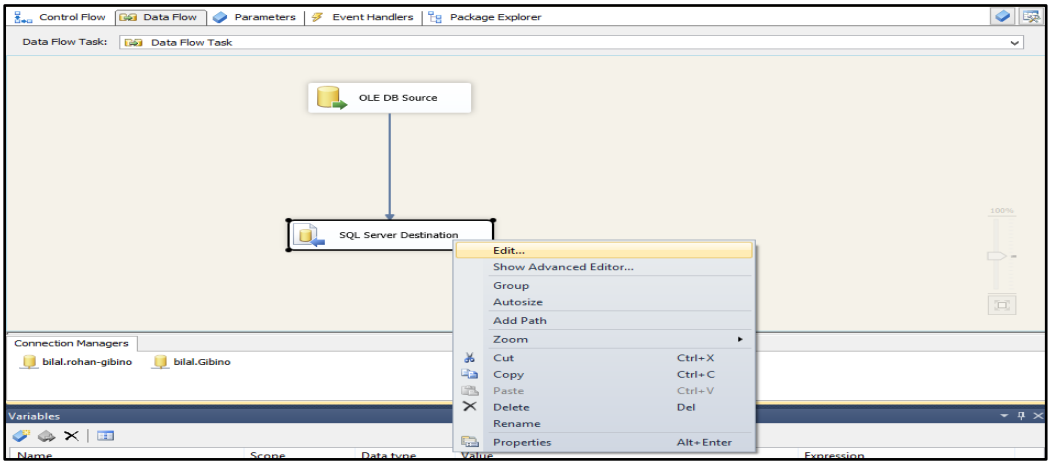
f. Adding source connection:



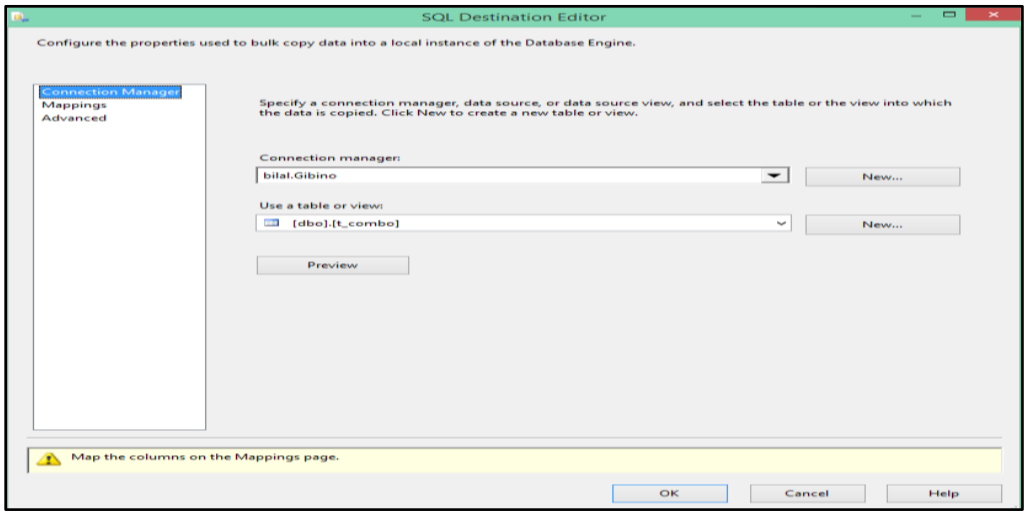
-Then



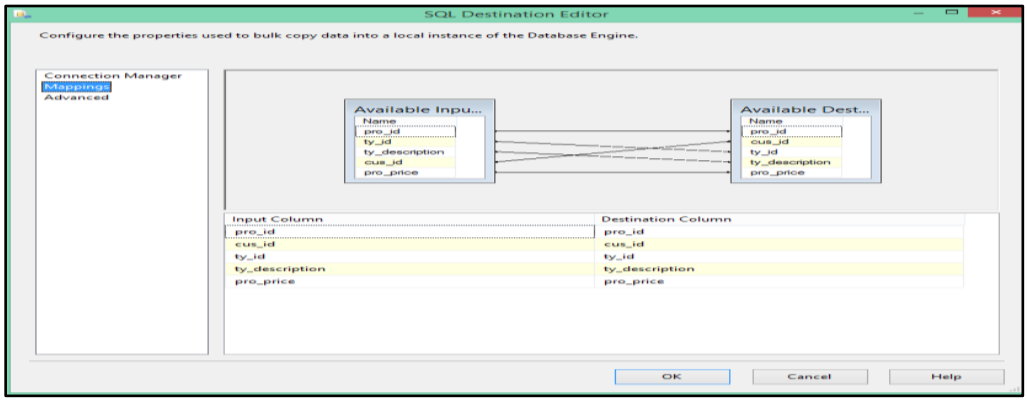
g. Connect with destination database by using SQL server destination assistant



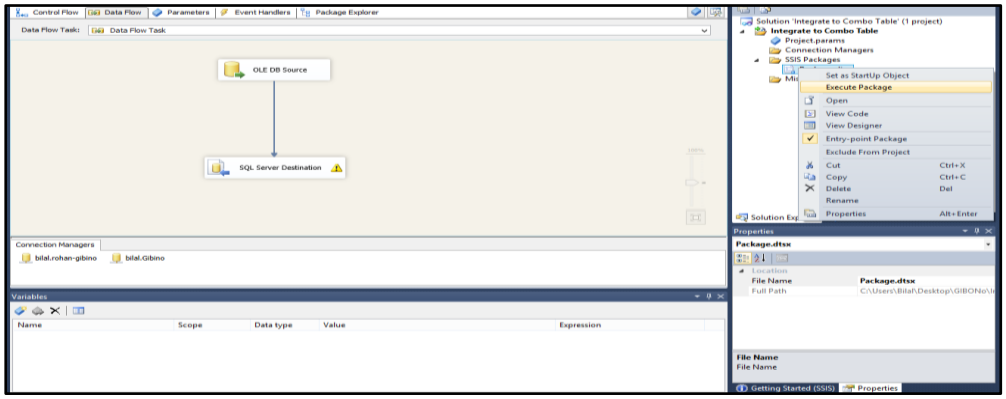
h. After edit: select the name of database



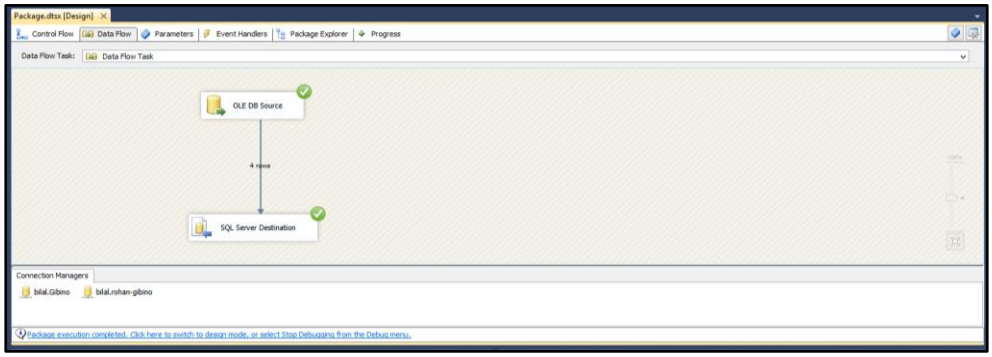
i. Mapping the table: source table to destination table



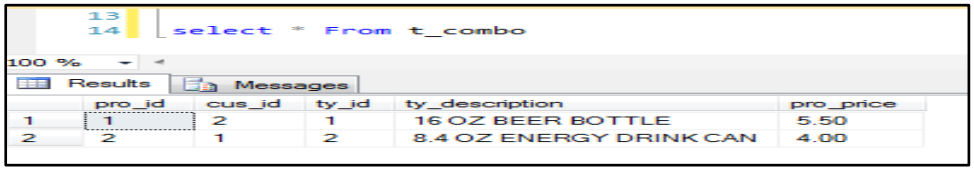
j. To run this integration



- After execution



-Data is transferred to combo:



```
select * from t_combo
```

100 %

Results | Messages

	pro_id	cus_id	ty_id	ty_description	pro_price
1	1	2	1	16 OZ BEER BOTTLE	5.50
2	2	1	2	8.4 OZ ENERGY DRINK CAN	4.00

3 - The system must be able to add new inventory items (products) with no duplicate records.

The following code creates a stored procedure to add new items. It validates that the product does already exist in the system and that the product does have a valid type.

```
use gibino
if objectproperty(object_id('dbo.sp_add_product'), N'IsProcedure') = 1
drop procedure dbo.sp_add_product
go

create procedure sp_add_product
(@pro_name varchar(30), @pro_instock int, @ty_id int, @pro_base numeric(5,2), @ven_id int)
as
begin

--Checks for duplicate product names. Both differing cases and spacing are accommodated.
if exists (select * from t_product where upper(replace(pro_name, ' ',
''))=upper(replace(@pro_name, ' ', '')))
	begin
		select @pro_name, 'already exists in this system as a product.'
		return
end
else

--Checks that type is valid
if not exists (select * from t_type where ty_id=@ty_id)
	begin
		select @pro_name, 'was not given a valid type.'
		return
end

--Inserts product data into t_product table
begin transaction
	 insert into t_product
	(pro_name, pro_instock, ty_id, pro_base, ven_id)
	 values
	(@pro_name, @pro_instock, @ty_id, @pro_base, @ven_id)

--Message for rollback
if @@error<>0
	begin
		rollback transaction
		select 'Product ', @pro_name, ' not added'
		return
	end

--Message for success
commit transaction
select   'Product ', @pro_name, ' was added'
end
```

This screen shot demonstrates the successful insertion of a product.

```
--Adds a valid product to the system / should run OK
use gibino execute sp_add_product 'Michelob Light',0,1,4.50, 3
```

100 %

Results | Messages

	(No column name)	(No column name)	(No column name)
1	Product	Michelob Light	was added

These screens show the t_product table before and after the above-mentioned successful insert.

100 %

Results | Messages

	pro_id	pro_name	pro_instock	ty_id	pro_base	ven_id
1	1	TB-Tiger Tail Ale	0	1	4.50	1
2	2	TB-Uni Lager	312	1	4.50	1
3	3	BSP-Ginger Beer	185	2	2.50	2
4	4	BSP-Root Beer	320	2	2.50	2
5	5	BSP-Orange Soda	255	2	2.50	2
6	6	GB-Primal	18	1	5.00	3
7	7	GB-Hoppy	25	1	6.00	3
8	8	GB-Porter	20	1	5.00	3
9	9	GB-Lager	15	1	5.50	3
10	10	GB-Ale	20	1	5.00	3

Results | Messages

	pro_id	pro_name	pro_instock	ty_id	pro_base	ven_id
1	1	TB-Tiger Tail Ale	0	1	4.50	1
2	2	TB-Uni Lager	312	1	4.50	1
3	3	BSP-Ginger Beer	185	2	2.50	2
4	4	BSP-Root Beer	320	2	2.50	2
5	5	BSP-Orange Soda	255	2	2.50	2
6	6	GB-Primal	18	1	5.00	3
7	7	GB-Hoppy	25	1	6.00	3
8	8	GB-Porter	20	1	5.00	3
9	9	GB-Lager	15	1	5.50	3
10	10	GB-Ale	20	1	5.00	3
11	11	Michelob Light	0	1	4.50	3

This screen shot demonstrates a failed attempt to insert an existing product. The product was matched, even though the case and spacing had changed.

```
--Attempts to add an existing product to the system / should fai
use gibino execute sp_add_product 'TB-U ni    Lager',0,1,4.50, 2
```

100 %

Results | Messages

	(No column name)	(No column name)
1	TB-U ni Lager	already exists in this system as a product.

This screen shot demonstrates a failed attempt to insert a product with an invalid type.

```
--Attempts to add product with invalid type / should fail
use gibino execute sp_add_product 'USB Draft',0,45,4.50, 1
```

100 %

Results | Messages

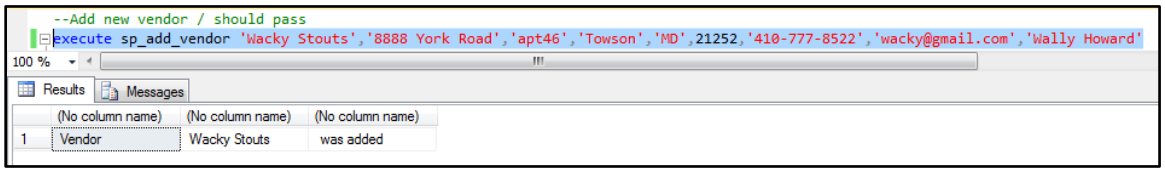
	(No column name)	(No column name)
1	USB Draft	was not given a valid type.

4 - The system must be able to add new vendors.

