EBUS3030 Assignment 2

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

October 25, 2018

Contents

1 Assignment Overview & Requirements

Business Intelligence - EBUS3030
Assignment 2

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/ETL to get the data in a usable format, Output of your analysis, 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 CEO.

Assignment Question

The CEO of 'BIA Inc' had been speaking to the Sales Executive and had heard about some recent work you had completed and thought you might be able to assist them with a problem they have.

"I've heard that you helped the Sales Exec recently with understanding more about our Newcastle site. I would now like your help to get a handle on the whole business. As you are aware, the Newcastle office is just 1 of 10 sites we have across the country. Unfortunately, sales in some items have dropped across the country in recent years and we are currently running at a loss.

We need to consider consolidating our company offices. We need to reduce costs for the longevity of the company as a whole. I need you to get some numbers together around the performance of our 10 offices, so that I can factor this information into any decision regarding which office (or offices) we might consider closing.

I would like a summary of recent numbers and some trend analysis as well please. It would be great if you could also project sales for the next 12 months for each office as well. It would be helpful if you could indicate the 3 most popular and 3 least popular items in each of our stores, as well as the worst performing items for the company as a whole.

As you can imagine, this is a very sensitive topic so, as part of your response I want you to provide the justification as to which office we may close. Our decision will upset some people and I want to make sure we have all the background information on hand. If you can provide a Ranking of offices based on your analysis that would be wonderful.

…. I believe you started to bring together a data store of this information from the Newcastle Office, can you expand that and load all of the sales information for all offices and complete your analysis.
"

Assignment Deliverables

Using the data file provided in Excel and notes about the data (*Assignment 2 – data.xlsx*), you are required to complete the following elements as part of the assignment.

- Data Model/Data Load Process
  - Provide an overview of the data model & ETL process completed to get the data ready for analysis
  - Ensure you record any assumptions you have made as part of this component and your reasoning behind the assumption.
- Analysis including any predictive work undertaken
  - Provide the SQL and raw output of your base analysis
  - Provide workings of the predictive work you completed for the trending & prediction on future sales.
  - Ensure you record any assumptions you have made as part of this component and your reasoning behind the assumption (this includes answers to any relevant questions put to management (your lecturer) during workshops.

- Executive Summary & Presentation in response to business question.
    - Provide an Executive report and Dashboard
        - Your Executive report should include a short Executive brief/summary that presents a clear concise response back to the CEO question about possible downsizing of operations including evidence/justification.
        - A dashboard should allow quick comparisons between the sites to be undertaken as well as contain at least one element of 'predictive' analysis
        - Present the material as if it is to be consumed in a formal boardroom meeting
            - All members of the team need to participate in a (15 – 20 minute) presentation to be given as part of the lab in Week 11.
            - Please be aware that any of the company Executive may ask questions as you present your findings.

NB: As part of your response, you should also specifically include any assumptions/external information you have made/used throughout the process as well as any quality processes/checking you have completed or limitations you discovered.

Breakup of assignment Marks (total course mark for assignment = Assignment Part B submission (28% + Presentation Two (7%) = 35%.

Assignment Component	Percentage Allocation
Data Model/ETL	10%
Core Analysis	50%
Executive Summary/Evidence	25%
Dashboard	15%
	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 Two drop folder by 12 noon, Thursday 25th October. You will also be required to submit a paper copy of your report at the beginning of the presentation workshop (make sure this is printed well before the workshop and has a group Assessment Cover sheet signed by ALL team members).

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

Your data model should include a printout of an ER diagram using the notation described in lectures. It should also include a printout of your SQL schema showing Primary and foreign keys, as well as all attributes.

Your presentation is worth 7% of the course mark. It should simulate presenting the report to management. You should time your presentation to be between 12-15 minutes with 5-10 minutes for questions. You presentation should include a demonstration of your dashboard, results and recommendations from your analysis. Your presentation marks will contain components for organisation, comprehension of presented results, and timing.

