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 2, 2018

Contents

1 Assignment Overview & Requirements

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.

1.1 Datamart Business Rules

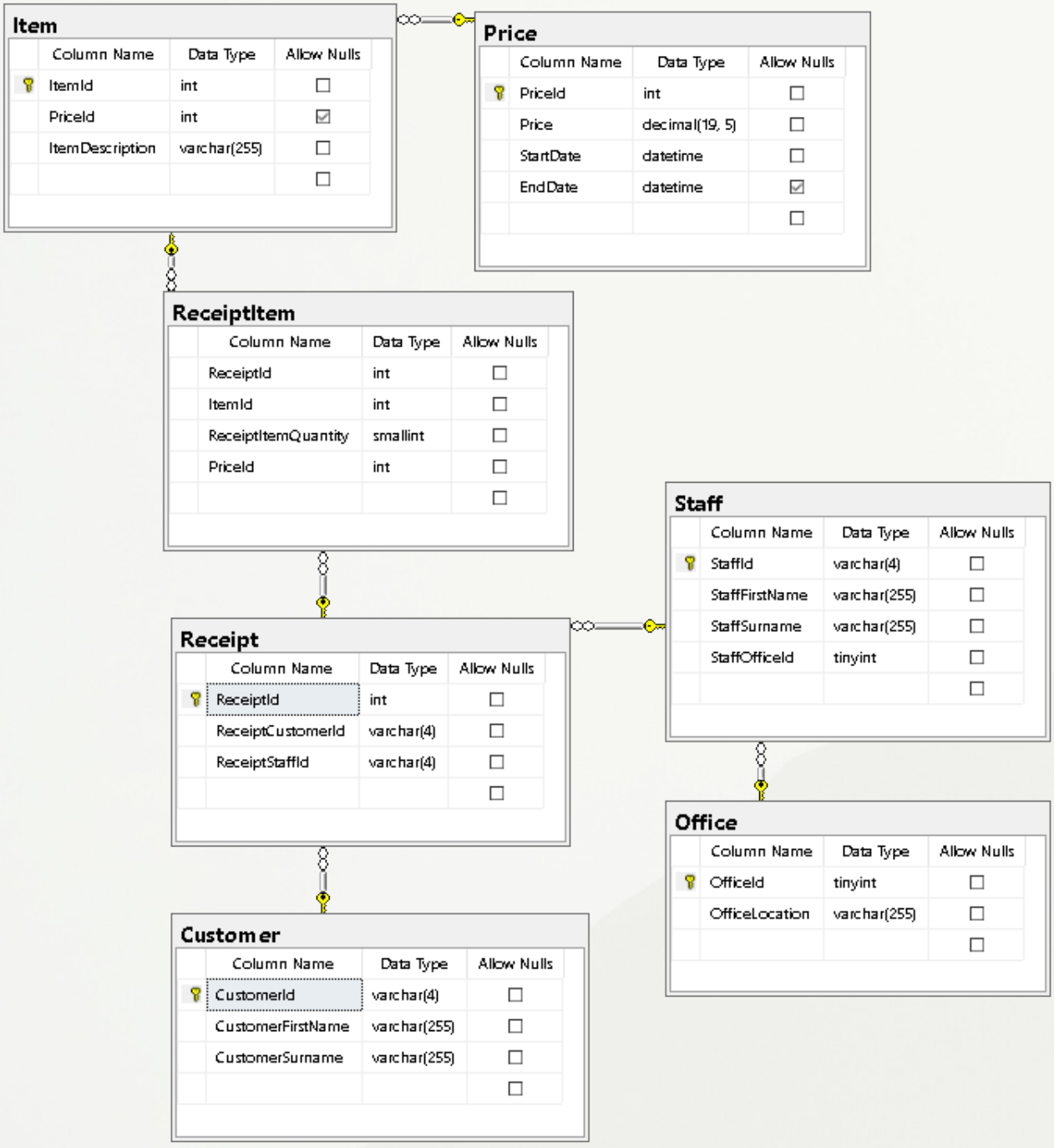
The following business rules were provided to be used in the context of this assignment:

* 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.

With these considerations in mind, the following report was created to outline the discovery, creation and polish to satisfy the assignment requirements.

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



It must be noted that the structure of this data model is less than efficient, and it would be expected in a datamart situation 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 data sets would this model prove a bad design. In such cases a model more representative of the snowflake or star schema 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 analyze 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 initially 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.

After clarifying the issues with the aforementioned cells with Peter, a data file without the offending formula was supplied and used for the remainder of the assignment.