The following code creates a stored procedure to add new vendors. It validates that the vendor's name does not already exist in the system.

```
use gibino
if objectproperty(object_id('dbo.sp_add_vendor'), N'IsProcedure') = 1
drop procedure dbo.sp_add_vendor
go

create procedure sp_add_vendor
(@ven_name varchar(30),@ven_street1 varchar(30), @ven_street2 varchar(30), @ven_city varchar(30)
, @ven_state char(2),
@ven_zip numeric(5,0), @ven_phone varchar(12), @ven_email varchar(50),@ven_contact varchar(50))
as
begin

--Checks for duplicate vendor names. Both differing cases and spacing are accommodated.
if exists (select * from t_vendor where upper(replace(ven_name, '
',''))=upper(replace(@ven_name, ' ','')))
       begin
              select @ven_name, 'already exists in this system as a vendor'
              return
end

--Inserts vendor information into t_vendor table
begin transaction
        insert into t_vendor
       (ven_name,ven_street1, ven_street2, ven_city, ven_state,
       ven_zip, ven_phone, ven_email, ven_contact)
        values
        (@ven_name,@ven_street1, @ven_street2, @ven_city, @ven_state,
       @ven_zip, @ven_phone, @ven_email, @ven_contact)

--Message for rollback
if @@error<>0
       begin
              rollback transaction
              select 'Vendor ', @ven_name, ' not added'
              return
       end

--Message for success
commit transaction
select   'Vendor ', @ven_name, ' was added'
end
```

This screen shot demonstrates the successful insertion of a new vendor.



These screenshots show the t_vendor table before and after the above-mentioned successful insert.

100 %

Results | Messages

	ven_id	ven_name	ven_street1	ven_street2	ven_city	ven_state	ven_zip	ven_phone	ven_email	ven_contact
1	1	TOWSON BREW GUYS	5852 York Road		Towson	MD	21252	443-458-8522	towson@gmail.com	Lisa Howard
2	2	BALTIMORE SODA POP	987 Penn Blvd.		Baltimore	MD	21239	443-100-8970	bchill@gmail.com	Jason Doe
3	3	GIBINO BREWING	123 Brewers Blvd.		Baltimore	MD	21239	443-200-8970	gb_brewery@gmail.com	Gib Ino
4	4	MONASTIC LAGER COMPANY	123 Monastery Road.		Baltimore	MD	21239	410-288-5566	those_monks@gmail.com	Greg Unkl

100 %

Results | Messages

	ven_id	ven_name	ven_street1	ven_street2	ven_city	ven_state	ven_zip	ven_phone	ven_email	ven_contact
1	1	TOWSON BREW GUYS	5852 York Road		Towson	MD	21252	443-458-8522	towson@gmail.com	Lisa Howard
2	2	BALTIMORE SODA POP	987 Penn Blvd.		Baltimore	MD	21239	443-100-8970	bchill@gmail.com	Jason Doe
3	3	GIBINO BREWING	123 Brewers Blvd.		Baltimore	MD	21239	443-200-8970	gb_brewery@gmail.com	Gib Ino
4	4	MONASTIC LAGER COMPANY	123 Monastery Road.		Baltimore	MD	21239	410-288-5566	those_monks@gmail.com	Greg Unkl
5	5	Wacky Stouts	8888 York Road	apt46	Towson	MD	21252	410-777-8522	wacky@gmail.com	Wally Howard

This screen shot demonstrates a failed attempt to insert a new vendor with an existing name.

```
--Add exisitng vendor / should fail
execute sp_add_vendor 'TOWSON BRE W GUYS','5852 gYork Road','apt46','Towson','MD',21252,'443-458-8522','towson@gmail.com','Lisa Howard'
```

100 %

Results | Messages

	(No column name)	(No column name)
1	TOWSON BRE W GUYS	already exsists in this system as a vendor

5 - The system must be able to update the vendor information.

The following code demonstrates the creation of a stored procedure, which allows the user to update the information for a given vendor. It validates that the vendor name is not updated to the name of an existing vendor.

```
use gibino
if objectproperty(object_id('dbo.sp_update_vendor'), N'IsProcedure') = 1
drop procedure dbo.sp_update_vendor
go

create procedure sp_update_vendor
(@ven_id int, @ven_name varchar(30),@ven_street1 varchar(30), @ven_street2 varchar(30),
@ven_city varchar(30) , @ven_state char(2),
@ven_zip numeric(5,0), @ven_phone varchar(12), @ven_email varchar(50),@ven_contact varchar(50))
as
begin

--Checks for that another vendor, with the same name, does not exist. Accommodates for differing
capitalization and spacing.
if exists (select * from t_vendor where upper(replace(ven_name, ' ', ''
))=upper(replace(@ven_name, ' ', '' )) and ven_id <> @ven_id)
	begin
		select @ven_name, 'already exists in this system as a vendor. Please check that
you have the correct vendor information.'
		return
end
else

--Checks that the vendor exists in the system
if not exists (select * from t_vendor where ven_id = @ven_id)
	begin
		select @ven_id, ' is not found, as a vendor, in the system.'
		return
end

--Updates vendor information in t_vendor table
begin transaction
	 update t_vendor set
	 ven_name=@ven_name, ven_street1=@ven_street1, ven_street2=@ven_street2,
ven_city=@ven_city, ven_state=@ven_state,
	 ven_zip=@ven_zip, ven_phone=@ven_phone, ven_email=@ven_email, ven_contact=@ven_contact
	 where ven_id=@ven_id

--Message for rollback
if @@error<>0
	begin
		rollback transaction
		select 'Vendor ', @ven_name, ' was not updated'
		return
	end

--Message for success
commit transaction
select 'Vendor ', @ven_name, ' was updated'
end
```

This shows the successful name change for a vendor.

```
-- Updates existing vendor / should work
use gibino execute sp_update_vendor 3, 'GIBINO BREWERY','123 NEW ADDRESS','Unit YYY','Towson','MD',21252,'443-444-8522','GIBINO@gmail.c
```

100 %

Results | Messages

	(No column name)	(No column name)	(No column name)
1	Vendor	GIBINO BREWERY	was updated

These screens show rows from the t_vendor table before and after the above-mentioned successful update.

100 %

Results | Messages

	ven_id	ven_name	ven_street1	ven_street2	ven_city	ven_state	ven_zip	ven_phone	ven_email	ven_contact
1	1	TOWSON BREW GUYS	5852 York Road		Towson	MD	21252	443-458-8522	towson@gmail.com	Lisa Howard
2	2	BALTIMORE SODA POP	987 Penn Blvd.		Baltimore	MD	21239	443-100-8970	bchill@gmail.com	Jason Doe
3	3	GIBINO BREWING	123 Brewers Blvd.		Baltimore	MD	21239	443-200-8970	gb_brewery@gmail.com	Gib Ino
4	4	MONASTIC LAGER COMPANY	123 Monastery Road.		Baltimore	MD	21239	410-288-5566	those_monks@gmail.com	Greg Unkl

100 %

Results | Messages

	ven_id	ven_name	ven_street1	ven_street2	ven_city	ven_state	ven_zip	ven_phone	ven_email	ven_contact
1	1	TOWSON BREW GUYS	5852 York Road		Towson	MD	21252	443-458-8522	towson@gmail.com	Lisa Howard
2	2	BALTIMORE SODA POP	987 Penn Blvd.		Baltimore	MD	21239	443-100-8970	bchill@gmail.com	Jason Doe
3	3	GIBINO BREWERY	123 NEW ADDRESS	Unit YYY	Towson	MD	21252	443-444-8522	GIBINO@gmail.com	Guy Bino
4	4	MONASTIC LAGER COMPANY	123 Monastery Road.		Baltimore	MD	21239	410-288-5566	those_monks@gmail.com	Greg Unkl

This shows the resulting error that occurs when the user attempts to update the vendor's name to the name of another vendor.

```
-- Tries to change name to that of another vendor/ should fail
use gibino execute sp_update_vendor 3, 'GIBINO BREWERY','583352 gYork Road','apt46','Towson','MD',21252,'443-458-8522','towson@gmail.com','Lis
```

100 %

Results | Messages

	(No column name)	(No column name)
1	GIBINO BREWERY	already exsists in this system as a vendor. Please check that you have the correct vendor information.

This shows the resulting error that occurs when the user attempts to update a vendor that does not exist.

```
-- Tries to update non-existent vendor / should fail
use gibino execute sp_update_vendor 55448, 'OLDHAM BREWERS','45880 York Road','Unit YYY','Towson','MD',21252,'443-444-8522','GIBINO@gma
```

100 %

Results | Messages

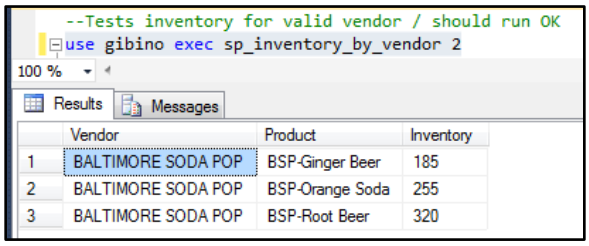
	(No column name)	(No column name)
1	55448	is not found, as a vendor, in the system.

6 - The system must be able to display the beverage inventory belonging to a given vendor.

The following code creates a stored procedure which can be used to query the inventory for a given vendor. It validates that the vendor exists in the system.

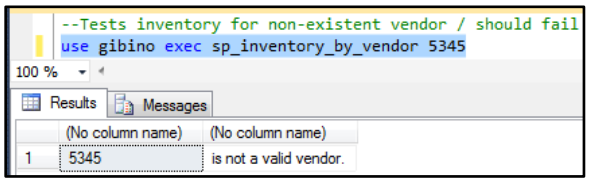
```
use gibino
if objectproperty(object_id('dbo.sp_inventory_by_vendor'), N'IsProcedure') = 1
drop procedure dbo.sp_inventory_by_vendor
go

create procedure sp_inventory_by_vendor
(@ven_id int)
as begin

--Checks that vendor exists
if exists (select * from t_vendor where ven_id=@ven_id)
	begin
		select v.ven_name as Vendor, p.pro_name as Product, p.pro_instock as Inventory
		from t_vendor v left join t_product p on v.ven_id = p.ven_id
		where v.ven_id=@ven_id
		order by Vendor, Product
		return
	end
else

	begin
		select @ven_id, 'is not a valid vendor.'
		return
	end

--Message for rollback
if @@error<>0
	begin
		rollback transaction
		select 'Transaction rolled back. There was an error.'
		return
	end

--Message for success
commit transaction
select	'Output success'
end
```

This screen shot demonstrates the stored procedure pulling the inventory for a given vendor.



	Vendor	Product	Inventory
1	BALTIMORE SODA POP	BSP-Ginger Beer	185
2	BALTIMORE SODA POP	BSP-Orange Soda	255
3	BALTIMORE SODA POP	BSP-Root Beer	320

This screenshot demonstrates the error that occurs when the stored procedure is run for a non-existent vendor.



7 - The system must produce a formatted report of all products and available quantities for every vendor.

This stored procedure uses nested cursors to produce a formatted report of products and inventories, by vendor.

```
use gibino
if objectproperty(object_id('dbo.sp_inventory_cursoring'), N'IsProcedure') = 1
drop procedure dbo.sp_inventory_cursoring
go

--This stored procedure uses two nested cursors to report on each vendor and the available
products and inventory
--The following URL served as a resource guide and template for the code.
http://technet.microsoft.com/en-us/library/aa258831(v=sql.80).aspx
--The first cursor (vendor_cursor) selects the vendors, one-by-one
--The second cursors (product_cursors) selects the products (beverages) and available inventory
for each product
       --for the current vendor in the vendor_cursor

create proc sp_inventory_cursoring
as begin
begin transaction

--Prevents the output from displaying row numbers.
set nocount on

--The various variables are declared. The length of the broadcase variable is set to 140, in
case this cursor
       --will be used for a twitter feed.
declare
@ven_id int,  @ven_name varchar(30), @broadcast varchar(140), @pro_name varchar(30),
@pro_instock int

--This is the title that will be presented at the top of the report
print 'INVENTORY REPORT, LISTS ALL VENDORS AND AVAILABLE QUANTITIES OF EACH BEVERAGE'

declare vendor_cursor cursor for
select ven_id, ven_name
from t_vendor
order by ven_name

open vendor_cursor

fetch next from vendor_cursor
into @ven_id, @ven_name

while @@fetch_status=0
begin
       print ' '
       select @broadcast = '---Beverages from Vendor ' + @ven_name

       print @broadcast

                     -- The product_cursor, nested inside of the vendor_cursor begins here
                     declare product_cursor cursor for
                     select p.pro_name, p.pro_instock
                     from t_product p
                     where p.ven_id=@ven_id
```

```
                    order by p.pro_name

                    open product_cursor
                    fetch next from product_cursor into @pro_name, @pro_instock

                    if @@fetch_status <> 0
                           print '       This vendor does not have any products in the current
inventory.'

                    while @@fetch_status=0
                    begin

                    --Had to cast the integer variable, pro_instock, into a varchar to
concatenate it.
                    select @broadcast = '                    ' + @pro_name + ', ' +
cast(@pro_instock as varchar) + ' units in stock.'
                    print @broadcast
                    fetch next from product_cursor into @pro_name, @pro_instock

                    --Closes the product_cursor for the current vendor
                    end

                    close product_cursor
                    deallocate product_cursor

--Vendor_cursor moves to the next vendor the next vendor
fetch next from vendor_cursor
into @ven_id, @ven_name
end

--Vendor cursor closes
close vendor_cursor
deallocate vendor_cursor

--Message for rollback
if @@error<>0
       begin
              rollback transaction
              select 'Transaction rolled back. There was an error.'
              return
       end

--Success will display report output
commit transaction
end
```

This screenshot shows the successful execution of the report.

```
sp_inventory_curso...nspiron\Media (55))*
exec sp_inventory_cursoring
100 %
Messages
INVENTORY REPORT, LISTS ALL VENDORS AND AVAILABLE QUANTITIES OF EACH BEVERAGE

---Beverages from Vendor BALTIMORE SODA POP
            BSP-Ginger Beer, 185 units in stock.
            BSP-Orange Soda, 255 units in stock.
            BSP-Root Beer, 320 units in stock.

---Beverages from Vendor GIBINO BREWING
            GB-Ale, 20 units in stock.
            GB-Hoppy, 25 units in stock.
            GB-Lager, 15 units in stock.
            GB-Porter, 20 units in stock.
            GB-Primal, 18 units in stock.

---Beverages from Vendor MONASTIC LAGER COMPANY
    This vendor does not have any products in the current inventory.

---Beverages from Vendor TOWSON BREW GUYS
            TB-Tiger Tail Ale, 0 units in stock.
            TB-Uni Lager, 312 units in stock.
```

8 – The system must display original prices and current prices, % difference in price from current to original, daily high price for products, daily low price for products.

