EBUS3030 Assignment 1

Steven Karmaniolos c3160280@uon.edu.au
Jay Rovacsek c3146220@uon.edu.au
Jacob Litherland c3263482@uon.edu.au
Edward Lonsdale c3252144@uon.edu.au

September 1, 2018

Business Intelligence - EBUS3030
Assignment 1

Due: Assignment One TurnItIn drop folder by 12 noon on Thursday 6th September
Paper copy at the beginning of week 6 workshop.

Assignment Outcomes

This assignment requires multiple outputs to be created to exhibit your understanding of business intelligence/data analysis through an example 'real world' question that is comparable to what you may be asked of you as you become an IT professional.

Key outcomes to be delivered are: Data Modelling of the provided dataset, Extract Transform Load (ETL) processing undertaken to make the data usable, the Output of your analysis, a Report summarising your findings and a presentation to the class of your work. The presentation is expected to concentrate more on your findings/recommendations as if it were a situation where you are presenting the response to the head sales executive's question.

Assignment Question

The head Sales Executive of 'BIA Inc' comes to you as the lead Business/Data Analyst and asks you to help with a problem they have.

"I've heard that people aren't motivated at the moment and sales aren't as good as we had hoped. To try and provide incentives for staff, I want to provide an award (and probably associated cash prize) to my best performer for sales from this Office, I need you to tell me who that is?"

"As part of your response I want you to provide the justification as to why the particular sales officer was selected because we need governance over things like this.

…. By the way, we don't currently have any of this information stored centrally in a database thingy, but I have gotten the Office Business Manager to collate a summary of the recent sales into a rough excel file that can be used as a starting basis. As part of the processes of getting me an answer on my best salesperson, can you also create a database as part of the preparation of the answer. We will then use that as the base of further reporting into the future. We haven't ever had people with your skills working with us before so I expect there will be lots of questions that will come up as we utilise your expertise."

Assignment Deliverables

Using the data file provided in Excel and associated notes about the data, (*AssOneData.xlsx* and *Datamart Business Notes*) you are required to complete the following elements as part of the assignment.

- Data Model
    - Using the information made available to you and your understanding of concepts around data mart design in the labs, design a "Sales" DataMart to store the information in a format that will allow the information to be expanded and one that would enable analysis to occur.
- Data Load Process undertaken
    - Provide an overview of the ETL/ELT process completed and what (if any) Quality Assurance processes you undertook as part of this.
    - Ensure you record any assumptions you have made as part of this component and your reasoning behind the assumption.
- Output of Analysis (including SQL used)
    - Once the data loaded and is available and ready for use, you need to create a set of sql scripts to be used to generate the results to the business question provided to you from the Head Sales Executive
    - Provide a snapshot of the raw results of your analysis that provides the basis of your recommendations
    - Ensure you record any assumptions you have made as part of this analysis component and your reasoning behind the assumption.

- Executive Summary in response to business question.
  - Provide a short Executive brief/summary that presents a clear concise response back to the Sales Executive's question about possible incentives to the best salesperson. This should clearly detail the recommendation and any key assumptions/restrictions the executive need to be aware of.

- Team Presentation
  - All members of the team need to participate in a (10-15 minute) presentation to be delivered as part of the lab in Week 6. This needs to be presented in a format as if you were summoned to the board room with the Head Sales Executive to provide a formal response to their question.
  - Please be aware that the Head Sales Executive may ask any of the team members questions as you present your analysis.

NB: As part of your responses, you should also specifically include any assumptions you have made throughout the process.

Breakup of assignment Marks (total course mark for assignment = Assignment Part A submission (20% + Presentation One (5%) = 25%.

Assignment Component	Percentage Allocation
Data Model	30%
ETL	10%
Base Analysis	30%
Executive Summary	10%
Team Presentation	20%
Assumptions	100%

Key Documents Required & Format

You are required to upload all files in a single zip file (including any presentation items for the team delivery within the lab) via blackboard to the Assignment One TurnItIn drop folder by 12 noon on Thursday 6th September. You will also be required to submit a paper copy of your deliverables at the workshop (make sure this is printed well before the workshop.

NB: Only 1 load per team only but it should contain all of the deliverable items in a .zip file.