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:

```
1 INSERT INTO Receipt(ReceiptId, ReceiptCustomerId,ReceiptStaffId)
2 SELECT DISTINCT(Reciept_Id),Customer_ID,Staff_ID
3 FROM Assignment1Data
4 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:

```
1 SELECT * FROM Assignment1Data
2 WHERE Customer_ID='C32'
3 AND Staff_ID='S15'
4 AND Sale_Date='2017-11-12 00:00:00.0000000';
5
6 SELECT * FROM Assignment1Data
7 WHERE Customer_ID='C13'
8 AND Staff_ID='S4'
9 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 occurred:

```
1  UPDATE Assignment1Data
2  SET Reciept_Id=51585,
3  Reciept_Transaction_Row_ID=(
4      SELECT MAX(Reciept_Transaction_Row_ID)+1
5      FROM Assignment1Data
6      WHERE Reciept_Id=51585)
7  WHERE Customer_ID='C32'
8  AND Staff_ID='S15'
9  AND Sale_Date='2017-11-12 00:00:00.0000000'
10 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:

```
1 UPDATE Assignment1Data
2 SET Reciept_Id=(
3     SELECT MAX(Reciept_Id)+1
4     FROM Assignment1Data)
5 WHERE Customer_ID='C27'
6 AND Staff_ID='S4'
7 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 yielded:

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 yielded 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]
GO
```

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 duplicate 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:

```
1  -- Auto-generated query to fix error of type: Item.Id Duplicate
2  -- Resolved error identified by UUID: 0ee74976129cce87fb1558eb5586b1511f5c8d8f
3  UPDATE Assignment1Data
4  SET [Item_Quantity]=(
5  SELECT SUM([Item_Quantity])
6  FROM Assignment1Data
7  WHERE Reciept_Id=51500
8  AND Item_ID = 20)
9  WHERE Reciept_Id=51500
10 AND Item_ID = 20
11 AND Item_Quantity = 1
12 GO
13
14 -- Auto-generated query to fix error of type: Item.Id Duplicate
15 -- Resolved error identified by UUID: 0ee74976129cce87fb1558eb5586b1511f5c8d8f
16 DELETE FROM Assignment1Data
17 WHERE Reciept_Id=51500
18 AND Item_ID = 20
19 AND Item_Quantity < 1
20 GO
```

Having now cleaned what we believed to be all discrepancies, we could finally start to look at evaluating data, our analysis outlined in base analysis

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:

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

With the results:

```
Max Item_ID
30
```

ItemDescription underwent some size optimisation, as the max data length 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:

```
1 -- Determine current max varchar used in Item_Description
2 SELECT MAX(DATALENGTH(Item_Description))
3 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.

3.2.2 Price Table

The price table was designed to hold historical data as required by the business rules, an effective range can be used here to determine item pricing for time frames, current items having no end date or an end date as some point in time into the future.

Accuracy on the pricing was important, we decided to use a decimal(19,5) structure to ensure no problems should arise at any point with calculation of totals. [1]

3.2.3 ReceiptItem

The receipt item table acts as a line-item style associative entity, the quantity and historical priceId used at time of transaction can allow an item's price to be updated and still maintain historical pricing associated with the receipt.

3.2.4 Receipt

The receipt table acts as a meta-table in this instance, other tables associate with this table with the receiptId field. Due to this it made it extremely easy to use a number of joins/inner joins to determine some of the metrics outlined in the base analysis.

3.2.5 Staff

Staff was left in a non-normalised state to ensure efficiency of queries into the future, normalising the table further would yield little value to the business based on the requirements. The office table is referenced by the staff table. This is merely to satisfy the assumption that, while the only office to exist was Newcastle in this setting, the requirement of more offices into the future is a possibility and the required join would be little impact on speed of queries in a datamart.

3.2.6 Customer

Customer, just like staff could be normalised further requiring more joins and potentially causing a performance issue into the future, for simplicity we kept only the supplied data in mind, and assumed no more data would be required by the datamart into the future.