2 Executive Summary

2.1 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.We only support electronic orders.
- 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.
- After consultation with your team, we have made the following change to discount applied to sales: Where a customer has multiple line items, any sale with 5 or more row items (containing at least 5 different items) is provided a 5% 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.
- Store Item prices can change at any point, however the price the customer pays is the amount listed for the store item that is sold on the sale date. We need to keep a record of all store item prices historically so that we can determine what the store item price was at any particular past date.
- Only one BIA sales assistant can be attributed to any receipt.
- Customers may visit multiple stores for purchases (ie they are not locked to a particular store). As a result, all customer records are replicated across all stores, so they do not need to be re-recorded at a store by store level.

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

3 Data Model

The following section outlines the models used in the design and creation of the database. It includes the EER Model with relations, attributes and relationships as well as the database schema from Microsoft SQL Server Management Studio.

3.1 Database Schema

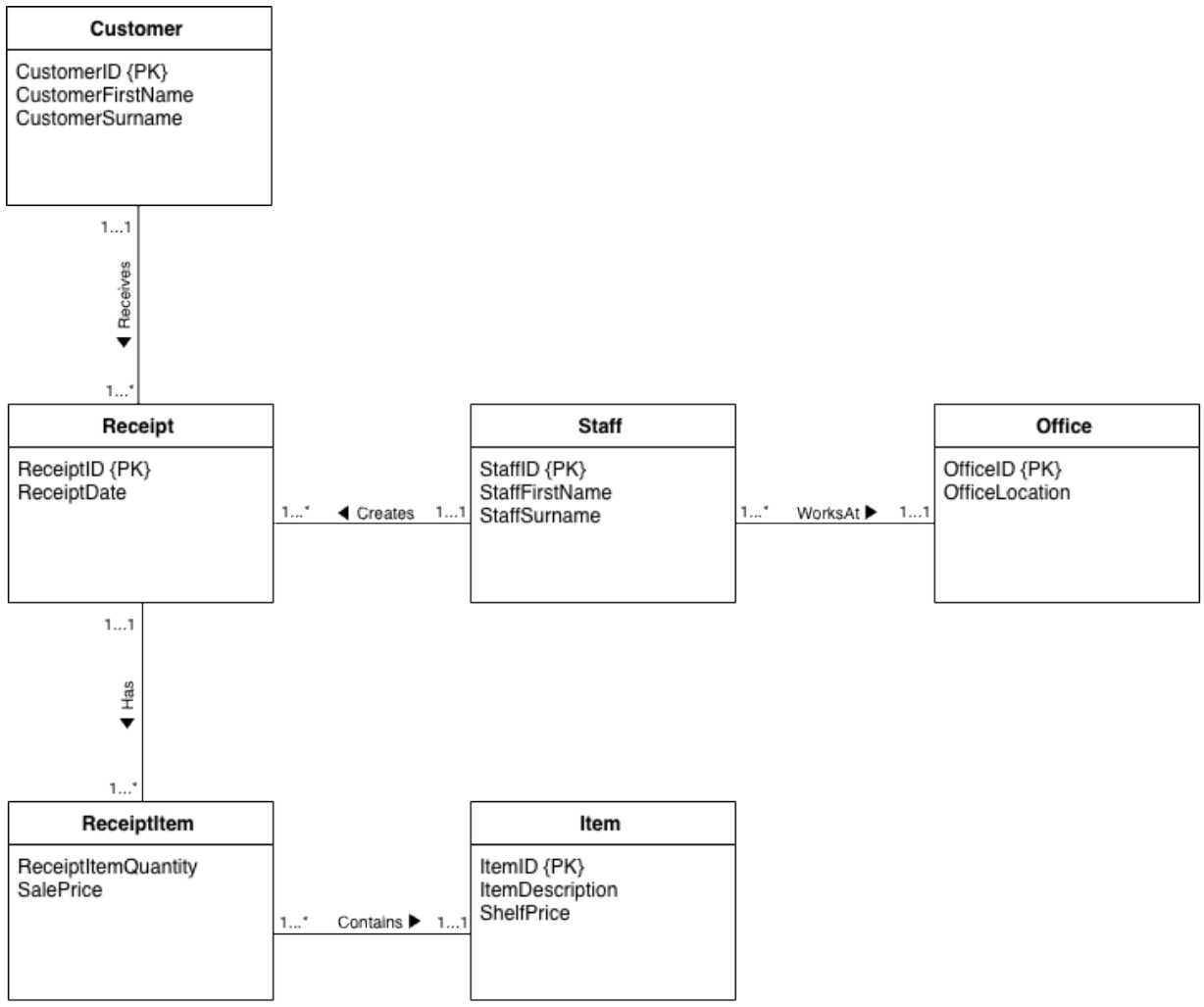
The below data model is only a suggestion and is still subject to change into the future. A full create script can be found in the appendix