The following code creates a stored procedure view the Price Report data with a date parameter.

```
/* CHECK IF PROCEDURE EXISTS, DROP AND RECREATE IT */
IF OBJECTPROPERTY(object_id('dbo.sp_price_report'), N'IsProcedure') = 1
DROP PROCEDURE [dbo].[sp_price_report]
GO
CREATE PROCEDURE dbo.sp_price_report
@date DATETIME
AS
BEGIN

BEGIN TRANSACTION
/* HELPER TABLE */
IF OBJECT_ID('dbo.t_price_diff', 'U') IS NOT NULL
DROP TABLE dbo.t_price_diff
CREATE TABLE dbo.t_price_diff
(
diff_id INT          IDENTITY(1,1)        PRIMARY KEY,
pro_id         INT                FOREIGN KEY REFERENCES dbo.t_product(pro_id),
diff_perc                  DECIMAL(5,1)  NOT null

);

/* CALCULATE THE PRICE DIFFERENCES */
INSERT INTO T_PRICE_DIFF (PRO_ID, DIFF_PERC)
SELECT pri.pro_id, ((pri.pro_price - pro.pro_base)/pro.pro_base) * 100
FROM t_price pri INNER JOIN t_product pro
ON pri.pro_id = pro.pro_id;

/* ROLLBACK ON ERROR */
IF @@error <> 0

       BEGIN
              ROLLBACK TRANSACTION
              SELECT ' There was a problem creating the price report'
              RETURN
       END

COMMIT TRANSACTION;

/* QUERY FOR THE REPORT */
SELECT pro.pro_name AS Product, pro.pro_base AS OriginalPrice, pri.pro_price AS CurrentPrice,
d.diff_perc AS PercentageDifference
 , (CASE WHEN max(s.pro_price) < pri.pro_price THEN pri.pro_price ELSE max(s.pro_price) END) AS
DailyHigh
 , min(s.pro_price) AS DailyLow
FROM t_price pri
INNER JOIN t_product pro ON pri.pro_id = pro.pro_id
INNER JOIN t_price_diff d ON d.pro_id = pri.pro_id
INNER JOIN t_pos_sales s ON s.pro_id = d.pro_id
WHERE DAY(s.pos_datetime) = DAY(@date)
GROUP BY pri.pro_id, pro.pro_name,pro.pro_base, pri.pro_price, d.diff_perc

UNION
```

```sql
/* ADD PRODUCTS WITH NO SALES */
SELECT pro.pro_name AS Product, pro.pro_base AS OriginalPrice, pri.pro_price AS CurrentPrice, 0
AS PercentageDifference, pro.pro_base AS DailyHigh, pro.pro_base AS DailyLow
FROM t_price pri
INNER JOIN t_product pro ON pri.pro_id = pro.pro_id
INNER JOIN t_price_diff d ON d.pro_id = pri.pro_id
WHERE pri.pro_id NOT IN (SELECT pro_id FROM t_pos_sales WHERE DAY(pos_datetime) = DAY(@date));
END
```

This screen capture demonstrates successful generation of a Price Report for the day input.

```sql
--exec sp price report @date = '2014-11-28 13:00'
```

100 %

Results | Messages

	Product	OriginalPrice	CurrentPrice	PercentageDifference	DailyHigh	DailyLow
1	BSP-Ginger Beer	2.50	2.50	0.0	2.50	2.50
2	BSP-Orange Soda	2.50	2.50	0.0	2.50	2.50
3	BSP-Root Beer	2.50	2.50	0.0	2.50	2.50
4	GB-Ale	5.00	5.00	0.0	5.00	5.00
5	GB-Hoppy	6.00	6.00	0.0	6.00	6.00
6	GB-Lager	5.50	5.50	0.0	5.50	5.50
7	GB-Porter	5.00	5.00	0.0	5.00	5.00
8	GB-Primal	5.00	5.75	15.0	5.75	5.00
9	TB-Tiger Tail Ale	4.50	4.50	0.0	4.50	4.50
10	TB-Uni Lager	4.50	4.50	0.0	4.50	4.50

9 – The system must display prices for most recently sold products.

The following code will create a stored procedure to return the products sold within the last hour, the price they sold for and the time of sale.

```
/* CHECK IF PROCEDURE EXISTS, DROP AND RECREATE IT */
IF OBJECTPROPERTY(object_id('dbo.sp_recent_sales'), N'IsProcedure') = 1
DROP PROCEDURE [dbo].[sp_recent_sales]
GO
CREATE PROCEDURE dbo.sp_recent_sales
AS
BEGIN

SELECT pro_name AS Product, pro_price AS PriceSold, pos_datetime AS TimeOfSale
FROM t_pos_sales s
INNER JOIN t_product p ON p.pro_id = s.pro_id
WHERE pos_datetime > DateAdd(Hour, -1, GETDATE()) and pos_datetime < GETDATE();

END
```

This screen capture demonstrates the successful generation of a recent sales report.

```
1   --exec sp_recent_sales
```

100 %

Results | Messages

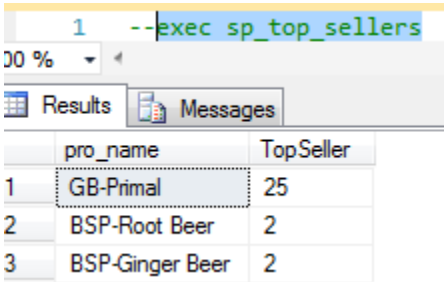
	Product	PriceSold	TimeOfSale
1	BSP-Ginger Beer	2.50	2014-11-28 16:20:23.603
2	BSP-Ginger Beer	2.50	2014-11-28 16:20:23.630
3	BSP-Orange Soda	2.50	2014-11-28 16:20:23.700
4	BSP-Root Beer	2.50	2014-11-28 16:20:23.700
5	BSP-Orange Soda	2.50	2014-11-28 16:20:23.713
6	BSP-Ginger Beer	2.50	2014-11-28 16:45:19.350
7	GB-Primal	5.00	2014-11-28 16:47:09.283

10 – The system must display top selling products of the day.

The following code will create a stored procedure to generate a report of the top 3 products for the day.

```
/* CHECK IF PROCEDURE EXISTS, DROP AND RECREATE IT */
IF OBJECTPROPERTY(object_id('dbo.sp_top_sellers'), N'IsProcedure') = 1
DROP PROCEDURE [dbo].[sp_top_sellers]
GO
CREATE PROCEDURE dbo.sp_top_sellers
AS
BEGIN
/* SELECT TOP 3 SELLING PRODUCTS FOR THE CURRENT DAY */
SELECT top 3 pro_name, max(pos_qty) AS TopSeller
FROM t_pos_sales s
INNER JOIN t_product p ON p.pro_id = s.pro_id
WHERE Day(pos_datetime) = Day(GetDate())
GROUP BY p.pro_id, pro_name
ORDER BY TopSeller DESC;

END
```

This screen capture demonstrates successful generation of a top sellers report.



	pro_name	TopSeller
1	GB-Primal	25
2	BSP-Root Beer	2
3	BSP-Ginger Beer	2

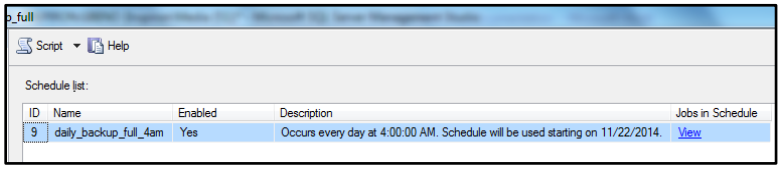
11 -The system must create a compressed nightly backup, after normal business hours, and compressed differential backups throughout the day.

The following code first demonstrates how to put the database into simple recovery mode. It then creates three stored procedures t setup (1) ad hoc, (2) full, and (3)differential backups.

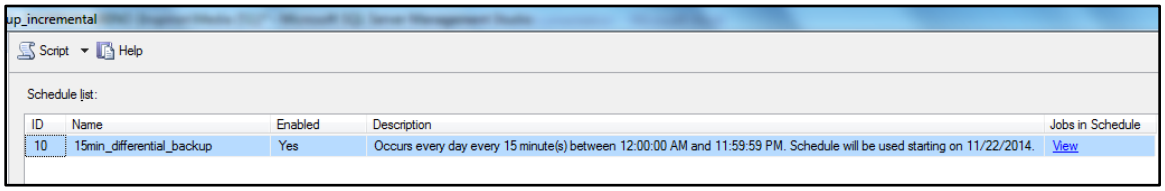
```
--Sets database to simple recovery mode
alter database gibino set recovery simple;


-SP1-----------------------------------------------------------------------------------
--This does a "COPY ONLY" backup. It should be used when making a one-off (ad-hoc) backup
--It will not interfere with the relationship between the full and differential backups
use gibino
if objectproperty(object_id('dbo.sp_backup_adhoc'), N'IsProcedure') = 1
drop procedure dbo.sp_backup_adhoc
go

create proc sp_backup_adhoc
as
begin
	backup database gibino to disk = 'C:\backups\GIBINO_ADHOC.bak' with copy_only,
compression;

--Message for problem
if @@error<>0
	begin
		select 'Problem creating ad hoc backup.'
		return
	end
end


--SP2-----------------------------------------------------------------------------------
--Does a full database backup / scheduled to run daily at 4AM / differential backups use this as
their base
use gibino
if objectproperty(object_id('dbo.sp_backup_full'), N'IsProcedure') = 1
drop procedure dbo.sp_backup_full
go

create proc sp_backup_full
as
begin
	backup database gibino to disk = 'C:\backups\GIBINO_FULL.bak' with compression;

--Message for problem
if @@error<>0
	begin
		select 'Problem creating full backup.'
		return
	end
end


--SP3-----------------------------------------------------------------------------------
```

```
--Does a differential backup (only backups changes since most recent full backup) / schedule to
run every fifteen minutes
use gibino
if objectproperty(object_id('dbo.sp_backup_diff'), N'IsProcedure') = 1
drop procedure dbo.sp_backup_diff
go

create proc sp_backup_diff
as
begin
	backup database gibino to disk = 'C:\backups\GIBINO_DIFFERENTIAL.bak' with differential,
compression;

--Message for problem
if @@error<>0
	begin
		select 'Problem creating differential backup.'
		return
	end
end
```

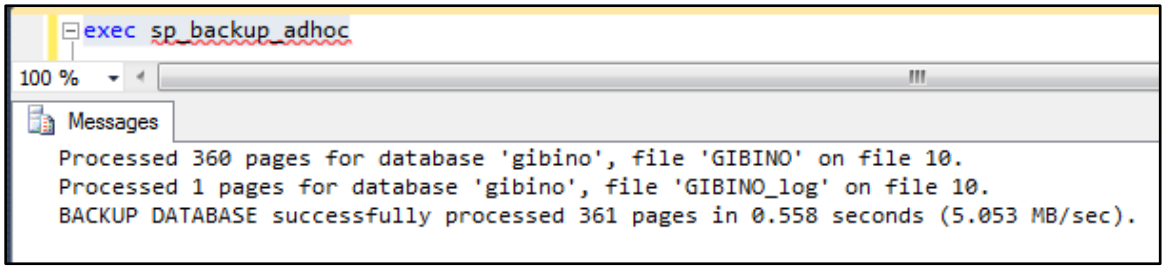
This screen capture shows that the full backup is a scheduled job, which runs at 4:00 AM every day.



This screen capture shows that the differential backup is a scheduled job, which runs every 15 minutes.



This screen capture demonstrates the output for a successful ad hoc backup.



These screen captures demonstrate the output and error messages for a failed (due to the backup location not existing) ad hoc backup.

```
exec sp_backup_adhoc
```

100 %

Results | Messages

	(No column name)
1	Problem creating ad hoc backup.

```
exec sp_backup_adhoc
```

100 %

Results | Messages

```
Msg 3201, Level 16, State 1, Procedure sp_backup_adhoc, Line 5
Cannot open backup device 'C:\badspot_backups\GIBINO_ADHOC.bak'. Operating system error 3(The system cannot find the path specified.).
Msg 3013, Level 16, State 1, Procedure sp_backup_adhoc, Line 5
BACKUP DATABASE is terminating abnormally.

(1 row(s) affected)
```

12 – The system must be able to update product prices based on sales.