4 Base Analysis

4.1 Raw Results

A number of metrics were considered to satisfy the request related to the best salesperson, as we are not certain if this is determined by a specific metric or a set of metrics we included a number of analyzed points for the project:

- Total receipts attributed to a staff member
- Total items sold by a staff member
- Ratio of discounted sales to normal sales for each staff member
- Total sale value per staff member
- Average sale value per staff member
- Average item value per staff member

4.1.1 Total Number of Sales

The total number of sales per staff member were considered with the following sql query:

```sql
-- Sales count per staff member (Receipt Count)
SELECT COUNT(*) AS 'Sales Count', s.StaffId,s.StaffFirstName,s.StaffSurname
FROM Receipt r
INNER JOIN ReceiptItem ri ON r.ReceiptId = ri.ReceiptId
INNER JOIN Item i ON i.ItemId = ri.ItemId
INNER JOIN Price p ON p.PriceId = ri.PriceId
INNER JOIN Staff s ON s.StaffId = r.ReceiptStaffId
GROUP BY s.StaffId,s.StaffFirstName,s.StaffSurname
ORDER BY 'Sales Count' DESC;
```

Leading to a range of 736 to 500, the top five staff were:

Sales Count	StaffId	StaffFirstName	StaffSurname
736	S17	Daniel	Baker
709	S19	Kaitlyn	Ortiz
702	S8	Michelle	Miller
701	S1	Lauren	Martin
700	S5	Stephanie	Watson

4.1.2 Total Items Sold

The total items attributed to each staff member were considered also, determined by the query:

```
-- Item count per staff member
SELECT SUM(ri.ReceiptItemQuantity) AS 'Item Count', s.StaffId,s.StaffFirstName,s.
    StaffSurname
FROM Receipt r
INNER JOIN ReceiptItem ri ON r.ReceiptId = ri.ReceiptId
INNER JOIN Staff s ON s.StaffId = r.ReceiptStaffId
GROUP BY s.StaffId,s.StaffFirstName,s.StaffSurname
ORDER BY 'Item Count' DESC;
```

Yeilding a range of 4481 to 2960, with the top five staff members in this analysis:

Item Count	StaffId	StaffFirstName	StaffSurname
4481	S17	Daniel	Baker
4414	S19	Kaitlyn	Ortiz
4373	S1	Lauren	Martin
4333	S8	Michelle	Miller
4308	S5	Stephanie	Watson

4.1.3 Discounted Sales Ratio

Consideration of the number of sales made by each staff member was also made, the following query yeilding the results we required:

```
-- Sales metrics for discounted and standard sales per staff member
SELECT s.StaffId,s.StaffFirstName,s.StaffSurname,
SUM(SubQuery.[Discounted Sales]) AS 'Discounted Sales',
SUM(SubQuery.[Standard Sales]) AS 'Standard Sales'
FROM (
    SELECT CAST(
        CASE
        WHEN COUNT(ri.[ReceiptItemQuantity]) >= 5
            THEN 1
        ELSE 0
        END AS int) AS 'Discounted Sales',
    CAST(
        CASE
        WHEN COUNT(ri.[ReceiptItemQuantity]) >= 5
            THEN 0
        ELSE 1
    END AS int) AS 'Standard Sales',
    r.ReceiptId
    FROM Receipt r
    INNER JOIN ReceiptItem ri ON r.ReceiptId = ri.ReceiptId
    INNER JOIN Item i ON i.ItemId = ri.ItemId
    INNER JOIN Price p ON p.PriceId = ri.PriceId
    GROUP BY r.ReceiptId
) AS SubQuery
INNER JOIN Receipt r ON SubQuery.ReceiptId = r.ReceiptId
INNER JOIN ReceiptItem ri ON r.ReceiptId = ri.ReceiptId
INNER JOIN Staff s ON s.StaffId = r.ReceiptStaffId
GROUP BY s.StaffId,s.StaffFirstName,s.StaffSurname
```

Results from the query yielded:

StaffId	StaffFirstName	StaffSurname	Discounted Sales	Standard Sales	Discounted Sales Rate
S4	Robert	Wood	559	102	84.57%
S20	Dylan	Hall	563	112	83.41%
S1	Lauren	Martin	584	117	83.31%
S16	Jordan	Turner	561	117	82.74%
S14	Noah	Brooks	565	118	82.72%
S17	Daniel	Baker	603	133	81.93%
S13	Molly	Carter	556	128	81.29%
S5	Stephanie	Watson	567	133	81.00%
S15	Bailey	Green	529	126	80.76%
S6	Evan	Hill	562	137	80.40%
S10	Jonathan	Jenkins	478	119	80.07%
S18	Megan	James	540	137	79.76%
S19	Kaitlyn	Ortiz	561	148	79.13%
S7	Molly	Jackson	499	138	78.34%
S9	Mélissa	Garcia	510	148	77.51%
S8	Michelle	Miller	541	161	77.07%
S2	Joseph	Reed	494	153	76.35%
S11	Gavin	Thompson	424	135	75.85%
S12	Leah	Harris	376	124	75.20%
S3	Amber	Hill	435	157	73.48%