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.

3.2 EER Diagram

An EER diagram of the suggested data model is provided below.



4 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:

Assignment2Data

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.

4.1 Quality Assurance Processes

After performing a basic query to determine if, like our previous project, multiple items of the same ID on a single receipt existed it was apparent that the same issue existed in this data set as did in the previous.
To mitigate this, the team developed a simple update to the ReceiptItem was performed, to ensure that a dicount would not be applied to a receipt with less than five items, when potentially the receipt did not actually have more than four distinct items.

The following query was used to determine this information:

```
1  -- Verify that no receipt has duplicate ItemIds and all are unique per order
2  SELECT *
3  FROM
4  (
5      SELECT [ReceiptItem].[ReceiptId],
6      COUNT([ReceiptItem].[ReceiptId]) AS 'ItemCount',
7      COUNT(DISTINCT [ReceiptItem].[ItemId]) AS 'ItemIdCount'
8      FROM [ReceiptItem]
9      GROUP BY [ReceiptItem].[ReceiptId]) AS SubQuery
10 WHERE [SubQuery].[ItemIdCount] != [SubQuery].[ItemCount]
```

Returning no entries once the data was cleaned with a C# Script internally developed. This took into account not only checking for data integrity but generated new sql to facilitate item consolidation. These scripts have been listed in the appendices of this report.

4.2 Assumptions and Reasoning

The following section outlines our assumptions and the reasoning behind certain decisions made by the team at any stage of the project. The team will always try to apply business rules supplied by the firm to any of the decisions made however this is not always possible. In this case, the reasoning has been listed in the section that follows.

4.2.1 Item

There were no major assumptions made for the items. We followed the feedback received from the previous project and used the associative entity of ReceiptItem to satisfy requirements of sale price and quantity.

4.2.2 ReceiptItem

As suggested above, the ReceiptItem table was created to satisfy feedback received on our previous project , allows for quantity and price association.

4.2.3 Receipt

The receipt had no changes in comparison to the previous analysis, the changes occured in related entities and did not affect the receipt table.

4.2.4 Staff

Staff were left untouched, the relation to location however did play a large role in this analysis, and a major assumption of a staff member only being at one location at a time or being directly related only to the one location was made.
This assumption normally would not be made, but assurances of data cleanliness were made, and the C# script did not find any discrepencies between our assumption and the supplied information.

4.2.5 Customer

~~We made assumptions that customers are likely to shop at only the one store. Given the geographic distances between stores generally being larger than we deemed most people would be likely to travel.~~ This assumption was quickly invalidated as most customers seem to have shopped at most if not all stores.

4.2.6 Office

Our analysis assumed future office potential growth to be related to the locality of the store. Such an assumption may suggest that we weighed in favour of stores in high population areas if the analysis was borderline for example: Newcastle and Broken Hill; Newcastle would be favoured as the growth potential was assumed to be greater and more predictable into the future.