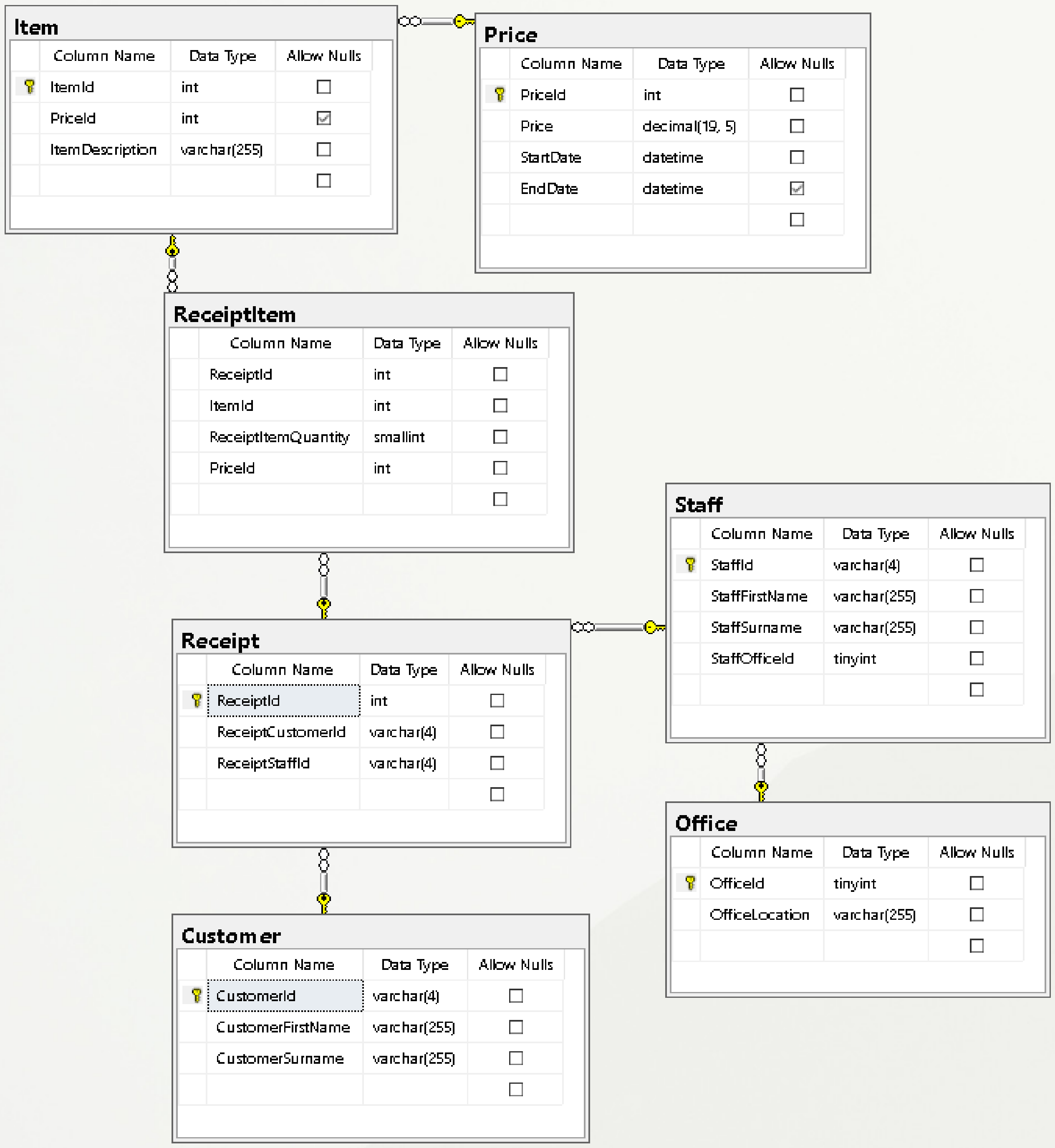
1 Datamart Business Notes

The following business rules were provided to be used in the context of this assignment:

* At BIA all customers interacts are in an online environment, there are no orders outside of electronic.
* Returning customers can provide POI information via the web interface and look up their record and that will flow with the sale.
* The sales associate can complete the order form/sale for the client.
* Each sale will have a receipt number/id.
* A receipt can have many line items.
* Each line item can only be for a single item, but the customer can purchase multiples of the same item.
* Where a customer has multiple line items, any sale with more than 5 row items (containing at least 5 different items) is provided a 15% discount.
* The system automatically handles the total for the sale by looking up the item, then multiplying the costs per item by number purchased, and then should store this final field total as a record in the system (but should also be able to see clearly sales that were provided a discount.
* Item prices can change at any point, and the price the customer pays is the amount listed for the item on the sale date. We need to keep a record of all item prices historically.
* Only 1 BIA sales assistant can be attributed to any receipt.

2 Data Model

The below data model is only a suggestion and is still subject to change into the future. A full create script can be found in the appendix



Item

	Column Name	Data Type	Allow Nulls
🔑	ItemId	int	☐
	PriceId	int	☑
	ItemDescription	varchar(255)	☐
			☐

Price

	Column Name	Data Type	Allow Nulls
🔑	PriceId	int	☐
	Price	decimal(19, 5)	☐
	StartDate	datetime	☐
	EndDate	datetime	☑
			☐

ReceiptItem

	Column Name	Data Type	Allow Nulls
	ReceiptId	int	☐
	ItemId	int	☐
	ReceiptItemQuantity	smallint	☐
	PriceId	int	☐
			☐

Staff

	Column Name	Data Type	Allow Nulls
🔑	StaffId	varchar(4)	☐
	StaffFirstName	varchar(255)	☐
	StaffSurname	varchar(255)	☐
	StaffOfficeId	tinyint	☐
			☐

Receipt

	Column Name	Data Type	Allow Nulls
🔑	ReceiptId	int	☐
	ReceiptCustomerId	varchar(4)	☐
	ReceiptStaffId	varchar(4)	☐
			☐

Office

	Column Name	Data Type	Allow Nulls
🔑	OfficeId	tinyint	☐
	OfficeLocation	varchar(255)	☐
			☐