5 Executive Summary

This report was created for the head sales executive of BIA Inc, in regard to which sales staff member was considered the best performer based on the given data. As no specific metric was given for the task of determining the best performer an analysis was performed, in order perform this task the data was cleaned, reviewed and queried.
The best sales officer in regards to the analysis of total number of sales was Daniel Baker.
Mr Baker had made the largest number of sales in the 12 months of data that was supplied with a total of 736 total sales counts. From this data it can also be seen that after Mr Baker is Kaitlyn Ortiz with 709 total number of sales (difference of 27).

The best sales officer in regards to the analysis of Total items sold was also Daniel Baker. In the 12 months of data that was supplied it can be seen that in Mr Baker's 736 total sales, he sold 4481 items. This is only a 67 item difference between Mr Baker and second place Kaitlyn Ortiz who had a total of 4414.

After considering theses analytics and reviewing the results it can be seen that Mr Daniel Baker may be considered the best sales officer in terms of Total sales and items sold and should be eligible for the reward that is being discussed.

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

6 Appendix

6.1 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

6.2 Python Script

```
#!/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
```

```
43          items.items[sale.receipt.items[item_id].id].sales_total +=
               total
44
45      generate_items_report(items)
46
47  # Function to parse all receipts once populated and add to customer
       totals
48  def populate_customer_totals(sales,customers):
49      for receipt_id,sale in sales.sales.items():
50          total = 0
51          for item_id,item in sale.receipt.items.items():
52              total = total + (item.quantity * item.price)
53              customers.customers[sale.receipt.customer.id].item_count
                   += item.quantity
54
55          if(len(sale.receipt.items.items()) > 4):
56              total *= 0.85
57              customers.customers[sale.receipt.customer.id].
                   discounted_sales += 1
58
59          customers.customers[sale.receipt.customer.id].sales_count +=
               1
60          customers.customers[sale.receipt.customer.id].sales_total +=
               total
61
62      generate_customer_report(customers)
63
64  # Generation of required output files
65  def generate_results_structures():
66      try:
67          if not os.path.exists('Results'):
68              os.makedirs('Results')
69
70          open('Results/Employee_Results.txt','w+').close()
71          open('Results/Item_Results.txt','w+').close()
72          open('Results/Customer_Results.txt','w+').close()
73
74      except Exception:
75          print("An error occurred: {}".format(traceback.format_exc()))
76
77  # Main branch of code to parse rows in excel file
78  def parse_rows(rows,logged_errors):
79      for row in rows:
80          receipt_id = row[1].value
81          if receipt_id in sales.sales:
82              staff_id = row[5].value
83              customer_id = row[2].value
84              item_id = row[11].value
85              item_quantity = row[13].value
86              for item in sales.sales[receipt_id].receipt.items.items()
                   :
87                  if item_id == item[0]:
88                      print("Error in data row; {} is the same as {}".
                           format(item_id,item_id))
89                      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)
90
91              if sales.sales[receipt_id].receipt.staff.id != staff_id:
92                  print("Error in data row; {} is not the same as {}".
                        format(sales.sales[receipt_id].receipt.staff.id,
                        staff_id))
93                  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)
94
95              if sales.sales[receipt_id].receipt.customer.id !=
                    customer_id:
96                  print("Error in data row; {} is not the same as {}".
                        format(sales.sales[receipt_id].receipt.customer.id
                        , customer_id))
97                  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)
98
99              print("Found existing receipt {}, adding items instead".
                    format(receipt_id))
100             sales.add_items_to_sale(row,receipt_id)
101         else:
102             sales.parse_row(row,employees,customers,items)
103
104 # Clear the current errors.txt file
105 def clear_error_log():
106     open('Results/Errors.txt','w+').close()
107     open('Results/SQL.txt','w+').close()
108
109 # Function to generate employee report and output to disk
110 def generate_employee_report(employees):
111     employee_output = ""
112     header = "Results for Employee analysis:"
113     for employee_id,employee in employees.employees.items():
114         employee_output += """Employee: {}, {} {} \n
115         Metrics: ##########################
116         Sales Count = {}
117         Total Discounted Sales: {}