5 Base Analysis

5.1 Notes on Analysis

Analysis was performed using Microsoft SQL Management Studio, with a flat file of the original data being imported into the table as defined here, as with the previous assignment utilisation of decimal(18.7) was our choice over using the money type, again citing the imprecision of money in MSSQL/TSQL [1].

Source files to imitate the analysis below is either included inline under their respective headers or can be found in the source files within the final submission. The following section also contains tables with a selection of results from the corresponding SQL script.

5.2 Raw Results

5.2.1 Total Revenue Per Store

The total revenue per store after discounts were applied was determined by the following query. Below the query, the selected results were included.

```
-- Item Count, total revenue, sold By Office Location with discount
SELECT (CAST(
        CASE
        WHEN COUNT(ri.[ReceiptItemQuantity]) >= 5
            THEN SUM(ri.[SalePrice] * ri.[ReceiptItemQuantity]) * 0.95
        ELSE SUM(ri.[SalePrice] * ri.[ReceiptItemQuantity])
        END AS decimal(19,5))) AS 'Revenue',
o.OfficeId, o.OfficeLocation
FROM Receipt r
    INNER JOIN ReceiptItem ri
                ON r.ReceiptId = ri.ReceiptId
    INNER JOIN Staff s
                ON s.StaffId = r.ReceiptStaffId
    INNER JOIN Office o
                ON s.StaffOfficeId = o.OfficeId
GROUP BY o.OfficeId
        , o.OfficeLocation
ORDER BY Revenue DESC;
```

Revenue	OfficeId	OfficeLocation
$1807157.30	9	Wagga Wagga
$1243761.42	2	Maitland
$1206763.76	10	Broken Hill
$1113333.40	4	Sydney
$1109330.77	1	Newcastle
$1092924.88	5	Port Macquarie
$1006184.56	3	Cessnock
$1001501.30	6	Grafton
$934267.47	7	Dubbo
$891871.68	8	Wollongong

5.2.2 Total Number of Sales

The total number of sales per staff member were considered with the following query. Below the query, the selected results were included.

```
SELECT COUNT(*) AS 'Sales Count'
                , s.StaffId
                , s.StaffFirstName
                , s.StaffSurname
                , o.OfficeId
                , o.OfficeLocation
FROM Receipt r
        INNER JOIN ReceiptItem ri
                                ON r.ReceiptId = ri.ReceiptId
        INNER JOIN Item i
                                ON i.ItemId = ri.ItemId
        INNER JOIN Staff s
                                ON s.StaffId = r.ReceiptStaffId
        INNER JOIN Office o
                                ON o.OfficeId = s.StaffOfficeId
GROUP BY s.StaffId
                , s.StaffFirstName
                , s.StaffSurname
                , o.OfficeId
                , o.OfficeLocation
ORDER BY 'Sales Count' DESC;
```

Sales Count	StaffId	StaffFirstName	StaffSurname	OfficeId	OfficeLocation
645	S190	Samuel	Anderson	4	Sydney
628	S196	Devin	Brown	6	Grafton
628	S122	Austin	Morris	6	Grafton
608	S101	Jenna	Cox	5	Port Macquarie
607	S45	Emma	Gutierrez	3	Cessnock

This suggests to us that individual stores had, either more sales or higher performing employees, which we wanted to confirm in analysis later, of which can be found here.

5.2.3 Total Items Sold

The total items attributed to each staff member were considered also, determined by the query. Below the query, the selected results were included.

```
SELECT SUM(ri.ReceiptItemQuantity) AS 'Item Count'
                , s.StaffId
                , s.StaffFirstName
                , s.StaffSurname
                , o.OfficeId
                , o.OfficeLocation
FROM Receipt r
        INNER JOIN ReceiptItem ri
                                        ON r.ReceiptId = ri.ReceiptId
        INNER JOIN Staff s
                                        ON s.StaffId = r.ReceiptStaffId
        INNER JOIN Office o
                                        ON o.OfficeId = s.StaffOfficeId
GROUP BY s.StaffId
                , s.StaffFirstName
                , s.StaffSurname
                , o.OfficeId
                , o.OfficeLocation
ORDER BY 'Item Count' DESC;
```