Customer

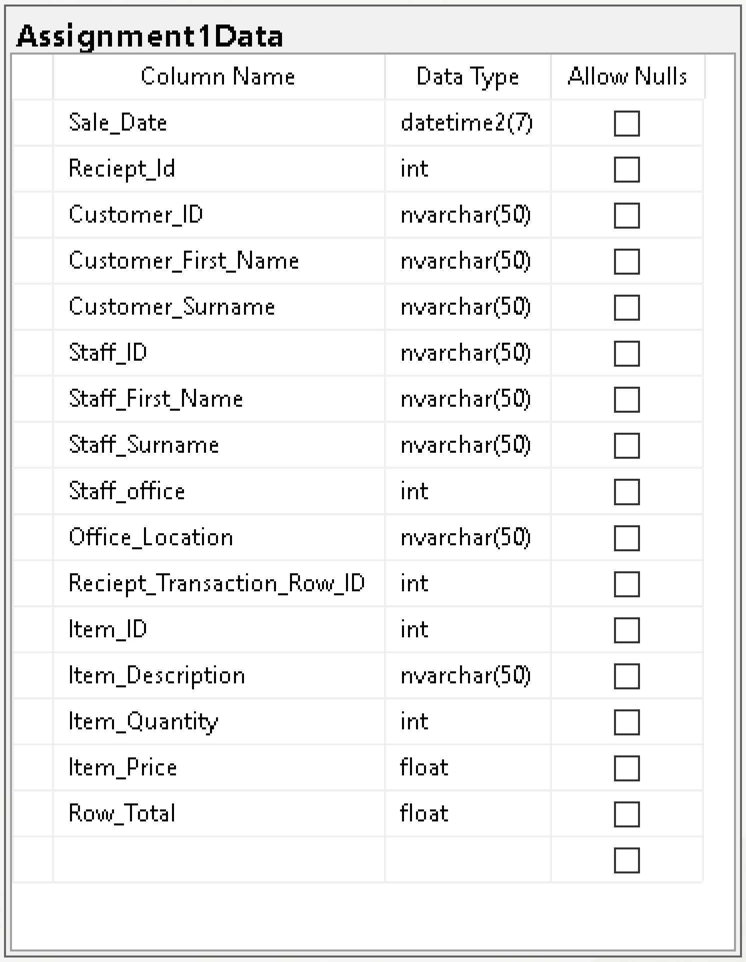
	Column Name	Data Type	Allow Nulls
🔑	CustomerId	varchar(4)	☐
	CustomerFirstName	varchar(255)	☐
	CustomerSurname	varchar(255)	☐
			☐

It must be noted that the structure of this data model is less than efficient, and it would be expected in a datamart situtation that only at lower levels of data would this schema remain responsive in the manner it is now, as the outline suggests the datamart is not necessarily the most suitable design for future use, however suits very well currently.

It would be expected that only at extremely large datasets would this model prove a bad design. In such cases a model more representative of the snowflake or star schemas would be heavily advised.

3 Data Load Process (ETL/ELT)

Initial import of the data supplied in the xlsx file generated a very basic table that allowed us to analyse the data for potential outliers, confirm the business requirements of the data and then create tables from which the data model was derived.
The Imported table structure was as follows:



Assignment1Data

Column Name	Data Type	Allow Nulls
Sale_Date	datetime2(7)	☐
Reciept_Id	int	☐
Customer_ID	nvarchar(50)	☐
Customer_First_Name	nvarchar(50)	☐
Customer_Surname	nvarchar(50)	☐
Staff_ID	nvarchar(50)	☐
Staff_First_Name	nvarchar(50)	☐
Staff_Surname	nvarchar(50)	☐
Staff_office	int	☐
Office_Location	nvarchar(50)	☐
Reciept_Transaction_Row_ID	int	☐
Item_ID	int	☐
Item_Description	nvarchar(50)	☐
Item_Quantity	int	☐
Item_Price	float	☐
Row_Total	float	☐
		☐

A decision to leave this initial import table as default was made to allow easy reference to the initally supplied excel data file.

In the following sections of Quality Assurance Processes, Assumptions and Reasoning and Base Analysis we intend to clarify the reasoning behind leaving the imported data in the default table suggested by SSMS.

3.1 Quality Assurance Processes

A number of queries were written to look for data which did not adhere to the spec outlined in business requirements and to ensure data was "clean" before entry. The first instance of potential issues were encountered with a basic python script which checked validity of column data, it was found that cells starting at B13777 to the end of file in the originally supplied excel file were formula values and not static values, this would not have caused an issue with importing into SSMS however certainly broke the script temporarily.

The next potential issue encountered was not until a suggested schema structure was complete and data was being scripted to be added to the new schema for analysis. The issue encountered was that receipt number 52136 seemed to be an incorrect entry, this was discovered when running the import query for the new schema:

```
INSERT INTO Receipt([ReceiptId], [ReceiptCustomerId],[ReceiptStaffId])
SELECT DISTINCT([Reciept_Id]),[Customer_ID],[Staff_ID]
FROM [Assignment1Data]
ORDER BY [Reciept_Id]
```

Which resulted in the error:

```
Violation of PRIMARY KEY constraint 'PK_Receipt'. Cannot insert duplicate key in object
'dbo.Receipt'. The duplicate key value is (52136).
```

Leading us to recognise that either one of the entries could be incorrect, therefore best to investigate both records of the customer Id against the rest of the database:

```
SELECT * FROM Assignment1Data WHERE Customer_ID='C32' AND Staff_ID='S15' AND
Sale_Date='2017-11-12 00:00:00.0000000';

SELECT * FROM Assignment1Data WHERE Customer_ID='C13' AND Staff_ID='S4' AND
Sale_Date='2017-12-30 00:00:00.0000000';
```