118         Discounted Sales Ratio: {}
119         Total Items Sold: {}\n
120         Financials: ########################
121         Sales Total = ${}
122         Average Sale Value: ${}
123         Average Item Sold Value: ${}
124         \n""".format(
125             employee_id,
126             employee.first_name,
127             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,
```

```
184             item.discounted_sales / item.sales_count,
185             item.item_count,
186             item.sales_total,
187             item.sales_total / item.sales_count,
188             item.sales_total / item.item_count)
189     write_report_results('Item_Results',header,items_output)
190
191 # Function to generate error report and output to disk
192 def generate_error_report(logged_errors):
193     header = "Error Report:"
194     error_output = ""
195     for error_log_id,error_log in logged_errors.logged_errors.items()
            :
196         error_output += "ErrorId = {}\nErrorType = {}\nReceiptId =
               {}\nError = {}\n\n""".format(
197             error_log_id,
198             error_log.error_type,
199             error_log.receipt_id,
200             error_log.trace)
201     write_report_results('Errors',header,error_output)
202
203 def generate_sql_move_items(logged_errors):
204     header = "USE EBUS3030;"
205     sql_output = ""
206     parsed_receipt_ids = []
207     for error_log_id,error_log in logged_errors.logged_errors.items()
            :
208         if error_log.receipt_id not in parsed_receipt_ids and
               error_log.error_type != "Item.Id Duplicate":
209             sql_output += """
210 -- Auto-generated query to fix error of type: {}
211 -- Resolved error identified by UUID: {}
212 UPDATE Assignment1Data
213 SET Reciept_Id=(
214 SELECT MAX(Reciept_Id)+1
215 FROM Assignment1Data)
216 WHERE Reciept_Id={}
217 AND """.format(error_log.error_type,error_log_id,error_log.receipt_id
        )
218             if error_log.customer_id is not None and error_log.
                   staff_id is not None:
219                 sql_output += "Customer_Id = '{}' AND Staff_Id =
                       '{}'\nGO\n".format(error_log.customer_id,error_log
                       .staff_id)
220             elif error_log.customer_id is not None:
221                 sql_output += "Customer_Id = '{}'\nGO\n".format(
                       error_log.customer_id)
222             elif error_log.staff_id is not None:
223                 sql_output += "Staff_Id = '{}'\nGO\n".format(
                       error_log.staff_id)
224             else:
225                 sql_output = None
226             parsed_receipt_ids.append(error_log.receipt_id)
227
228     write_report_results('SQL',header,sql_output)
229
```

```
230 def generate_sql_fix_duplicate_items(logged_errors):
231     header = "USE EBUS3030;"
232     sql_output = ""
233     for error_log_id,error_log in logged_errors.logged_errors.items()
        :
234         print(error_log.error_type)
235         if error_log.error_type == "Item.Id Duplicate":
236             sql_output += """
237 -- Auto-generated query to fix error of type: {}
238 -- Resolved error identified by UUID: {}
239 UPDATE Assignment1Data
240 SET [Item_Quantity]=(
241 SELECT SUM([Item_Quantity])
242 FROM Assignment1Data
243 WHERE Reciept_Id={}
244 AND Item_ID = {})
245 WHERE Reciept_Id={}
246 AND Item_ID = {}
247 AND Item_Quantity = {}\nGO\n""".format(error_log.error_type,
248                                       error_log_id,
249                                       error_log.receipt_id,
250                                       error_log.item_id,
251                                       error_log.receipt_id,
252                                       error_log.item_id,
253                                       error_log.item_quantity)
254
255             sql_output += """
256 -- Auto-generated query to fix error of type: {}
257 -- Resolved error identified by UUID: {}
258 DELETE FROM Assignment1Data
259 WHERE Reciept_Id={}
260 AND Item_ID = {}
261 AND Item_Quantity < {}\nGO\n""".format(error_log.error_type,
262                                       error_log_id,
263                                       error_log.receipt_id,
264                                       error_log.item_id,
265                                       error_log.item_quantity)
266
267     write_report_results('SQL',header,sql_output)
268
269
270 # Generalised function to write a report to disk
271 def write_report_results(report_name,header,report_body):
272     with open('Results/{}.txt'.format(report_name),'a+') as report:
273         report.write(header + 2*'\n')
274         report.write(report_body)
275
276 # Main hook
277 if __name__ == '__main__':
278     # Open excel file stored in child folder
279     excel_file = openpyxl.load_workbook('Data/Assignment1Data.xlsx')
280     data = excel_file['Asgn1 Data']
281     sales = Classes.Sales()
282     employees = Classes.Employees()
283     customers = Classes.Customers()
284     items = Classes.Items()
```

```
285     logged_errors = Classes.LoggedErrors()
286 
287     clear_error_log()
288 
289     # Main branch of code to parse excel file.