Item Count	StaffId	StaffFirstName	StaffSurname	OfficeId	OfficeLocation
4005	S190	Samuel	Anderson	4	Sydney
3796	S122	Austin	Morris	6	Grafton
3730	S101	Jenna	Cox	5	Port Macquarie
3716	S45	Emma	Gutierrez	3	Cessnock
3684	S108	Isaiah	Powell	9	Wagga Wagga

5.2.4 Discounted Sales Ratio

Consideration of the number of sales made by each staff member was also made, the following query yielding the results we required. Below the query, the selected results were included.

```
SELECT s.StaffId
                , s.StaffFirstName
                , s.StaffSurname
                , o.OfficeId
                , o.OfficeLocation
                , 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
        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
        INNER JOIN Office o
                                ON o.OfficeId = s.StaffOfficeId
GROUP BY s.StaffId
                , s.StaffFirstName
                , s.StaffSurname
                , o.OfficeId
                , o.OfficeLocation
ORDER BY [Discounted Sales];
```

StaffId	StaffFirstName	StaffSurname	OfficeId	OfficeLocation	Discounted Sales	Standard Sales
S51	Haley	Taylor	7	Dubbo	263	91
S161	Jason	Wood	7	Dubbo	273	117
S17	Daniel	Baker	1	Newcastle	278	112
S135	Lexi	James	4	Sydney	280	100
S73	John	White	2	Maitland	286	121

5.2.5 Total Sales Value per Staff Member

```
SELECT CAST(
                CASE
                WHEN COUNT(ri.[ReceiptItemQuantity]) >= 5
                        THEN SUM(ri.[SalePrice] * ri.[ReceiptItemQuantity]) * 0.95
                ELSE SUM(ri.[SalePrice] * ri.[ReceiptItemQuantity])
                END AS decimal(19,5)) AS 'Sales Totals'
                , s.StaffId
                , s.StaffFirstName
                , s.StaffSurname
                , o.OfficeId
                , o.OfficeLocation
FROM Receipt r
        INNER JOIN ReceiptItem ri
                                ON r.ReceiptId = ri.ReceiptId
        INNER JOIN Item i
                                ON i.ItemId = ri.ItemId
        INNER JOIN Staff s
                                ON s.StaffId = r.ReceiptStaffId
        INNER JOIN Customer c
                                ON c.CustomerId = r.ReceiptCustomerId
        INNER JOIN Office o
                                ON o.OfficeId = s.StaffOfficeId
GROUP BY s.StaffId
                , s.StaffFirstName
                , s.StaffSurname
                , o.OfficeId
                , o.OfficeLocation
ORDER BY 'Sales Totals' DESC;
```

Sales Totals	StaffId	StaffFirstName	StaffSurname	OfficeId	OfficeLocation
$77152.49	S187	Savannah	Jones	8	Wollongong
$75113.60	S45	Emma	Gutierrez	3	Cessnock
$72475.45	S178	Kaitlyn	Nguyen	2	Maitland
$71814.20	S122	Austin	Morris	6	Grafton
$71497.09	S71	Danielle	Myers	6	Grafton