When both queries were performed it was apparent that the data associated with C32 was the likely broken record and modification of the data occured:

```
UPDATE Assignment1Data
SET Reciept_Id=51585,
Reciept_Transaction_Row_ID=(
    SELECT MAX(Reciept_Transaction_Row_ID)+1
    FROM Assignment1Data
    WHERE Reciept_Id=51585)
WHERE Customer_ID='C32'
AND Staff_ID='S15'
AND Sale_Date='2017-11-12 00:00:00.0000000'
AND Item_ID='14';
```

The next issue arose when again, attempting to run the aforementioned query to import into the new Receipt table, this time not one stray record was found, but a complete collision on the ReceiptId of 52137, this time as neither record seemed to have records that were correct, it was decided to move one to the maximum ReceiptId + 1:

```
UPDATE Assignment1Data SET Reciept_Id=(SELECT MAX(Reciept_Id)+1 FROM Assignment1Data)
WHERE Customer_ID='C27' AND Staff_ID='S4' AND Sale_Date='2017-12-30 00:00:00.0000000';
```

The same issue was replicated on ReceiptId 52138, resolved via:

```
UPDATE Assignment1Data SET Reciept_Id=(SELECT MAX(Reciept_Id)+1 FROM Assignment1Data)
WHERE Customer_ID='C30' AND Staff_ID='S19' AND Sale_Date='2017-05-16 00:00:00.0000000';
```

At this point we recognised the broken data likely continued for a while, and evaluated our hypothesis by looking at the original excel file. It turned out that data with ReceiptId from 52137-52145 was all broken in the same manner. The following query shows this well:

```
SELECT Reciept_Id, Customer_ID,Staff_ID FROM Assignment1Data
WHERE Reciept_Id BETWEEN 52137 AND 52150
GROUP BY Reciept_Id, Customer_ID,Staff_ID
ORDER BY Reciept_Id;
```

In order to clean this data we looked at a number of potential methods, with an emphasis on avoiding effort in the task if possible but not breaking the data further, which to this point just appeared to be a collision of a number of receipts.
We knew a structure such as a CTE [3] would allow us to easily split distinct records which shared a receiptId and filter by a value such as row number.

```
WITH CTE AS
(
    SELECT ROW_NUMBER() OVER (ORDER BY Reciept_Id) AS RowNumber,
            Reciept_Id,
            Customer_ID,
            Staff_ID
    FROM  Assignment1Data
    WHERE Reciept_Id BETWEEN 52137 AND 52150
    GROUP BY Reciept_Id, Customer_ID,Staff_ID
)
SELECT Reciept_Id,Customer_ID,Staff_ID FROM CTE WHERE (RowNumber % 2 = 0)
```

Results of the above query yeilded:

Reciept_Id	Customer_Id	Staff_Id
52137	C59	S2
52138	C30	S19
52139	C31	S20
52140	C52	S10
52141	C42	S7
52142	C47	S6
52143	C8	S13
52144	C50	S4
52145	C40	S15
52146	C38	S5
52147	C9	S19
52148	C43	S16
52149	C45	S11
52150	C57	S7

Whereas the original result without a modulo comparison on the row would have yeilded a much different result, the raw table supplied in the appendix

With this known, and additional section was added to the python script to generate update statements that would be easy to add to the current migrations.sql script we were prototyping.
The generated update statements appeared as:

```
-- Auto-generated query to fix error of type: Staff.Id Mismatch
-- Resolved error identified by UUID: dcf16fba08c63ecc85556c385204d9524ec359cf
UPDATE Assignment1Data
SET Reciept_Id=(
SELECT MAX(Reciept_Id)+1
FROM Assignment1Data)
WHERE Reciept_Id=52136
AND Customer_Id = 'C13' AND Staff_Id = 'S4'
GO
```

Determining now potential entries that broke further rules was our next objective. We pursued the idea that entries of receipts could potentially have duplicate items recorded against the ReceiptItem table. A simple script was generated to check our assumptions of this:

```
-- Verify that no receipt has duplicate ItemIds and all are unique per order
SELECT *
FROM
(
    SELECT [ReceiptItem].[ReceiptId],
    COUNT([ReceiptItem].[ReceiptId]) AS 'ItemCount',
    COUNT(DISTINCT [ReceiptItem].[ItemId]) AS 'ItemIdCount'
    FROM [ReceiptItem]
    GROUP BY [ReceiptItem].[ReceiptId]) AS SubQuery
WHERE [SubQuery].[ItemIdCount] != [SubQuery].[ItemCount]
ORDER BY [SubQuery].[ReceiptId]
```

This query returned a result of 912 rows out of the total 2514, which we believed was a large amount given the issues identified earlier numbered in only the teens, however on manual inspection of a number of the reported issue records, it was apparent this figure was actually correct.

Given the large task associated with the entries, an additional module was written for generation of SQL in python which resulted in two queries for each dulicate item entry per receipt, the first query updating the total of one of the records to reflect the real item quantity, the later dropping the non-altered entry after the first had been completed.