The following code will create a stored procedure to update product prices based on sales.

```
/* CHECK IF PROCEDURE EXISTS, DROP AND RECREATE IT */
IF OBJECTPROPERTY(object_id('dbo.sp_update_price_from_sales'), N'IsProcedure') = 1
DROP PROCEDURE [dbo].[sp_update_price_from_sales]
GO
CREATE PROCEDURE dbo.sp_update_price_from_sales

/* DATE PARAMETERS */
@start_time DATETIME,
@end_time DATETIME
AS
BEGIN
BEGIN TRANSACTION;
/* HELPER TABLE */
IF OBJECT_ID('dbo.t_sales_info', 'U') IS NOT NULL
DROP TABLE dbo.t_sales_info
CREATE TABLE dbo.t_sales_info
(
sal_id INT          IDENTITY(1,1)       PRIMARY KEY,
pro_id        INT                FOREIGN KEY REFERENCES dbo.t_product(pro_id),
qty_sold                  INT    NOT null

);

/* GRAB ALL THE SALES FOR THE TIME PERIOD */
INSERT INTO T_SALES_INFO (PRO_ID, QTY_SOLD)
SELECT b.pro_id,sum(b.qty_sold)
FROM v_beer_sales b
WHERE time_of_sale between @start_time AND @end_time
GROUP BY b.pro_id

/* HELPER TABLE */
IF OBJECT_ID('dbo.t_sales_perc', 'U') IS NOT NULL
DROP TABLE dbo.t_sales_perc
CREATE TABLE dbo.t_sales_perc
(
sal_id INT          IDENTITY(1,1)       PRIMARY KEY,
pro_id        INT                FOREIGN KEY REFERENCES dbo.t_product(pro_id),
pct_of_sales              INT    NOT null

);

/* PERCENTAGE OF SALES MATH */
INSERT INTO T_SALES_PERC (PRO_ID, PCT_OF_SALES)
SELECT s.pro_id, ROUND(s.qty_sold * 100.0/(SELECT sum(s.qty_sold) FROM t_sales_info s),2)
FROM t_sales_info s

/* HELPER TABLE */
IF OBJECT_ID('dbo.t_price_adjust', 'U') IS NOT NULL
DROP TABLE dbo.t_price_adjust
CREATE TABLE dbo.t_price_adjust
(
pa_id INT           IDENTITY(1,1)       PRIMARY KEY,
pro_id        INT                FOREIGN KEY REFERENCES dbo.t_product(pro_id),
price_adjust  DECIMAL(5,2)  NOT null

);
```

```sql
/* PRICE ADJUSTMENT CALCULATIONS */
INSERT INTO T_PRICE_ADJUST(PRO_ID, PRICE_ADJUST)
SELECT p.pro_id, CASE
                                WHEN pct_of_sales = 0 THEN 0
                                WHEN pct_of_sales > 0 AND pct_of_sales < 20 THEN 0
                                WHEN pct_of_sales >= 20 AND pct_of_sales < 40 THEN .25
                                WHEN pct_of_sales >= 40 AND pct_of_sales < 60 THEN .5
                                WHEN pct_of_sales >= 60 AND pct_of_sales < 80 THEN .75
                                WHEN pct_of_sales >= 80 AND pct_of_sales <= 100 THEN 1
                            END
FROM t_sales_perc p;

/* UPDATE T_PRICE TABLE */
UPDATE p
SET p.pro_price = p.pro_price + pa.price_adjust
FROM T_PRICE p INNER JOIN T_PRICE_ADJUST pa
ON p.pro_id = pa.pro_id;

/* ROLLBACK IF THERE WAS AN ERROR */
IF @@error <> 0

       BEGIN
              ROLLBACK TRANSACTION
              SELECT ' Price update was unsuccessful'
              RETURN
       END

COMMIT TRANSACTION;
SELECT * FROM T_PRICE;

END
```

This screen capture demonstrates successful update of product prices. Product 6 original price is $5.00 in this scenario and after the stored procedure is executed, the price is updated to $5.75 based on sales for 11/28 between 4 and 5 pm.

```
2  --exec sp_update_price_from_sales @start_time = '2014-11-28 16:00',@end_time = '2014-11-28 17:00';
```

100 %

Results | Messages

	pri_id	pro_id	pro_price
1	1	1	4.50
2	2	2	4.50
3	3	3	2.50
4	4	4	2.50
5	5	5	2.50
6	6	6	5.00
7	7	7	6.00
8	8	8	5.00
9	9	9	5.50
10	10	10	5.00

	pri_id	pro_id	pro_price
1	1	1	4.50
2	2	2	4.50
3	3	3	2.50
4	4	4	2.50
5	5	5	2.50
6	6	6	5.75
7	7	7	6.00
8	8	8	5.00
9	9	9	5.50
10	10	10	5.00

13 – The system must not be able to sell a quantity which exceeds the inventory available, and the system must not be able to sell inventory to a customer below the age of 21.

The following code will create a stored procedure to create point of sale transactions. Once a sale is made and entry to the table is created and inventory levels are updated. There is validation for alcoholic products.

```
/* CHECK IF PROCEDURE EXISTS, DROP AND RECREATE IT */
IF OBJECTPROPERTY(object_id('dbo.sp_add_pos_sale'), N'IsProcedure') = 1
DROP PROCEDURE [dbo].[sp_add_pos_sale]
GO
CREATE PROCEDURE dbo.sp_add_pos_sale
@pro_id INT,
@pur_qty INT,
@cus_id INT,
@pos_paid INT
AS
BEGIN
/*  VALIDATE PRODUCT   */

IF NOT EXISTS(SELECT * FROM t_product WHERE pro_id = @pro_id)
       BEGIN
              SELECT 'This product does NOT exist!'
              RETURN
       END

/*  VALIDATE CUSTOMER   */

IF NOT EXISTS(SELECT * FROM t_customer WHERE cus_id = @cus_id)
       BEGIN
              SELECT 'This customer does NOT exist!'
              RETURN
       END

/*  VALIDATE CUSTOMER   */

IF @pur_qty <= 0
       BEGIN
              SELECT 'This is NOT a valid purchase amount!'
              RETURN
       END


/*  VALIDATE SUFFICIENT INVENTORY   */

IF NOT EXISTS(SELECT * FROM dbo.t_product WHERE pro_id=@pro_id AND pro_instock>=@pur_qty)
       BEGIN
              SELECT 'We do NOT have enough of this product IN stock. The amount IN stock is',
(SELECT pro_instock FROM t_product WHERE pro_id=@pro_id)
              RETURN
       END

/*  VALIDATE AGE FOR RESTRICTED ITEM  */
IF EXISTS(SELECT * FROM t_product WHERE pro_id=@pro_id AND ty_id IN (SELECT ty_id FROM t_type
WHERE ty_restricted=1))
       BEGIN
              /* CHECK FOR CUSTOMER AGE, CREATE A HELPER TABLE TO DETERMINE AGE */
              BEGIN TRANSACTION
              IF OBJECT_ID('dbo.t_cus_age', 'U') IS NOT NULL
```

```
            DROP TABLE dbo.t_cus_age
            CREATE TABLE dbo.t_cus_age
            (
            cage_id INT          IDENTITY(1,1)        PRIMARY KEY,
            cus_id         INT                 FOREIGN KEY REFERENCES dbo.t_product(pro_id),
            age                  INT    NOT null
            );

            INSERT INTO T_CUS_AGE(CUS_ID, AGE)
            SELECT cus_id, DATEDIFF(hour, cus_dob, GETDATE())/8766 AS AgeInYears FROM
t_customer;

            /* CUSTOMER IS NOT OF AGE */
            IF NOT EXISTS(SELECT * FROM t_cus_age a
                          INNER JOIN t_customer c ON a.cus_id = c.cus_id
                          WHERE age < 21 AND a.cus_id = @cus_id)
            BEGIN
                  ROLLBACK TRANSACTION
                  SELECT 'Alcoholic beverages can only be purchased BY customers who are 21
years of age or older'
                  RETURN
            END
            COMMIT TRANSACTION
      END

BEGIN TRANSACTION

/* CREATE A SALE */
insert into dbo.t_pos_sales(pos_qty, cus_id, pro_id, pos_paid, pro_price)
values(@pur_qty, @cus_id, @pro_id, @pos_paid, (SELECT pro_price FROM dbo.t_price WHERE pro_id =
@pro_id));

update t_product
set pro_instock = (pro_instock - @pur_qty)
WHERE pro_id = @pro_id;

/* ROLLBACK IF AN ERROR HAS OCCURRED */
IF @@error <> 0

      BEGIN
            ROLLBACK TRANSACTION
            SELECT ' Sale was NOT completed'
            RETURN
      END

COMMIT TRANSACTION;

SELECT * FROM t_pos_sales;
SELECT   'Sold ', @pur_qty, 'units of ', (SELECT pro_name FROM t_product WHERE pro_id =
@pro_id), ' at ', (SELECT pro_price FROM t_price WHERE pro_id = @pro_id), ' each.'

END
```

This screen capture demonstrates a valid purchase.

```
EXEC sp_add_pos_sale @pro_id = 6, @pur_qty = 25, @cus_id = 1, @pos_paid = 1;
EXEC sp_add_pos_sale @pro_id = 7, @pur_qty = 5, @cus_id = 2, @pos_paid = 1;
```

Results | Messages

	pos_id	pos_datetime	pos_qty	cus_id	pro_id	pro_price	pos_paid
1	1	2014-11-28 16:20:23.603	2	1	3	2.50	1
2	2	2014-11-28 16:20:23.630	2	1	3	2.50	1
3	3	2014-11-28 16:20:23.700	1	5	5	2.50	1
4	4	2014-11-28 16:20:23.700	2	1	4	2.50	1
5	5	2014-11-28 16:20:23.713	1	5	5	2.50	1
6	13	2014-11-28 16:45:19.350	2	1	3	2.50	1
7	14	2014-11-28 16:47:09.283	25	1	6	5.00	1
8	15	2014-11-28 17:58:29.190	25	1	6	5.75	1

	(No column name)	(No column name)	(No column name)	(No column name)	(No column name)	(No column name)	(No column name)
1	Sold	25	units of	GB-Primal	at	5.75	each.

This screen capture demonstrates a customer under 21 cannot purchase an alcoholic beverage.

```
EXEC sp_add_pos_sale @pro_id = 8, @pur_qty = 10, @cus_id = 3, @pos_paid = 1;
*/
```

Results | Messages

	(No column name)
1	Alcoholic beverages can only be purchased by customers who are 21 years of age or older

This screen capture demonstrates validation for available inventory.

```
EXEC sp_add_pos_sale @pro_id = 4, @pur_qty = 1000, @cus_id = 3, @pos_paid = 0;
```

Results | Messages

(No column name)	(No column name)
We do not have enough of this product in stock. The amount in stock is	318

This screen capture demonstrates validation for an existing customer.

```
EXEC sp_add_pos_sale @pro_id = 8, @pur_qty = 10, @cus_id = 6, @pos_paid = 1;
```

Results | Messages

(No column name)
This customer does not exist!

This screen capture demonstrates validation for a valid purchase amount (greater than 0)

```
13 EXEC sp_add_pos_sale @pro_id = 8, @pur_qty = 0, @cus_id = 1, @pos_paid = 1;
```

0 %

Results | Messages

(No column name)
This is not a valid purchase amount!

This screen capture demonstrates validation for a valid product.

```
13 EXEC sp_add_pos_sale @pro_id = 12, @pur_qty = 0, @cus_id = 1, @pos_paid = 1;
```

Results | Messages

(No column name)
This product does not exist!

14 - The system must update real-time inventory levels according to sales.

This requirement is satisfied by the stored procedure, sp_add_pos_sale, which is fully displayed in the preceding requirement. The relevant portion of the stored procedure follows.

```
/* CREATE A SALE */
insert into dbo.t_pos_sales(pos_qty, cus_id, pro_id, pos_paid, pro_price)
values(@pur_qty, @cus_id, @pro_id, @pos_paid, (SELECT pro_price FROM dbo.t_price WHERE pro_id =
@pro_id));

update t_product
set pro_instock = (pro_instock - @pur_qty)
WHERE pro_id = @pro_id;

/* ROLLBACK IF AN ERROR HAS OCCURRED */
IF @@error <> 0

	BEGIN
		ROLLBACK TRANSACTION
		SELECT ' Sale was NOT completed'
		RETURN
	END

COMMIT TRANSACTION;

SELECT * FROM t_pos_sales;
SELECT   'Sold ', @pur_qty, 'units of ', (SELECT pro_name FROM t_product WHERE pro_id =
@pro_id), ' at ', (SELECT pro_price FROM t_price WHERE pro_id = @pro_id), ' each.'

END
```

This screen capture demonstrates an update to t_product after a successful purchase. (Product 4 instock was 316, now is 291).

```
5  EXEC sp_add_pos_sale @pro_id = 5, @pur_qty = 10, @cus_id = 3, @pos_paid = 0;
6  EXEC sp_add_pos_sale @pro_id = 4, @pur_qty = 25, @cus_id = 5, @pos_paid = 1;
7  EXEC sp_add_pos_sale @pro_id = 4, @pur_qty = 25, @cus_id = 900, @pos_paid = 1;
```

00 %

Results | Messages

	pro_id	pro_name	pro_instock	ty_id	pro_base	ven_id
1	4	BSP-Root Beer	316	2	2.50	2

	pos_id	pos_datetime	pos_qty	cus_id	pro_id	pro_price	pos_paid
4	4	2014-11-28 16:20:23.700	2	1	4	2.50	1
5	5	2014-11-28 16:20:23.713	1	5	5	2.50	1
6	13	2014-11-28 16:45:19.350	2	1	3	2.50	1
7	14	2014-11-28 16:47:09.283	25	1	6	5.00	1
8	15	2014-11-28 17:58:29.190	25	1	6	5.75	1
9	16	2014-11-30 14:26:17.460	2	1	3	2.50	1
10	17	2014-12-01 21:27:08.087	1	3	4	2.50	0
11	18	2014-12-01 21:30:54.680	1	3	4	2.50	0

	pro_id	pro_name	pro_instock	ty_id	pro_base	ven_id
1	4	BSP-Root Beer	291	2	2.50	2

	pos_id	pos_datetime	pos_qty	cus_id	pro_id	pro_price	pos_paid
5	5	2014-11-28 16:20:23.713	1	5	5	2.50	1
6	13	2014-11-28 16:45:19.350	2	1	3	2.50	1
7	14	2014-11-28 16:47:09.283	25	1	6	5.00	1
8	15	2014-11-28 17:58:29.190	25	1	6	5.75	1
9	16	2014-11-30 14:26:17.460	2	1	3	2.50	1
10	17	2014-12-01 21:27:08.087	1	3	4	2.50	0
11	18	2014-12-01 21:30:54.680	1	3	4	2.50	0
12	19	2014-12-01 21:31:54.820	25	5	4	2.50	1

	(No column name)	(No column name)	(No column name)	(No column name)	(No column name)	(No column name)	(No column name)
1	Sold	25	units of	BSP-Root Beer	at	2.50	each.