5.2.6 Average Value Per Sale

```
SELECT (CAST(
                CASE
                WHEN COUNT(ri.[ReceiptItemQuantity]) >= 5
                        THEN SUM(ri.[SalePrice] * ri.[ReceiptItemQuantity]) * 0.95
                ELSE SUM(ri.[SalePrice] * ri.[ReceiptItemQuantity])
                END AS decimal(19,5)) / COUNT(r.ReceiptId)) AS 'Sales Average'
                , s.StaffId
                , s.StaffFirstName
                , s.StaffSurname
                , o.OfficeId
                , o.OfficeLocation
        FROM Receipt r
        INNER JOIN ReceiptItem ri
                                        ON r.ReceiptId = ri.ReceiptId
        INNER JOIN Item i
                                        ON i.ItemId = ri.ItemId
        INNER JOIN Staff s
                                        ON s.StaffId = r.ReceiptStaffId
        INNER JOIN Office o
                                        ON o.OfficeId = s.StaffOfficeId
GROUP BY s.StaffId
                , s.StaffFirstName
                , s.StaffSurname
                , o.OfficeId
                , o.OfficeLocation
ORDER BY 'Sales Average' DESC;
```

Sales Average	StaffId	StaffFirstName	StaffSurname	OfficeId	OfficeLocation
$138.40	S109	Nicole	Hernandez	10	Broken Hill
$134.88	S187	Savannah	Jones	8	Wollongong
$134.71	S52	Isabella	Rivera	5	Port Macquarie
$134.53	S165	Marcus	Ross	2	Maitland
$134.12	S199	Maria	Smith	5	Port Macquarie

5.3 Total Items and Revenue per Store

```sql
-- Item Count, total revenue, sold By Office Location with discount
SELECT (CAST(
        CASE
        WHEN COUNT(ri.[ReceiptItemQuantity]) >= 5
            THEN SUM(ri.[SalePrice] * ri.[ReceiptItemQuantity]) * 0.95
        ELSE SUM(ri.[SalePrice] * ri.[ReceiptItemQuantity])
        END AS decimal(19,5))) AS 'Revenue',
SUM(ri.ReceiptItemQuantity) AS ItemCount,
o.OfficeId, o.OfficeLocation
FROM Receipt r
    INNER JOIN ReceiptItem ri
                ON r.ReceiptId = ri.ReceiptId
    INNER JOIN Staff s
                ON s.StaffId = r.ReceiptStaffId
    INNER JOIN Office o
                ON s.StaffOfficeId = o.OfficeId
GROUP BY o.OfficeId
        , o.OfficeLocation
ORDER BY ItemCount DESC;
```

Revenue	ItemCount	OfficeId	OfficeLocation
$1807157.30	97020	9	Wagga Wagga
$1243761.42	65637	2	Maitland
$1206763.76	65568	10	Broken Hill
$1109330.77	59946	1	Newcastle
$1113333.40	59229	4	Sydney
$1092924.88	56429	5	Port Macquarie
$1006184.56	52315	3	Cessnock
$1001501.30	51533	6	Grafton
$934267.47	49456	7	Dubbo
$891871.68	46433	8	Wollongong

5.4 Unique Customers Per Store

Total number of customers per store were determine via the below query:

```
-- Unique Customer Count Per Store
SELECT COUNT(*) AS 'Customer Count', o.[OfficeLocation]
FROM Customer c
INNER JOIN Receipt r ON r.ReceiptCustomerId = c.CustomerId
INNER JOIN Staff s ON s.StaffId = r.ReceiptStaffId
INNER JOIN Office o ON o.OfficeId = s.StaffOfficeId
GROUP BY o.[OfficeLocation]
ORDER BY 'Customer Count' DESC;
```

Customer Count	OfficeLocation
600	Wagga Wagga
584	Maitland
581	Port Macquarie
581	Broken Hill
577	Sydney
574	Newcastle
571	Cessnock
564	Dubbo
559	Grafton
556	Wollongong

This suggests that customers will shop at a number of stores, not just the single store in their locality. Having reflected upon the findings and knowing that the unique pool of customers in this dataset of 600 also suggests that a customer can order from anywhere and have the item delivered or transfered to a local store potentially. Given this, we assume that closing a store will not greatly impact customer retention if an online outlet can be reached.