The script was as follows:

```
-- Auto-generated query to fix error of type: Item.Id Duplicate
-- Resolved error identified by UUID: 0ee74976129cce87fb1558eb5586b1511f5c8d8f
UPDATE Assignment1Data
SET [Item_Quantity]=(
SELECT SUM([Item_Quantity])
FROM Assignment1Data
WHERE Reciept_Id=51500
AND Item_ID = 20)
WHERE Reciept_Id=51500
AND Item_ID = 20
AND Item_Quantity = 1
GO

-- Auto-generated query to fix error of type: Item.Id Duplicate
-- Resolved error identified by UUID: 0ee74976129cce87fb1558eb5586b1511f5c8d8f
DELETE FROM Assignment1Data
WHERE Reciept_Id=51500
AND Item_ID = 20
AND Item_Quantity < 1
GO
```

3.2 Assumptions and Reasoning

3.2.1 Item Table

An assumption of the ItemId never needing to be larger than a smallint was followed, as a basic query into the maximum range within the test data suggested that the maximum Id that currently existed was 30:

```
-- Some basic queries for us to determine potential outlier data:
-- What is the max of each column where datatype is int?
SELECT MAX(Item_ID) AS 'Max Item_ID'
FROM Assignment1Data;
```

With the results:

```
Max Item_ID
30
```

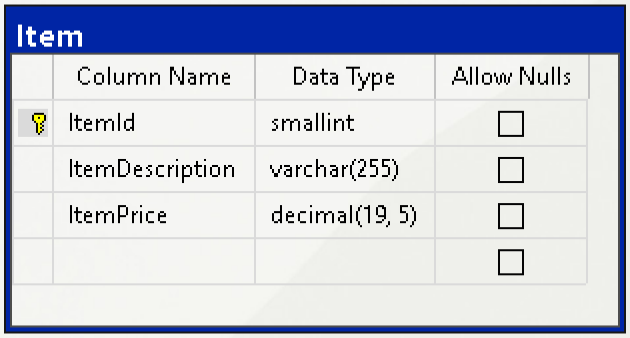
ItemDescription underwent some size optimisation, as the max datalength that currently existed within the supplied data was 52, and we are to assume that into the future more items may be added, a value of 255 should allow for a varied range of descriptions.
SQL queried to determine to above assumption:

```
-- Determine current max varchar used in Item_Description
SELECT MAX(DATALENGTH(Item_Description))
FROM Assignment1Data;
```

We do recognise the requirements for optimisation may not require such measures, and acknowledge that a varchar(max)/text datatype would also be reasonable.

ItemPrice while imported as float type was considered too precise for the usecase of a monetary value. While MONEY and derivatives exist in the TSQL ecosphere, there are real concerns of accuracy of the datatype [1], and therefore we decided for a decimal(19,5) typing [2].

The final Item table structure is reflected as:



Item

	Column Name	Data Type	Allow Nulls
🔑	ItemId	smallint	☐
	ItemDescription	varchar(255)	☐
	ItemPrice	decimal(19, 5)	☐
			☐

4 Base Analysis

4.1 Raw Results

5 Executive Summary

6 Assumptions

References

[1] Reasons against TSQL Money type: Stackoverflow User; *SQLMenace* https://stackoverflow.com/questions/582797/should-you-choose-the-money-or-decimalx-y-datatypes-in-sql-server

[2] Microsoft TSQL documentation of Decimal/Numeric types https://docs.microsoft.com/en-us/sql/t-sql/data-types/decimal-and-numeric-transact-sql?view=sql-server-2017

[3] Microsoft documentation: WITH common_table_expression (Transact-SQL) https://docs.microsoft.com/en-us/sql/t-sql/queries/with-common-table-expression-transact-sql?view=sql-server-2017

7 Appendix

7.1 CTE Raw Results

Reciept_Id	Customer_Id	Staff_Id
52137	C27	S4
52137	C59	S2
52138	C29	S13
52138	C30	S19
52139	C3	S5
52139	C31	S20
52140	C38	S4
52140	C52	S10
52141	C24	S19
52141	C42	S7
52142	C46	S8
52142	C47	S6
52143	C51	S17
52143	C8	S13
52144	C11	S10
52144	C50	S4
52145	C21	S8
52145	C40	S15
52146	C38	S16
52146	C38	S5
52147	C40	S18
52147	C9	S19
52148	C26	S8
52148	C43	S16
52149	C10	S19
52149	C45	S11
52150	C15	S10
52150	C57	S7