15 – The system must be able to display the unpaid items for a customer.

The following code creates a stored procedure to query the system for unpaid items for a customer.

```
/* CHECK IF PROCEDURE EXISTS, DROP AND RECREATE IT */
IF OBJECTPROPERTY(object_id('sp_customer_tab'), N'IsProcedure') = 1
DROP PROCEDURE [dbo].[sp_customer_tab]
GO
CREATE PROCEDURE dbo.sp_customer_tab
@cus_id INT
AS
BEGIN
	/* ROLLBACK IF AN ERROR HAS OCCURRED */
	IF NOT EXISTS (SELECT * FROM T_CUSTOMER WHERE cus_id = @cus_id)
		BEGIN
			SELECT 'Invalid Customer!'
			RETURN
		END
	/* ITEMIZED LIST OF UNPAID ITEMS */
	SELECT pro.pro_name, s.pro_price, s.pos_qty
	FROM t_customer c
	INNER JOIN t_pos_sales s ON c.cus_id=s.cus_id
	INNER JOIN t_product pro ON pro.pro_id=s.pro_id
	WHERE s.pos_paid=0
	AND s.cus_id = @cus_id

	/* CUSTOMER NAME AND SUBTOTAL */
	SELECT c.cus_fname, c.cus_lname,sum(s.pro_price) AS SubTotal
	FROM t_customer c
	INNER JOIN t_pos_sales s ON c.cus_id=s.cus_id
	INNER JOIN t_product pro ON pro.pro_id=s.pro_id
	WHERE s.pos_paid=0
	AND s.cus_id = @cus_id
	GROUP BY c.cus_id,c.cus_fname, c.cus_lname

END
	Select c.cus_fname, c.cus_lname,sum(s.pro_price) As SubTotal
	from t_customer c
	inner join t_pos_sales s on c.cus_id=s.cus_id
	inner join t_product pro on pro.pro_id=s.pro_id
	where s.pos_paid=0
	and s.cus_id = @cus_id
	group by c.cus_id,c.cus_fname, c.cus_lname

End
```

This screen capture demonstrates an execution of stored procedure for an existing customer.

```
--exec sp_customer_tab @cus_id = 3
```

100 %

Results | Messages

	pro_name	pro_price	pos_qty
1	BSP-Root Beer	2.50	1
2	BSP-Orange Soda	2.50	1
3	BSP-Root Beer	2.50	1

	cus_fname	cus_lname	SubTotal
1	Carl	Cox	7.50

This screen capture demonstrates the validation for an existing customer.

sp_customer_tab.s...OL\daquino38 (52)) | sp_add_pos_sale.sq...OL\daquino38 (55))* | insert_data.s

```
--exec sp_customer_tab @cus_id = 25
IF OBJECTPROPERTY(object_id('sp_customer_tab'), N'IsProcedure') = 1
DROP PROCEDURE [dbo].[sp_customer_tab]
GO
```

100 %

Results | Messages

	(No column name)
1	Invalid Customer!

16 - The system must be able to predict when to order inventory based on sales and current inventory levels.

This procedure uses 30 days of average sales to determine how many days of a product remain, given the current level of inventory. It creates temporary tables and then custom messages.

```
use gibino
if objectproperty(object_id('dbo.sp_rem_inv'), N'IsProcedure') = 1
drop procedure dbo.sp_rem_inv
go

IF OBJECT_ID('dbo.t_tmp_sales_30', 'U') IS NOT NULL
drop table dbo.t_tmp_sales_30
go

IF OBJECT_ID('dbo.t_tmp_inv_remain', 'U') IS NOT NULL
drop table dbo.t_tmp_inv_remain
go

create proc sp_rem_inv
as begin
begin transaction

--Creates temporary table showing average sales for past thirty days
select a.pro_id, sum(a.acct_qty)/30 as dailysold into t_tmp_sales_30
from t_acct_sales a
where a.acct_datetime >= dateadd (day,-10000, getdate() )
group by a.pro_id;


--Creates temporary table showing amount on hand, given current inventory
select p.pro_id, p.pro_name, p.pro_instock/nullif(s.dailysold,0) as invdays, s.dailysold,
p.pro_instock into t_tmp_inv_remain
from t_product p left join t_tmp_sales_30 s
on p.pro_id=s.pro_id;

--Creates messages for when:
--     There is no product in stock but there have been sales.
--     There is product in stock and there have not been sales.
--     There is product in stock and there have been sales.
select inv_message =
case
when inv.pro_instock = 0 then inv.pro_name+ ' - No' +inv.pro_name+ ' in stock. The average daily
sales are '+cast(dailysold as varchar)+' units.'
when inv.invdays is NULL then inv.pro_name+ ' - There is not enough sales information to
determine amount of ' + cast(inv.pro_name as varchar) + ' in stock. There are currently ' +
cast(inv.pro_instock as varchar) + ' in stock.'
else inv.pro_name+ '- There are '+cast(inv.invdays as varchar)+' days (estimated)
of'+inv.pro_name+ ' product remaining.'
end
from
t_tmp_inv_remain inv
order by inv_message
```

```
--Message for rollback
if @@error<>0
	begin
		rollback transaction
		select 'Unable to process inventory levels.'
		return
	end

--Message for success
commit transaction
select	'Inventory levels processed. Please refer to output.'
end
```

The following output shows the inventory messages which appear when the query is run.

- When a beverage has a 30-day sales history and product in-stock, a message with the number of remaining days of inventory appears.
- When a beverage does not have 30-day sales history but does have product in-stock, a message indicates that there is not enough sales history to predict days-on-hand. The message also displays the number of units in stock.
- When a beverage is not in stock, the messages indicates that the beverage is not in stock, and the message also includes the average daily sales for that beverage.

Results | Messages

	inv_message
1	BSP-Ginger Beer- There are 37 days (estimated) ofBSP-Ginger Beer product remaining.
2	BSP-Orange Soda - There is not enough sales information to determine amount of BSP-Orange Soda in stock. There are currently 255 in stock.
3	BSP-Root Beer- There are 53 days (estimated) ofBSP-Root Beer product remaining.
4	GB-Ale - There is not enough sales information to determine amount of GB-Ale in stock. There are currently 20 in stock.
5	GB-Hoppy- There are 3 days (estimated) ofGB-Hoppy product remaining.
6	GB-Lager - There is not enough sales information to determine amount of GB-Lager in stock. There are currently 15 in stock.
7	GB-Porter - There is not enough sales information to determine amount of GB-Porter in stock. There are currently 20 in stock.
8	GB-Primal- There are 3 days (estimated) ofGB-Primal product remaining.
9	TB-Tiger Tail Ale - NoTB-Tiger Tail Ale in stock. The average daily sales are 9 units.
10	TB-Uni Lager- There are 26 days (estimated) ofTB-Uni Lager product remaining.

17 - The system must be able to generate profit report.

```
--Profit Report
IF OBJECT_ID ('Profit_Report', 'P') IS NOT NULL
DROP PROCEDURE Profit_Report;
GO

CREATE PROCEDURE Profit_Report
@Start_Date datetime,
@End_Date datetime
AS
Begin
Begin transaction

--To check if any transaction occured on given date parameter

if not exists (select *  from t_POS_Sales where pos_datetime between @Start_Date and @End_Date)
       Begin
              Select 'Error! No Report Found'
       end
--If transaction exists select--
If exists (select pro_id from t_POS_Sales where pos_datetime between @Start_Date and @End_Date)
Begin Try

       Select tpr.pro_id,tpr.pro_name,  (Sum(ts.pro_price*ts.pos_qty)-
sum(tpu.pur_unt_price*ts.pos_qty)) As Net_Profit
       from t_purchase tpu join t_product tpr on tpr.pro_id=tpu.pro_id join t_pos_sales ts on
tpr.pro_id=ts.pro_id
       where ts.pos_datetime between @Start_Date and @End_Date
       Group by tpr.pro_id,tpr.pro_name
       Order by tpr.pro_id;
END TRY
--Error Check
BEGIN CATCH
    SELECT
        ERROR_NUMBER() AS ErrorNumber,
        ERROR_SEVERITY() AS ErrorSeverity,
        ERROR_STATE() as ErrorState,
        ERROR_PROCEDURE() as ErrorProcedure,
        ERROR_LINE() as ErrorLine,
        ERROR_MESSAGE() as ErrorMessage;

IF @@TRANCOUNT > 0
        ROLLBACK TRANSACTION;
End Catch;

IF @@TRANCOUNT > 0
    COMMIT TRANSACTION


              End;
--Exec Profit_Report @Start_Date='10/10/14',@End_Date='10/21/14'
```

This screenshot demonstrates the successful generation of Profit report

```
Exec Profit_Report @Start_Date='07/10/14',@End_Date='11/21/14'
```

100 %

Results | Messages

	pro_id	pro_name	Net_Profit
1	1	Bud Light	56.25
2	2	Red Bull	4.00
3	3	Bud Lime	21.00
4	4	Corona	35.00
5	6	Coke	31.50

This screenshot demonstrates a failed attempt of generation of Profit report- NO REPORT FOUND

```
Exec Profit_Report @Start_Date='10/06/14',@End_Date='10/21/14'
```

100 %

Results | Messages

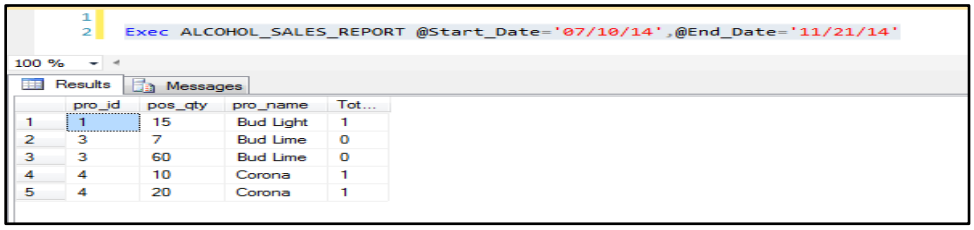
	(No column name)
1	Error! No Report Found

18- The system must be able to display total number of inventory sold by type per day

18A. Generation of Alcoholic Product Sales report:

```
--Alcoholic drinks Sales Report:
IF OBJECT_ID ('ALCOHOL_SALES_REPORT', 'P') IS NOT NULL
    DROP PROCEDURE ALCOHOL_SALES_REPORT;
GO

CREATE PROCEDURE ALCOHOL_SALES_REPORT
@Start_Date datetime,
@End_Date datetime
AS
Begin
Begin transaction
--To check if any transaction occured on given date parameter
if not exists (select *  from t_POS_Sales where pos_datetime between @Start_Date and @End_Date)
       Begin
              Select 'Error! No Report Found'
       end
--If transaction exists select--
If exists (select * from t_POS_Sales where pos_datetime between @Start_Date and @End_Date)
       Begin Try

        SELECT tp.pro_id,tp.pos_qty,SUM(tp.pos_paid)as Total_Sales, p.pro_name
    FROM T_POS_SALES tp
       join t_product p
       on tp.pro_id=p.pro_id join t_type tt
       on p.ty_id=tt.ty_id
       where tp.pos_datetime between @Start_Date and @End_Date AND tt.TY_RESTRICTED=1
       Group by tp.pro_id,pos_qty,p.pro_name
       Order by tp.pro_id;

END TRY
--Error Check
BEGIN CATCH
    SELECT
        ERROR_NUMBER() AS ErrorNumber,
        ERROR_SEVERITY() AS ErrorSeverity,
        ERROR_STATE() as ErrorState,
        ERROR_PROCEDURE() as ErrorProcedure,
        ERROR_LINE() as ErrorLine,
        ERROR_MESSAGE() as ErrorMessage;

IF @@TRANCOUNT > 0
        ROLLBACK TRANSACTION;
End Catch;

IF @@TRANCOUNT > 0
    COMMIT TRANSACTION

END;
--Exec ALCOHOL_SALES_REPORT @Start_Date='09/10/14',@End_Date='09/21/14'
```

This screenshot demonstrates the successful generation of Sales report for Alcohol product.



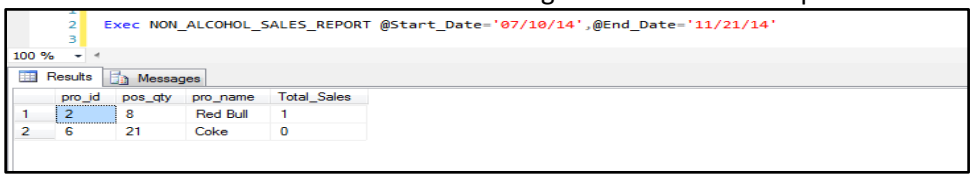
This screenshot demonstrates the failed attempt of generation of Sales report for Alcohol product.