290     parse_rows(data.rows,logged_errors)
291 
292     # If results folder and required text files don't exist, create
            them
293     generate_results_structures()
294 
295     # Output error report to disk.
296     generate_error_report(logged_errors)
297 
298     # Output sql to disk to fix errors found
299     generate_sql_move_items(logged_errors)
300     generate_sql_fix_duplicate_items(logged_errors)
301 
302     # Iterate over sales and employees to generate reports
303     # populate_receipt_totals(sales,employees,customers,items)
```

```
#!/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
```

```
53          self.total = 0
54
55      # Function to add items to receipt
56      def add_item(self,item):
57          self.items[item.id] = item
58          self.item_count += 1
59
60  # Sale class to hold one receipt (Kinda redundant)
61  class Sale:
62      def __init__(self,date,receipt):
63          self.date = date
64          self.receipt = receipt
65
66  # Sales class to hold record of all sales
67  class Sales:
68      def __init__(self):
69          self.sales = {}
70
71      # Parse row function, intended to determine if row is a header
            row or contains formula
72      def parse_row(self,row,employees,customers,items):
73          if row[0].value != 'Sale Date' and isinstance(row[1].value,
                int):
74              item = Item(row[11].value,row[12].value,row[14].value,row
                    [13].value)
75              customer = Customer(row[2].value,row[3].value,row[4].
                    value)
76              office = Office(row[8].value,row[9].value)
77              staff = Staff(row[5].value,row[6].value,row[7].value,
                    office)
78              receipt = Receipt(row[1].value,customer,staff)
79              receipt.add_item(item)
80              sale = Sale(row[0].value,receipt)
81              self.sales[sale.receipt.id] = sale
82              print("Added sale: {}".format(sale.receipt.id))
83
84              if staff.id in employees.employees.items():
85                  print("Duplicate employee: {}".format(staff.id))
86              else:
87                  employees.add_employee(staff.id,staff)
88
89              if customer.id in customers.customers.items():
90                  print("Duplicate customer: {}".format(customer.id))
91              else:
92                  customers.add_customer(customer.id,customer)
93
94              # We itemed your items so you can .items() your items
95              if item.id in items.items.items():
96                  print("Duplicate item: {}".format(item.id))
97              else:
98                  items.add_item(item.id,item)
99
100         else:
101             print("Skipped row, either it was a row header: {} or it
                    was a formula: {}".format(row[0].value,row[1].value))
102
```

```
103     # Add items to sale if the receipt already exists
104     def add_items_to_sale(self,row,existing_sale_identifier):
105         item = Item(row[11].value,row[12].value,row[14].value,row
                [13].value)
106         self.sales[existing_sale_identifier].receipt.add_item(item)
107         print("Added items to receipt {} : ID: {}, Desc: {}, Price:
                {}, Quantity: {}".format(existing_sale_identifier,item.id,
                item.description,item.price,item.quantity))
108
109 # Employees class to hold all staff
110 class Employees:
111     def __init__(self):
112         self.employees = {}
113
114     # Function to add new employees if they currently don't exist
115     def add_employee(self,employee_id,employee):
116         self.employees[employee_id] = employee
117
118 # Customers class to hold all customers
119 class Customers:
120     def __init__(self):
121         self.customers = {}
122
123     # Function to add new customer if they currently don't exist
124     def add_customer(self,customer_id,customer):
125         self.customers[customer_id] = customer
126
127 # Items class to hold all items
128 class Items:
129     def __init__(self):
130         self.items = {}
131
132     # Function to add new items if they currently don't exist
133     def add_item(self,item_id,item):
134         self.items[item_id] = item
135
136 class Error_Log:
137     def __init__(self,trace,error_type,receipt_id,customer_id = None,
138                  staff_id = None,item_id = None,item_quantity = None,
                       duplicate_item_quantity = None):
139         self.trace = trace
140         self.receipt_id = receipt_id
141         self.error_type = error_type
142         self.customer_id = None
143         self.staff_id = None
144         if customer_id is not None:
145             self.customer_id = customer_id
146         if staff_id is not None:
147             self.staff_id = staff_id
148         if item_id is not None:
149             self.item_id = item_id
150         if item_quantity is not None:
151             self.item_quantity = item_quantity
152         if duplicate_item_quantity is not None:
153             self.duplicate_item_quantity = duplicate_item_quantity
154         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
```