7.2 Python Script

Listing 1: Main.py

```
#!/usr/bin/env python3.7
import classes as Classes
import os
import sys
import csv
import openpyxl
import traceback

# Function to parse all receipts once populated and add to employee totals
def populate_receipt_totals(sales,employees,customers,items):
    for receipt_id,sale in sales.sales.items():
        total = 0
        for item_id,item in sale.receipt.items.items():
            total = total + (item.quantity * item.price)
            employees.employees[sale.receipt.staff.id].item_count += item.quantity

        if(len(sale.receipt.items.items()) > 4):
            print("Total was adjusted from {} to {} due to business rules related to 
                number\nof items in a sale.".format(total, total * 0.85))
            total *= 0.85
            employees.employees[sale.receipt.staff.id].discounted_sales += 1

        print("Total calculated for receipt {} is: {}, Items count was: {}".format(
            receipt_id, total, len(sale.receipt.items.items())))
        employees.employees[sale.receipt.staff.id].sales_count += 1
        employees.employees[sale.receipt.staff.id].sales_total += total

    populate_customer_totals(sales,customers)
    populate_item_totals(sales,items)
    generate_employee_report(employees)

# Function to parse all receipts once populated and add to customer totals
def populate_item_totals(sales,items):
    for receipt_id,sale in sales.sales.items():
        total = 0
        for item_id,item in sale.receipt.items.items():
            total = total + (item.quantity * item.price)
            items.items[sale.receipt.items[item_id].id].item_count += item.quantity

        if(len(sale.receipt.items.items()) > 4):
            total *= 0.85
            items.items[sale.receipt.items[item_id].id].discounted_sales += 1

        items.items[sale.receipt.items[item_id].id].sales_count += 1
        items.items[sale.receipt.items[item_id].id].sales_total += total

    generate_items_report(items)

# Function to parse all receipts once populated and add to customer totals
def populate_customer_totals(sales,customers):
    for receipt_id,sale in sales.sales.items():
        total = 0
        for item_id,item in sale.receipt.items.items():
            total = total + (item.quantity * item.price)
            customers.customers[sale.receipt.customer.id].item_count += item.quantity

        if(len(sale.receipt.items.items()) > 4):
            total *= 0.85
            customers.customers[sale.receipt.customer.id].discounted_sales += 1

        customers.customers[sale.receipt.customer.id].sales_count += 1
        customers.customers[sale.receipt.customer.id].sales_total += total

    generate_customer_report(customers)

# Generation of required output files
def generate_results_structures():
    try:
        if not os.path.exists('Results'):
            os.makedirs('Results')

        open('Results/Employee_Results.txt', 'w+').close()
        open('Results/Item_Results.txt', 'w+').close()
        open('Results/Customer_Results.txt', 'w+').close()

    except Exception:
        print("An error occurred: {}".format(traceback.format_exc()))

# Main branch of code to parse rows in excel file
def parse_rows(rows,logged_errors):
    for row in rows:
        receipt_id = row[1].value
        if receipt_id in sales.sales:
            staff_id = row[5].value
            customer_id = row[2].value
            item_id = row[11].value
            item_quantity = row[13].value
            for item in sales.sales[receipt_id].receipt.items.items():
                if item_id == item[0]:
                    print("Error in data row; {} is the same as {}".format(item_id,
                        item_id))
                    logged_errors.add_error(receipt_id,"Error in data row id: {}; {} is
                        the same as {}".format(receipt_id,item_id,item_id),"Item.Id
                        Duplicate",customer_id,staff_id,item_id,item_quantity,sales.sales[
                        receipt_id].receipt.items[item_id].quantity)

            if sales.sales[receipt_id].receipt.staff.id != staff_id:
                print("Error in data row; {} is not the same as {}".format(sales.sales[
                    receipt_id].receipt.staff.id, staff_id))
                logged_errors.add_error(receipt_id,"Error in data row id: {}; {} is not
                    the same as {}".format(receipt_id, sales.sales[receipt_id].receipt.
                    staff.id, staff_id),"Staff.Id Mismatch",customer_id,staff_id,item_id,
                    item_quantity,None)

            if sales.sales[receipt_id].receipt.customer.id != customer_id:
                print("Error in data row; {} is not the same as {}".format(sales.sales[
                    receipt_id].receipt.customer.id, customer_id))
                logged_errors.add_error(receipt_id,"Error in data row id: {}; {} is not
                    the same as {}".format(receipt_id, sales.sales[receipt_id].receipt.
                    customer.id, customer_id),"Customer.Id Mismatch",customer_id,staff_id,
                    item_id,item_quantity,None)

            print("Found existing receipt {}, adding items instead".format(receipt_id))
            sales.add_items_to_sale(row,receipt_id)
        else:
            sales.parse_row(row,employees,customers,items)

# Clear the current errors.txt file
def clear_error_log():
    open('Results/Errors.txt','w+').close()
    open('Results/SQL.txt','w+').close()

# Function to generate employee report and output to disk
def generate_employee_report(employees):
    employee_output = ""
    header = "Results for Employee analysis:"
    for employee_id,employee in employees.employees.items():
        employee_output += """Employee: {}, {} {} \n
        Metrics: ###########################
        Sales Count = {}
        Total Discounted Sales: {}
        Discounted Sales Ratio: {}
        Total Items Sold: {}\n
        Financials: ########################
        Sales Total = ${}
        Average Sale Value: ${}
        Average Item Sold Value: ${}
        \n""".format(
            employee_id,
            employee.first_name,
            employee.surname,
            employee.sales_count,
            employee.discounted_sales,
            employee.discounted_sales / employee.sales_count,
            employee.item_count,
            employee.sales_total,
            employee.sales_total / employee.sales_count,
            employee.sales_total / employee.item_count)