18B. Generation of Non- Alcohol Product Sales report:

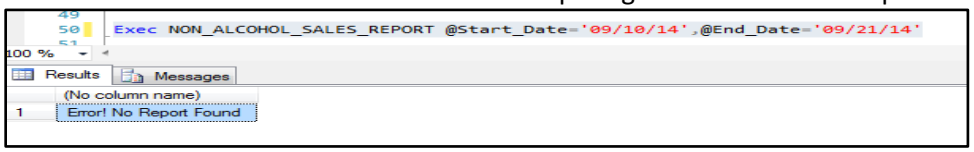
```
--NON_ Alcohol drinks Sales Report:
IF OBJECT_ID ('NON_ALCOHOL_SALES_REPORT', 'P') IS NOT NULL
    DROP PROCEDURE NON_ALCOHOL_SALES_REPORT;
GO

CREATE PROCEDURE NON_ALCOHOL_SALES_REPORT
@Start_Date datetime,
@End_Date datetime
AS
Begin
Begin transaction
--To check if any transaction occured on given date parameter
if not exists (select *  from t_POS_Sales where pos_datetime between @Start_Date and @End_Date)
       Begin
              Select 'Error! No Report Found'
       end
--If transaction exists select--
If exists (select * from t_POS_Sales where pos_datetime between @Start_Date and @End_Date)
       Begin Try
        SELECT tp.pro_id,tp.pos_qty,SUM(tp.pos_paid)as Total_Sales,p.pro_name
    FROM T_POS_SALES tp
       join t_product p
       on tp.pro_id=p.pro_id join t_type tt
       on p.ty_id=tt.ty_id
       where tp.pos_datetime between @Start_Date and @End_Date AND tt.TY_RESTRICTED=2
       Group by tp.pro_id,pos_qty,p.pro_name
       Order by tp.pro_id;

END TRY
```

```
--Error Check
BEGIN CATCH
    SELECT
        ERROR_NUMBER() AS ErrorNumber,
        ERROR_SEVERITY() AS ErrorSeverity,
        ERROR_STATE() as ErrorState,
        ERROR_PROCEDURE() as ErrorProcedure,
        ERROR_LINE() as ErrorLine,
        ERROR_MESSAGE() as ErrorMessage;

IF @@TRANCOUNT > 0
        ROLLBACK TRANSACTION;
End Catch;

IF @@TRANCOUNT > 0
    COMMIT TRANSACTION
       End;


--Exec NON_ALCOHOL_SALES_REPORT @Start_Date='09/10/14',@End_Date='09/21/14'
```

This screenshot demonstrates the successful generation of Sales report for Non- Alcohol product.



This screenshot demonstrates the failed attempt of generation of Sales report for Non-Alcohol product.



19 – The system must be able to display total revenue for a day.

```
--Daily Revenue Report
IF OBJECT_ID ('Daily_Revenue_Report', 'P') IS NOT NULL
    DROP PROCEDURE Daily_Revenue_Report;
GO

CREATE PROCEDURE Daily_Revenue_Report
@Sales_date datetime

As
Begin
Begin transaction
--To check if any transaction occured on given date parameter

if not exists (select * from t_Acct_Sales where @Sales_date=acct_datetime)
       Begin
              Select 'Error! No Transaction Found.'
       end
--If transaction exists select--
If exists (select * from t_Acct_Sales where  @Sales_date=acct_datetime)
BEGIN TRY


Select ta.pro_id,tp.pro_name, Sum(ta.acct_qty*ta.acct_qty) As Net_Sales
from t_product tp join t_acct_sales ta on ta.pro_id=tp.pro_id
where @Sales_date=ta.acct_datetime
group by ta.pro_id,tp.pro_name
Order by ta.pro_id
END TRY
--Error Check
BEGIN CATCH
    SELECT
        ERROR_NUMBER() AS ErrorNumber,
        ERROR_SEVERITY() AS ErrorSeverity,
        ERROR_STATE() as ErrorState,
        ERROR_PROCEDURE() as ErrorProcedure,
        ERROR_LINE() as ErrorLine,
        ERROR_MESSAGE() as ErrorMessage;

IF @@TRANCOUNT > 0
        ROLLBACK TRANSACTION;
End Catch;

IF @@TRANCOUNT > 0
    COMMIT TRANSACTION
       End;

--Exec Daily_Revenue_Report @Sales_date='2014-12-06'
```

This screenshot demonstrates the successful generation of Revenue report based on POS date.

```
Exec Daily_Revenue_Report @Sales_date='2014-10-21'
```

100 %

Results | Messages

	pro_id	pro_name	Net_Sales
1	1	Bud Light	225
2	2	Red Bull	64
3	3	Bud Lime	3649
4	4	Corona	500
5	6	Coke	441

This screenshot demonstrates the s failed attempt of generation of Revenue report based on POS date.

```
Exec Daily_Revenue_Report @Sales_date='2014-12-06'
```

100 %

Results | Messages

	(No column name)
1	Error! No Transaction Found.

20 - The system must display monthly sales reports.

```
--Sales_Report

IF OBJECT_ID ('Sales_REPORT', 'P') IS NOT NULL
    DROP PROCEDURE Sales_REPORT;
GO

CREATE PROCEDURE Sales_REPORT
@Start_Date datetime,
@End_Date datetime
AS
Begin
Begin transaction
--To check if any transaction occured on given date parameter
If not exists (select *  from t_POS_Sales where pos_datetime between @Start_Date and @End_Date)
       Begin
              Select 'Error! No Report Found'
       end
--If transaction exists select--
If exists (select * from t_POS_Sales where pos_datetime between @Start_Date and @End_Date)
       Begin Try

 SELECT tp.pro_id,tp.pos_qty,SUM(tp.pro_price * tp.pos_qty)as Total_Sales,p.pro_name
    FROM T_POS_SALES tp
       join t_product p
       on tp.pro_id=p.pro_id join t_type tt
       on p.ty_id=tt.ty_id
       where tp.pos_datetime between @Start_Date and @End_Date
       Group by tp.pro_id,pos_qty,p.pro_name
       Order by tp.pro_id;
END TRY
--Error Check
BEGIN CATCH
    SELECT
        ERROR_NUMBER() AS ErrorNumber,
        ERROR_SEVERITY() AS ErrorSeverity,
        ERROR_STATE() as ErrorState,
        ERROR_PROCEDURE() as ErrorProcedure,
        ERROR_LINE() as ErrorLine,
        ERROR_MESSAGE() as ErrorMessage;

IF @@TRANCOUNT > 0
        ROLLBACK TRANSACTION;
End Catch;

IF @@TRANCOUNT > 0
    COMMIT TRANSACTION
END;

--Exec Sales_REPORT @Start_Date='09/11/14',@End_Date='10/10/14'
```

This screenshot demonstrates the successful generation of Sales report based on POS date.

```
Exec Sales_REPORT @Start_Date='09/11/14',@End_Date='11/10/14'
```

100 %

Results | Messages

	pro_id	pos_qty	Total_Sales	pro_name
1	1	15	82.50	Bud Light
2	2	8	32.00	Red Bull
3	3	7	31.50	Bud Lime
4	3	60	270.00	Bud Lime
5	4	10	55.00	Corona
6	4	20	110.00	Corona
7	6	21	73.50	Coke

This screenshot demonstrates the failed attempt of generation of Sales report based on POS date.

```
Exec Sales_REPORT @Start_Date='09/11/14',@End_Date='10/10/14'
```

100 %

Results | Messages

	(No column name)
1	Error! No Report Found

21- The system must display monthly purchase reports.

```
--Purchase_Report
IF OBJECT_ID ('Purchase_REPORT', 'P') IS NOT NULL
    DROP PROCEDURE PURCHASE_REPORT;
GO

CREATE PROCEDURE PURCHASE_REPORT
@Start_Date datetime,
@End_Date datetime
AS
Begin
Begin transaction
--To check if any transaction occurred on given date parameter
if not exists (select *  from t_purchase where pur_date between @Start_Date and @End_Date)
       Begin
              Select 'Error! No Report Found'
       end
--If transaction exists select--
If exists (select * from t_purchase where pur_date between @Start_Date and @End_Date)
       Begin Try
        SELECT tv.ven_id,tv.ven_name,tp.pur_id,tp.pro_id, sum(tp.pur_qty * tp.pur_unt_price) as
Total_Paid,
        p.pro_name, p.pro_id
        from t_vendor tv join t_purchase tp on tv.ven_id=tp.ven_id
        join t_product p on p.pro_id=tp.pro_id
        where tp.pur_date between @Start_Date and @End_Date
        group by tv.ven_id,tv.ven_name,tp.pur_id,tp.pro_id, p.pro_name, p.pro_id
        order by tv.ven_id;

END TRY
--Error Check
BEGIN CATCH
    SELECT
        ERROR_NUMBER() AS ErrorNumber,
        ERROR_SEVERITY() AS ErrorSeverity,
        ERROR_STATE() as ErrorState,
        ERROR_PROCEDURE() as ErrorProcedure,
        ERROR_LINE() as ErrorLine,
        ERROR_MESSAGE() as ErrorMessage;

IF @@TRANCOUNT > 0
        ROLLBACK TRANSACTION;
End Catch;

IF @@TRANCOUNT > 0
    COMMIT TRANSACTION
END;

--Exec Purchase_REPORT @Start_Date='09/10/14',@End_Date='11/21/14'
```

This screenshot demonstrates the successful generation of Purchase report based on purchase date.

```
Exec Purchase_REPORT @Start_Date='06/10/14',@End_Date='11/21/14'
```

100 %

Results | Messages

	ven_id	ven_name	pur_id	pro_id	Total_Paid	pro_name	pro_id
1	1	Towson-Sunbelt Group.	5	3	375.00	Bud Lime	3
2	1	Towson-Sunbelt Group.	6	4	600.00	Corona	4
3	1	Towson-Sunbelt Group.	15	6	364.00	Coke	6
4	1	Towson-Sunbelt Group.	16	7	187.50	Diet Coke	7
5	1	Towson-Sunbelt Group.	17	8	612.50	Monster	8
6	1	Towson-Sunbelt Group.	1	2	1487.50	Red Bull	2
7	2	Balto-Churchill, LTD.	2	1	420.00	Bud Light	1

This screenshot demonstrates the failed attempt of generation of Purchase report based on purchase date.

```
Exec Purchase_REPORT @Start_Date='06/11/14',@End_Date='10/11/14'
```

100 %

Results | Messages

	(No column name)
1	Error! No Report Found

22 – The system shall keep historical data.

The following code will create a stored procedure that will be run prior to purging the pos table to populate a permanent table which will keep historical information.

```
/* CHECK IF PROCEDURE EXISTS, DROP AND RECREATE IT */
IF OBJECTPROPERTY(object_id('dbo.sp_update_acct_sales'), N'IsProcedure') = 1
DROP PROCEDURE [dbo].[sp_update_acct_sales]
GO
CREATE PROCEDURE dbo.sp_update_acct_sales
AS
BEGIN

BEGIN TRANSACTION

/* INSERT DATA FROM THE POS TERMINAL INTO THE PERMANENT ACCOUNTING TABLE */
INSERT INTO T_ACCT_SALES (ACCT_DATETIME, ACCT_QTY, CUS_ID, PRO_ID, ACCT_PRICE)
SELECT pos_datetime, pos_qty, cus_id, pro_id, pro_price
FROM T_POS_SALES
WHERE POS_PAID = 1;

/* ROLLBACK ON ERROR */
IF @@error <> 0

	BEGIN
		ROLLBACK TRANSACTION
		SELECT ' There was a problem migrating the sales information'
		RETURN
	END

COMMIT TRANSACTION;

SELECT * FROM T_ACCT_SALES;
END
```

This screen capture demonstrates the successful insert of data from the pos table.

```
1 --exec sp_update_acct_sales
```

100 %

Results | Messages

	acct_id	acct_datetime	acct_qty	cus_id	pro_id	acct_price

	acct_id	acct_datetime	acct_qty	cus_id	pro_id	acct_price
1	1	2014-11-28 16:20:23.603	2	1	3	2.50
2	2	2014-11-28 16:20:23.630	2	1	3	2.50
3	3	2014-11-28 16:20:23.700	1	5	5	2.50
4	4	2014-11-28 16:20:23.700	2	1	4	2.50
5	5	2014-11-28 16:20:23.713	1	5	5	2.50
6	6	2014-11-28 16:45:19.350	2	1	3	2.50
7	7	2014-11-28 16:47:09.283	25	1	6	5.00
8	8	2014-11-28 17:58:29.190	25	1	6	5.75
9	9	2014-11-30 14:26:17.460	2	1	3	2.50
10	10	2014-12-01 21:31:54.820	25	5	4	2.50

23 - The system must insert new purchases and update the inventory levels in the product table.