    write_report_results('Employee_Results',header,employee_output)

# Function to generate customer report and output to disk
def generate_customer_report(customers):
    customer_output = ""
    header = "Results for Customer analysis:"
    for customer_id,customer in customers.customers.items():
        customer_output += """Customer: {}, {} {} \n
        Metrics: ###########################
        Sales Count = {}
        Total Discounted Sales: {}
        Discounted Sales Ratio: {}
        Total Items Sold: {}\n
        Financials: ########################
        Sales Total = ${}
        Average Sale Value: ${}
        Average Item Sold Value: ${}
        \n""".format(
            customer_id,
            customer.first_name,
            customer.surname,
            customer.sales_count,
            customer.discounted_sales,
            customer.discounted_sales / customer.sales_count,
            customer.item_count,
            customer.sales_total,
            customer.sales_total / customer.sales_count,
            customer.sales_total / customer.item_count)
    write_report_results('Customer_Results',header,customer_output)

# Function to generate item report and output to disk
def generate_items_report(items):
    items_output = ""
    header = "Results for Item analysis:"
    for item_id,item in items.items.items():
        items_output += """Item: {}\n
        Metrics: ###########################
        Sales Count = {}
        Total Discounted Sales: {}
        Discounted Sales Ratio: {}
        Total Items Sold: {}\n
        Financials: ########################
        Sales Total = ${}
        Average Sale Value: ${}
        Average Item Sold Value: ${}
        \n""".format(
            item_id,
            item.sales_count,
            item.discounted_sales,
            item.discounted_sales / item.sales_count,
            item.item_count,
            item.sales_total,
            item.sales_total / item.sales_count,
            item.sales_total / item.item_count)
    write_report_results('Item_Results',header,items_output)

# Function to generate error report and output to disk
def generate_error_report(logged_errors):
    header = "Error Report:"
    error_output = ""
    for error_log_id,error_log in logged_errors.logged_errors.items():
        error_output += "ErrorId = {}\nErrorType = {}\nReceiptId = {}\nError = {}\n\n""".
            format(
            error_log_id,
            error_log.error_type,
            error_log.receipt_id,
            error_log.trace)
    write_report_results('Errors',header,error_output)

def generate_sql_move_items(logged_errors):
    header = "USE EBUS3030;"
    sql_output = ""
    parsed_receipt_ids = []
    for error_log_id,error_log in logged_errors.logged_errors.items():
        if error_log.receipt_id not in parsed_receipt_ids and error_log.error_type != "
            Item.Id Duplicate":
            sql_output += """
-- Auto-generated query to fix error of type: {}
-- Resolved error identified by UUID: {}
UPDATE Assignment1Data
SET Reciept_Id=(
SELECT MAX(Reciept_Id)+1
FROM Assignment1Data)
WHERE Reciept_Id={}
AND """.format(error_log.error_type,error_log_id,error_log.receipt_id)
            if error_log.customer_id is not None and error_log.staff_id is not None:
                sql_output += "Customer_Id = '{}' AND Staff_Id = '{}'\nGO\n".format(
                    error_log.customer_id,error_log.staff_id)
            elif error_log.customer_id is not None:
                sql_output += "Customer_Id = '{}'\nGO\n".format(error_log.customer_id)
            elif error_log.staff_id is not None:
                sql_output += "Staff_Id = '{}'\nGO\n".format(error_log.staff_id)
            else:
                sql_output = None
            parsed_receipt_ids.append(error_log.receipt_id)

    write_report_results('SQL',header,sql_output)

def generate_sql_fix_duplicate_items(logged_errors):
    header = "USE EBUS3030;"
    sql_output = ""
    for error_log_id,error_log in logged_errors.logged_errors.items():
        print(error_log.error_type)
        if error_log.error_type == "Item.Id Duplicate":
            sql_output += """
-- Auto-generated query to fix error of type: {}
-- Resolved error identified by UUID: {}
UPDATE Assignment1Data
SET [Item_Quantity]=(
SELECT SUM([Item_Quantity])
FROM Assignment1Data
WHERE Reciept_Id={}
AND Item_ID = {})
WHERE Reciept_Id={}
AND Item_ID = {}
AND Item_Quantity = {}\nGO\n""".format(error_log.error_type,
                                error_log_id,
                                error_log.receipt_id,
                                error_log.item_id,
                                error_log.receipt_id,
                                error_log.item_id,
                                error_log.item_quantity)

            sql_output += """
-- Auto-generated query to fix error of type: {}
-- Resolved error identified by UUID: {}
DELETE FROM Assignment1Data
WHERE Reciept_Id={}
AND Item_ID = {}
AND Item_Quantity < {}\nGO\n""".format(error_log.error_type,
                                error_log_id,
                                error_log.receipt_id,
                                error_log.item_id,
                                error_log.item_quantity)

    write_report_results('SQL',header,sql_output)


# Generalised function to write a report to disk
def write_report_results(report_name,header,report_body):
    with open('Results/{}.txt'.format(report_name),'a+') as report:
        report.write(header + 2*'\n')
        report.write(report_body)

# Main hook
if __name__ == '__main__':
    # Open excel file stored in child folder
    excel_file = openpyxl.load_workbook('Data/Assignment1Data.xlsx')
    data = excel_file['Asgn1_Data']
    sales = Classes.Sales()
    employees = Classes.Employees()
    customers = Classes.Customers()
    items = Classes.Items()
    logged_errors = Classes.LoggedErrors()
    clear_error_log()

    # Main branch of code to parse excel file.
    parse_rows(data.rows,logged_errors)

    # If results folder and required text files don't exist, create them
    generate_results_structures()

    # Output error report to disk.
    generate_error_report(logged_errors)

    # Output sql to disk to fix errors found
    generate_sql_move_items(logged_errors)
    generate_sql_fix_duplicate_items(logged_errors)

    # Iterate over sales and employees to generate reports
    # populate_receipt_totals(sales,employees,customers,items)
```