This stored procedure is used to purchase inventory. It validates that the vendor and product exists. It also validates that the purchase quantity is greater than zero. It writes to the purchase table and updates the inventory level in the products table.

```
use gibino
if objectproperty(object_id('dbo.sp_purchase_inventory'), N'IsProcedure') = 1
drop procedure dbo.sp_purchase_inventory
go

create procedure sp_purchase_inventory
(@pro_id int, @ven_id int, @pur_qty int, @pur_unt_pricenumeric(5,2), @pur_date     datetime)
as
begin

--Checks that vendor and product exist in t_product
if not exists (select * from t_product where ven_id=@ven_id and pro_id=@pro_id)
       begin
              select 'Vendor and product do not exist in the product table'
              return
end
else

--Checks that quantity purchase is >0
if @pur_qty <=0
       begin
              select 'A quantity greater than 0 must be purchased.'
              return
end

--Inserts values into t_purchase table
begin transaction
        insert into t_purchase
        (pro_id, ven_id, pur_qty, pur_unt_price, pur_date)
        values
        (@pro_id, @ven_id, @pur_qty, @pur_unt_price, @pur_date);
       --Updates inventory amount in t_product
       update t_product
       set pro_instock=pro_instock+@pur_qty
       where pro_id=@pro_id

--Message for rollback
if @@error<>0
       begin
              rollback transaction
              select 'Purchase was not completed.'
              return
       end

--Message for success
commit transaction
select   'Purchase successfully added for ', @pur_qty, 'units of ', (select pro_name from
t_product where pro_id=@pro_id), ' at ', @pur_unt_price, ' each.'
end
```

This screenshot shows a successful purchase of a valid product.

```
--Purchase legitimate inventory / should work
use gibino exec sp_purchase_inventory 2,1,425,3.5,'2014-11-26 14:50:20.937'
```

100 %

Results | Messages

	(No column name)	(No column name)	(No column name)	(No column name)	(No column name)	(No column name)	(No column name)
1	Purchase successfully added for	425	units of	TB-Uni Lager	at	3.50	each.

These screenshots show the t_product table before and after the above-mentioned successful purchase.

100 %

Results | Messages

	pro_id	pro_name	pro_instock	ty_id	pro_base	ven_id
1	1	TB-Tiger Tail Ale	0	1	4.50	1
2	2	TB-Uni Lager	312	1	4.50	1

100 %

Results | Messages

	pro_id	pro_name	pro_instock	ty_id	pro_base	ven_id
1	1	TB-Tiger Tail Ale	0	1	4.50	1
2	2	TB-Uni Lager	737	1	4.50	1

This screenshot shows a failed attempt to purchase an invalid product.

```
--Purchase invalid item / should fail
use gibino exec sp_purchase_inventory 55564,1,55,3.5,'2014-11-26 14:50:20.937'
```

100 %

Results | Messages

	(No column name)
1	Vendor and product do not exist in the product t...

This screenshot shows a failed attempt to purchase an invalid amount (<=0) of a product.

```
--Purchase inventory 0 qauntity / should fail
use gibino exec sp_purchase_inventory 2,1,-50,3.5,'2014-11-26 14:50:20.937'
```

100 %

Results | Messages

	(No column name)
1	A quantity greater than 0 must be purchased.

24 – The system must reset prices to their base levels.

The following code will create a stored procedure to reset product prices.

```
/* CHECK IF PROCEDURE EXISTS, DROP AND RECREATE IT */
IF OBJECTPROPERTY(object_id('dbo.sp_reset_price'), N'IsProcedure') = 1
DROP PROCEDURE [dbo].[sp_reset_price]
GO
CREATE PROCEDURE dbo.sp_reset_price
AS
BEGIN
BEGIN TRANSACTION
/* RESET PRICES BACK TO THEIR ORIGINAL PRICES */
UPDATE pri
SET pri.pro_price = pro.pro_base
FROM t_price pri INNER JOIN t_product pro
ON pri.pro_id = pro.pro_id;

IF @@error <> 0

       BEGIN
              ROLLBACK TRANSACTION
              SELECT ' Sale was NOT completed'
              RETURN
       END

COMMIT TRANSACTION
SELECT * FROM T_PRICE;
END
```

This screen capture demonstrates the successful reset of product prices. Product 6 was at $5.75 after a price update, now all prices are reflective of their base price in T_PRODUCT.

```
1  --exec sp_reset_price
```

00 %

Results | Messages

	pri_id	pro_id	pro_price
1	1	1	4.50
2	2	2	4.50
3	3	3	2.50
4	4	4	2.50
5	5	5	2.50
6	6	6	5.00
7	7	7	6.00
8	8	8	5.00
9	9	9	5.50
10	10	10	5.00

These screenshots show the t_price table before and after a reset.

```
1 --exec sp_reset_price
```

100 %

Results | Messages

	pri_id	pro_id	pro_price
1	1	1	4.50
2	2	2	4.50
3	3	3	2.50
4	4	4	2.50
5	5	5	2.50
6	6	6	6.50
7	7	7	6.00
8	8	8	5.00

	pri_id	pro_id	pro_price
1	1	1	4.50
2	2	2	4.50
3	3	3	2.50
4	4	4	2.50
5	5	5	2.50
6	6	6	5.00
7	7	7	6.00
8	8	8	5.00

25 - The system must be able to add new product types without duplicates.

This stored procedure inserts a new type (i.e., a volume and category of beverage). It validates that the type does not already exist in the system.

```
use gibino
if objectproperty(object_id('dbo.sp_add_pro_type'), N'IsProcedure') = 1
drop procedure dbo.sp_add_pro_type
go

create procedure sp_add_pro_type
(@ty_description varchar(50), @ty_restricted int)
as
begin

--Checks for duplicate product type
if exists (select * from t_type where upper(replace(ty_description, ' ',
''))=upper(replace(@ty_description, ' ', '')))
       begin
              select UPPER(@ty_description), 'already exists as a product type in the system.'
              return
end

--Inserts vendor information into t_vendor table
begin transaction
        insert into t_type
       (ty_description, ty_restricted)
        values
       (@ty_description, @ty_restricted)

--Message for rollback
if @@error<>0
       begin
              rollback transaction
              select @ty_description, ' not added to the system.'
              return
       end

--Message for success
commit transaction
select @ty_description, ' was successfully added to the system.'
end
```

This screenshot shows a successful insertion of a new type of product.

```
--TEST Insert new type / should pass
use gibino execute sp_add_pro_type '48 OZ BEER BOTTLE', 1
```

100 %

Results | Messages

	(No column name)	(No column name)
1	48 OZ BEER BOTTLE	was successfully added to the system.

These screenshots show the t_type table after the above-mentioned successful insert of a new product type.

100 %

Results | Messages

	ty_id	ty_description	ty_restricted
1	1	16 OZ BEER BOTTLE	1
2	2	12 OZ SODA CAN	0
3	3	64 OZ BEER GROWLER	1
4	4	12 OZ BEER CAN	1

100 %

Results | Messages

	ty_id	ty_description	ty_restricted
1	1	16 OZ BEER BOTTLE	1
2	2	12 OZ SODA CAN	0
3	3	64 OZ BEER GROWLER	1
4	4	12 OZ BEER CAN	1
5	5	48 OZ BEER BOTTLE	1

This screenshots shows a failed attempt to insert a type of product which already exists in the system.

```
--TEST Insert existing type / should fail
use gibino execute sp_add_pro_type '   16 OZ BEeR BOttLE  ', 0
```

100 %

Results | Messages

	(No column name)	(No column name)
1	16 OZ BEER BOTTLE	already exsists as a product type in the system.

26 - The system must be able to insert new customers without duplicates.

This stored procedure is used to insert new customers. It validates that the system does not already contain a customer with the same first name, last name, and date of birth.

```
use gibino
if objectproperty(object_id('dbo.sp_add_customer'), N'IsProcedure') = 1
drop procedure dbo.sp_add_customer
go

create procedure sp_add_customer
(@cus_dob datetime, @cus_street1 varchar(30), @cus_street2 varchar(30), @cus_city varchar(30),
@cus_state char(2), @cus_zip numeric(5,0),
@cus_email varchar(50), @cus_fname varchar(25), @cus_lname varchar(30), @cus_mi varchar(1),
@cus_suffix varchar(5))
as
begin

--Checks for duplicate customer names with the same date of birth.
if exists (
select * from t_customer where upper(replace(cus_fname, ' ', '' ))=upper(replace(@cus_fname, '
', '' ))
and upper(replace(cus_lname, ' ', '' ))=upper(replace(@cus_lname, ' ', '' ))
and cast(cus_dob as date)=cast(@cus_dob as date)
)
	begin
		select @cus_fname, @cus_lname, 'already exists in this system as a customer.'
		return
end

--Inserts vendor information into t_customer table
begin transaction
	 insert into t_customer
	(cus_dob, cus_street1, cus_street2, cus_city, cus_state, cus_zip,
cus_email, cus_fname, cus_lname, cus_mi, cus_suffix)
	 values
(@cus_dob, @cus_street1, @cus_street2, @cus_city, @cus_state, @cus_zip,
@cus_email, @cus_fname, @cus_lname, @cus_mi, @cus_suffix)

--Message for rollback
if @@error<>0
	begin
		rollback transaction
		select 'Customer, ', @cus_fname, @cus_lname, ' not added.'
		return
	end

--Message for success
commit transaction
select 'Customer, ', @cus_fname, @cus_lname, ' was added.'
end
```

This screenshot shows the successful insertion of a new customer.

```
--Add new customer / should pass
exec sp_add_customer '1947-01-01 00:00:00.000','5588 York Road','','Towson','MD',21252,'jdoe@gmail.com','Jane','Doe','','PhD'
```

100 %

Results | Messages

	(No column name)	(No column name)	(No column name)	(No column name)
1	Customer,	Jane	Doe	was added.

These screenshots show the t_customer table before and after the above-mentioned successful insert.

100 %

Results | Messages

	cus_id	cus_dob	cus_street1	cus_street2	cus_city	cus_state	cus_zip	cus_email	cus_fname	cus_lname	cus_mi	cus_suffix
1	1	1995-01-01 00:00:00.000	123 York Road		Towson	MD	21252	joesmith@gmail.com	Joe	Smith		Dr.
2	2	1980-01-01 00:00:00.000	456 ABC Bld.		Baltimore	MD	21239	samdan@yahoo.com	Sam	Daniel		
3	3	1950-01-01 00:00:00.000	789 Old Terrace		Baltimore	MD	21239	howdee.com	Daggly	Howard		
4	4	1962-01-01 00:00:00.000	7778 Route 99		Baltimore	MD	21239	freddy@yahoo.com	Hoord	Fred		
5	5	1978-01-01 00:00:00.000	3435 Joppa Road		Baltimore	MD	21239	jlanen@yahoo.com	Lane	Jane		

100 %

Results | Messages

	cus_id	cus_dob	cus_street1	cus_street2	cus_city	cus_state	cus_zip	cus_email	cus_fname	cus_lname	cus_mi	cus_suffix
1	1	1995-01-01 00:00:00.000	123 York Road		Towson	MD	21252	joesmith@gmail.com	Joe	Smith		Dr.
2	2	1980-01-01 00:00:00.000	456 ABC Bld.		Baltimore	MD	21239	samdan@yahoo.com	Sam	Daniel		
3	3	1950-01-01 00:00:00.000	789 Old Terrace		Baltimore	MD	21239	howdee.com	Daggly	Howard		
4	4	1962-01-01 00:00:00.000	7778 Route 99		Baltimore	MD	21239	freddy@yahoo.com	Hoord	Fred		
5	5	1978-01-01 00:00:00.000	3435 Joppa Road		Baltimore	MD	21239	jlanen@yahoo.com	Lane	Jane		
6	6	1947-01-01 00:00:00.000	5588 York Road		Towson	MD	21252	jdoe@gmail.com	Jane	Doe		PhD

This screenshot shows a failed attempt to insert a customer with the same first name, last name, and date of birth as an existing customer.

```
--Add exisitng customer / should fail
exec sp_add_customer '1980-01-01 00:00:00.000','ABC Bld.','','Baltimore','MD',21239,'samdan@yahoo.com','Sam','Daniel','',''
```

100 %

Results | Messages

	(No column name)	(No column name)	(No column name)
1	Sam	Daniel	already exsists in this system as a customer.

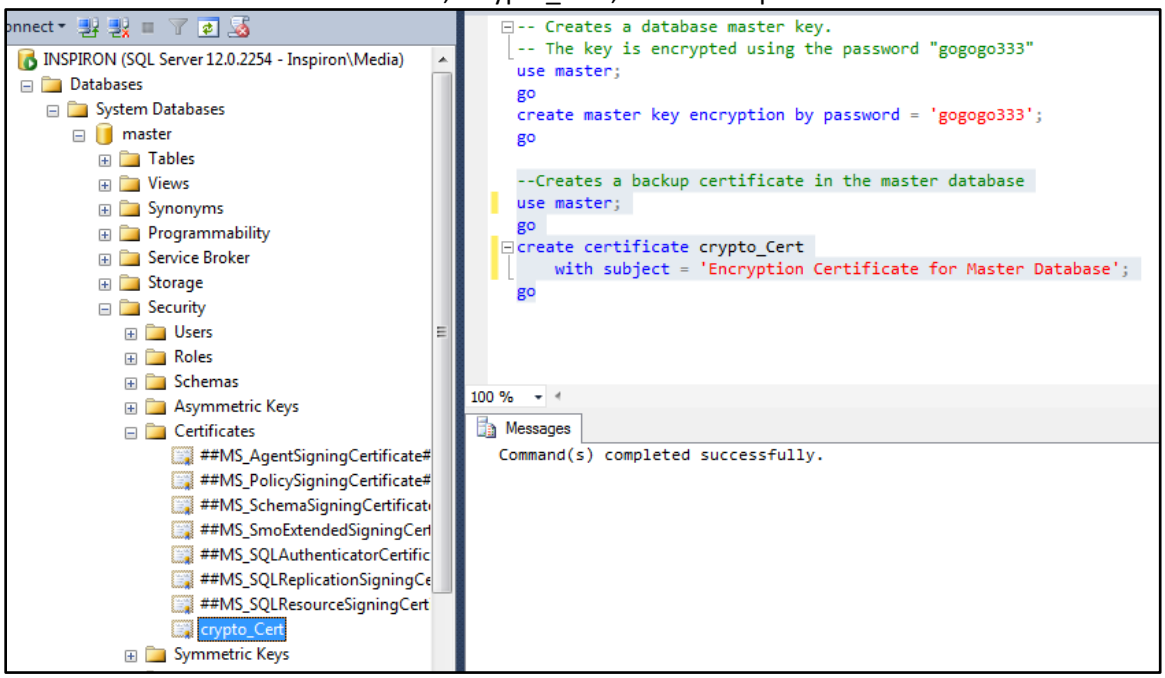
27 - The system must be able to create encrypted ad hoc backups, for transferring across potentially insecure connections.

In the event that an ad hoc backup of the database is made, it should be secured with encryption before being sent across a potentially insecure connection or being transferred to unencrypted physical media.

This code is used to create a database master key and then to create a backup certificate in the master database.

```
-- Creates a database master key.
-- The key is encrypted using the password "gogogo333"
use master;
go
create master key encryption by password = 'gogogo333';
go

--Creates a backup certificate in the master database
use master
go
create certificate cypto_Cert
   with subject = 'Encryption Certificate for Master Database';
go
```

This screenshot shows the certificate, “crypto_Cert,” in the left panel.

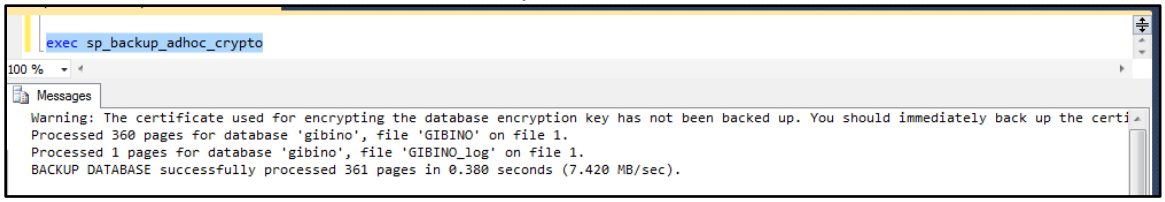


This stored procedure allows the user to make a compressed, encrypted, ad hoc backup of the gibino database.

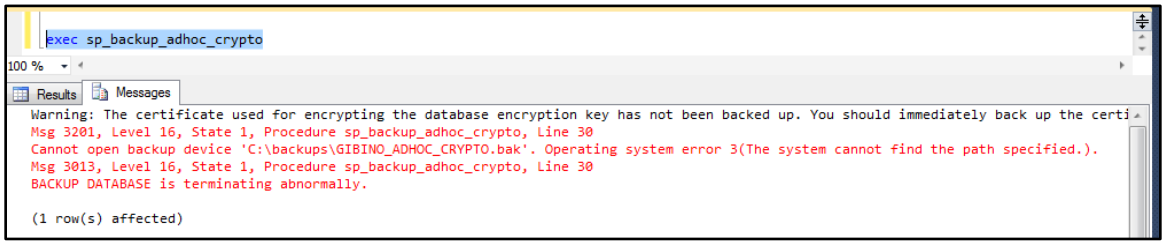
```
--This procedure creates a compress, encrypted, ad hoc copy of the gibino database
--It should be used if a copy of the database has to be taken off site
use gibino
go
```

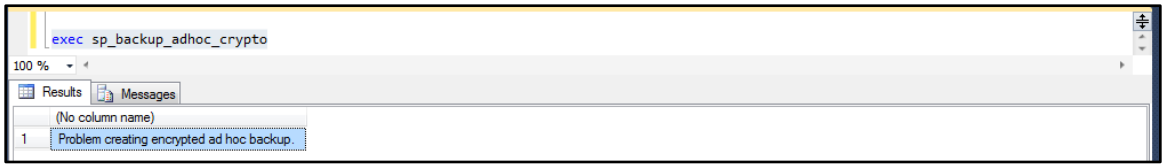
```sql
if objectproperty(object_id('sp_backup_adhoc_crypto'), N'IsProcedure') = 1
drop procedure sp_backup_adhoc_crypto
go

create proc sp_backup_adhoc_crypto
as
begin
	backup database [gibino]
	to disk = 'C:\backups\GIBINO_ADHOC_CRYPTO.bak'
	with copy_only,	compression, encryption
	(algorithm = AES_256, server certificate = crypto_Cert)

--Message for problem
if @@error<>0
	begin
		select 'Problem creating encrypted ad hoc backup.'
		return
	end
end
```

This screenshot shows the successful backup run.

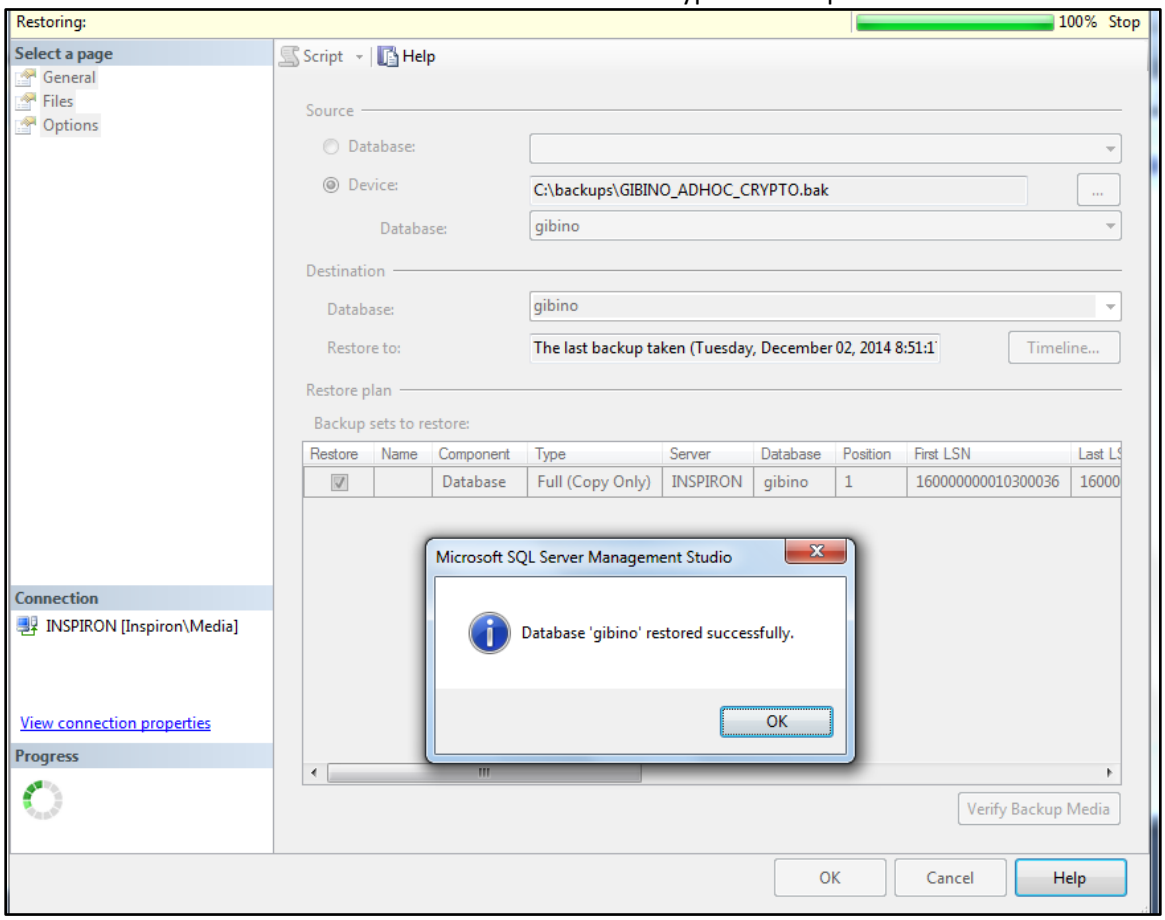


These screenshots show the output for a failed run, which was caused by the specified directory being unavailable.





This screenshot shows a successful restore from the encrypted backup.



Sample Data

T_VENDOR

	ven_id	ven_name	ven_street1	ven_street2	ven_city	ven_state	ven_zip	ven_phone	ven_email	ven_contact
1	1	TOWSON BREW GUYS	5852 York Road		Towson	MD	21252	443-458-8522	towson@gmail.com	Lisa Howard
2	2	BALTIMORE SODA POP	987 Penn Blvd.		Baltimore	MD	21239	443-100-8970	bchill@gmail.com	Jason Doe
3	3	GIBINO BREWING	123 Brewers Blvd.		Baltimore	MD	21239	443-200-8970	gb_brewery@gmail.com	Gib Ino
4	4	MONASTIC LAGER COMPANY	123 Monastery Road.		Baltimore	MD	21239	410-288-5566	those_monks@gmail.com	Greg Unkl

T_PRODUCT

	pro_id	pro_name	pro_instock	ty_id	pro_base	ven_id
1	1	TB-Tiger Tail Ale	0	1	4.50	1
2	2	TB-Uni Lager	312	1	4.50	1
3	3	BSP-Ginger Beer	185	2	2.50	2
4	4	BSP-Root Beer	320	2	2.50	2
5	5	BSP-Orange Soda	255	2	2.50	2
6	6	GB-Primal	18	1	5.00	3
7	7	GB-Hoppy	25	1	6.00	3
8	8	GB-Porter	20	1	5.00	3
9	9	GB-Lager	15	1	5.50	3
10	10	GB-Ale	20	1	5.00	3

T_TYPE

	ty_id	ty_description	ty_restricted
1	1	16 OZ BEER BOTTLE	1
2	2	12 OZ SODA CAN	0
3	3	64 OZ BEER GROWLER	1
4	4	12 OZ BEER CAN	1

T_PURCHASE

	pur_id	pro_id	ven_id	pur_qty	pur_unt_price	pur_date
1	1	2	1	425	3.50	2014-11-29 13:27:14.900
2	2	1	2	240	1.75	2014-11-29 13:27:14.903
3	3	3	3	240	1.75	2014-11-29 13:27:14.903
4	4	4	3	240	2.50	2014-11-29 13:27:14.903
5	5	5	3	240	3.00	2014-11-29 13:27:14.903
6	6	6	3	240	3.50	2014-11-29 13:27:14.903
7	7	7	3	240	2.25	2014-11-29 13:27:14.907

T_ACCT_SALES

	acct_id	acct_datetime	acct_qty	cus_id	pro_id	acct_price
1	1	2014-11-23 17:25:02.650	40	2	1	5.50
2	2	2014-11-22 17:25:02.650	350	1	2	4.00
3	3	2014-11-21 17:25:02.650	15	2	2	5.50
4	4	2014-11-02 17:25:02.650	220	1	1	4.00
5	5	2014-11-23 17:25:02.650	15	2	3	5.50
6	6	2014-11-15 17:25:02.650	8	1	4	4.00
7	7	2014-11-23 17:25:02.650	155	2	3	5.50
8	8	2014-11-23 17:25:02.650	188	1	4	4.00
9	9	2014-11-23 17:25:02.650	15	2	6	5.50
10	10	2014-11-13 17:25:02.650	8	1	2	2.00
11	11	2014-11-13 17:25:02.650	155	2	6	5.50
12	12	2014-11-23 17:25:02.650	81	1	7	2.00
13	13	2014-11-13 17:25:02.650	152	2	7	4.50
14	14	2014-11-12 17:25:02.650	8	1	2	4.00
15	15	2014-11-11 17:25:02.650	15	2	9	5.50
16	16	2014-11-11 17:25:02.650	8	1	10	4.00
17	17	2014-11-11 17:25:02.650	15	2	1	5.50
18	18	2014-11-11 17:25:02.650	8	1	2	4.00

T_PRICE

	pri_id	pro_id	pro_price
1	1	4	5.50
2	2	2	6.00
3	3	3	5.00
4	4	5	5.75
5	5	2	5.25
6	6	6	5.00

T_CUSTOMER

	cus_id	cus_dob	cus_street1	cus_street2	cus_city	cus_state	cus_zip	cus_email	cus_fname	cus_lname	cus_mi	cus_suffix
1	1	1995-01-01 00:00:00.000	123 York Road		Towson	MD	21252	joesmith@gmail.com	Joe	Smith		Dr.
2	2	1980-01-01 00:00:00.000	456 ABC Bld.		Baltimore	MD	21239	samdan@yahoo.com	Sam	Daniel		
3	3	1950-01-01 00:00:00.000	789 Old Terrace		Baltimore	MD	21239	howdee.com	Daggly	Howard		
4	4	1962-01-01 00:00:00.000	7778 Route 99		Baltimore	MD	21239	freddy@yahoo.com	Hoord	Fred		
5	5	1978-01-01 00:00:00.000	3435 Joppa Road		Baltimore	MD	21239	jlanen@yahoo.com	Lane	Jane		

T_SALES_INFO

	si_id	pro_id	qty_sold
1	1	1	15
2	2	2	4
3	3	3	1
4	4	2	3
5	5	3	11
6	6	1	4

T_SALES_PERC

	sp_id	pro_id	pct_of_sales
1	1	1	20
2	2	2	40
3	3	3	10
4	4	5	5
5	5	6	20
6	6	7	5

T_POS_SALES

	pos_id	pos_datetime	pos_qty	cus_id	pro_id	pro_price	pos_paid
1	1	2014-11-29 13:35:57.343	15	2	1	5.50	1
2	2	2014-11-29 13:35:57.343	8	1	2	4.00	1
3	3	2014-11-29 13:35:57.343	8	1	2	4.00	0
4	4	2014-11-29 13:35:57.343	7	2	3	5.00	0
5	5	2014-11-29 13:35:57.347	12	2	3	5.00	0