Listing 2: Classes.py

```
#!/usr/bin/env python3.7
import hashlib

# Item class, to imitate item entries in receipt
class Item:
    def __init__(self,item_id,item_description,item_price,item_quantity):
        self.id = item_id
        self.description = item_description
        self.price = item_price
        self.quantity = item_quantity
        self.sales_count = 0
        self.sales_total = 0
        self.item_count = 0
        self.discounted_sales = 0

# Office class, to imitate office entries in staff
class Office:
    def __init__(self,office_id,office_location):
        self.id = office_id
        self.location = office_location

# Staff class, to emulate staff
class Staff:
    def __init__(self,staff_id,staff_first_name,staff_surname,office):
        self.id = staff_id
        self.first_name = staff_first_name
        self.surname = staff_surname
        self.office = office
        self.sales_count = 0
        self.sales_total = 0
        self.item_count = 0
        self.discounted_sales = 0

# Customer class to emulate customers
class Customer:
    def __init__(self,customer_id,customer_first_name,customer_surname):
        self.id = customer_id
        self.first_name = customer_first_name
        self.surname = customer_surname
        self.sales_count = 0
        self.sales_total = 0
        self.item_count = 0
        self.discounted_sales = 0

# Receipt class to hold data for a sale
class Receipt:
    def __init__(self,receipt_id,customer,staff):
        self.id = receipt_id
        self.customer = customer
        self.staff = staff
        self.items = {}
        self.item_count = 0
        self.total = 0

    # Function to add items to receipt
    def add_item(self,item):
        self.items[item.id] = item
        self.item_count += 1

# Sale class to hold one receipt (Kinda redundant)
```

```
class Sale:
    def __init__(self,date,receipt):
        self.date = date
        self.receipt = receipt

# Sales class to hold record of all sales
class Sales:
    def __init__(self):
        self.sales = {}

    # Parse row function, intended to determine if row is a header row or contains
        formula
    def parse_row(self,row,employees,customers,items):
        if row[0].value != 'Sale_Date' and isinstance(row[1].value,int):
            item = Item(row[11].value,row[12].value,row[14].value,row[13].value)
            customer = Customer(row[2].value,row[3].value,row[4].value)
            office = Office(row[8].value,row[9].value)
            staff = Staff(row[5].value,row[6].value,row[7].value,office)
            receipt = Receipt(row[1].value,customer,staff)
            receipt.add_item(item)
            sale = Sale(row[0].value,receipt)
            self.sales[sale.receipt.id] = sale
            print("Added sale: {}".format(sale.receipt.id))

            if staff.id in employees.employees.items():
                print("Duplicate employee: {}".format(staff.id))
            else:
                employees.add_employee(staff.id,staff)

            if customer.id in customers.customers.items():
                print("Duplicate customer: {}".format(customer.id))
            else:
                customers.add_customer(customer.id,customer)

            # We itemed your items so you can .items() your items
            if item.id in items.items.items():
                print("Duplicate item: {}".format(item.id))
            else:
                items.add_item(item.id,item)

        else:
            print("Skipped row, either it was a row header: {} or it was a formula: {}".
                format(row[0].value,row[1].value))

    # Add items to sale if the receipt already exists
    def add_items_to_sale(self,row,existing_sale_identifier):
        item = Item(row[11].value,row[12].value,row[14].value,row[13].value)
        self.sales[existing_sale_identifier].receipt.add_item(item)
        print("Added items to receipt {} : ID: {}, Desc: {}, Price: {}, Quantity: {}".
            format(existing_sale_identifier,item.id,item.description,item.price,item.
            quantity))

# Employees class to hold all staff
class Employees:
    def __init__(self):
        self.employees = {}

    # Function to add new employees if they currently don't exist
    def add_employee(self,employee_id,employee):
        self.employees[employee_id] = employee
```

```
# Customers class to hold all customers
class Customers:
    def __init__(self):
        self.customers = {}

    # Function to add new customer if they currently don't exist
    def add_customer(self,customer_id,customer):
        self.customers[customer_id] = customer

# Items class to hold all items
class Items:
    def __init__(self):
        self.items = {}

    # Function to add new items if they currently don't exist
    def add_item(self,item_id,item):
        self.items[item_id] = item

class Error_Log:
    def __init__(self,trace,error_type,receipt_id,customer_id = None,
                staff_id = None,item_id = None,item_quantity = None,
                    duplicate_item_quantity = None):
        self.trace = trace
        self.receipt_id = receipt_id
        self.error_type = error_type
        self.customer_id = None
        self.staff_id = None
        if customer_id is not None:
            self.customer_id = customer_id
        if staff_id is not None:
            self.staff_id = staff_id
        if item_id is not None:
            self.item_id = item_id
        if item_quantity is not None:
            self.item_quantity = item_quantity
        if duplicate_item_quantity is not None:
            self.duplicate_item_quantity = duplicate_item_quantity
        self.hash = self.generate_hash(trace + error_type + str(receipt_id))

    def generate_hash(self,hashcontent):
        return str(hashlib.sha1(hashcontent.encode(encoding='UTF-8',errors='strict')).
            hexdigest())

# Logged errors class to avoid logging the same error multiple times
class LoggedErrors:
    def __init__(self):
        self.logged_errors = {}
        self.error_count = 0

    # Determine if error related to receipt is already logged
    def add_error(self,receipt_id,trace,error_type,customer_id = None,staff_id = None,
        item_id = None,item_quantity = None,duplicate_item_quantity = None):
        error_log = Error_Log(trace,error_type,receipt_id,customer_id,staff_id,item_id,
            item_quantity,duplicate_item_quantity)
        if error_log.hash not in self.logged_errors:
            self.logged_errors[error_log.hash] = error_log
```