5.5 Item Popularity

The script below addresses the concern of the CEO which required the team to analyse the popularity of items across all stores, and each store individually. The SQL query joins multiple tables in our database and uses an integer, which is a unique identifier for each office location. The results of this script can be found listed below in two separate outputs. The first outlines the highest performing items per store whereas the second highlights the worst performing items for each store.

```
1  SELECT SUM(ri.ReceiptItemQuantity) AS ItemCount
2  , i.ItemId
3  , i.ItemDescription
4  , o.OfficeId
5  , o.OfficeLocation
6      FROM Receipt r
7          INNER JOIN ReceiptItem ri
8                      ON ri.ReceiptId = r.ReceiptId
9          INNER JOIN Item i
10                     on i.ItemId = ri.ItemId
11         INNER JOIN Staff s
12                     on s.StaffId = r.ReceiptStaffId
13         INNER JOIN Office o
14                     on o.OfficeId = s.StaffOfficeId
15     WHERE o.OfficeId = 1
16     GROUP BY i.ItemId
17             , i.ItemDescription
18             , o.OfficeId
19             , o.OfficeLocation
20     ORDER BY ItemCount ASC;
```

Office ID	Item ID	Highest Quality 1	Item ID	Highest Quality 2	Item ID	Highest Quality 3
1	24	2144	13	2153	26	2176
2	7	2328	28	2371	5	2374
3	10	1885	18	1900	22	2144
4	16	2151	20	2157	4	2185
5	20	2038	27	2042	10	2051
6	25	1846	16	1856	2	1881
7	20	1807	30	1808	24	1864
8	5	1674	8	1696	16	1765
9	25	3405	21	3478	14	3566
10	14	2376	22	2384	8	2436

Office ID	Item ID	Lowest Quality 1	Item ID	Lowest Quality 2	Item ID	Lowest Quality 3
1	11	1756	2	1765	17	1866
2	27	1947	9	1966	2	2027
3	5	1465	8	1500	27	1561
4	23	1767	13	1810	14	1829
5	1	1563	8	1663	6	1680
6	29	1537	20	1562	17	1573
7	25	1397	4	1471	2	1494
8	14	1383	17	1422	24	1424
9	16	2945	23	3001	28	3010
10	5	1939	11	1962	13	1980

5.6 Worst Performing Item

Listed below is the SQL scripts associated with finding the worst performing item, overall, across the entire company including all locations. The results for this SQL script can be found below. These results list the count of items sold, the items unique identifier and a description of the item.

```
--worst perfroming items for the company as whole
SELECT SUM(ri.ReceiptItemQuantity) AS ItemCount
        , i.ItemId
        , i.ItemDescription
FROM Receipt r
    INNER JOIN ReceiptItem ri
                ON ri.ReceiptId = r.ReceiptId
    INNER JOIN Item i
                ON i.ItemId = ri.ItemId
GROUP BY i.ItemId, i.ItemDescription
ORDER BY ItemCount ASC;
```

ItemCount	ItemId	ItemDescription
19481	23	Drill Bit 6mm
19613	13	Ruler
19741	2	Screwdriver Set
19744	9	Box of Screws
19746	17	Punch

6 Conclusion and Recommendations

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

[4] Upselling - Business Dictionary http://www.businessdictionary.com/definition/upselling.html

7 Appendix

7.1 C# Code Excerpt: Cleaning

```
public Tuple<Dictionary<int, Receipt>, Dictionary<int, Receipt>, int> GenerateReceipts(DataSet dataSet)
{
    var receipts = new Dictionary<int, Receipt>();
    var invalidReceipts = new Dictionary<int, Receipt>();
    var counter = 0;

    // Iterate over each sheet in the dataset, in our case it's only one anyway.
    foreach (DataTable table in dataSet.Tables)
        // Iterate over each row in the dataset, it seems this iterates until it finds row with no values in any
            column.
        foreach (DataRow row in table.Rows)
        {
            // Ignore the header of the table
            if (counter > 0)
            {
                // Generation of receipt from dataset, ignoring transaction row id as it is not
                // relevant and we are consolidating the duplicate items into total quantities anyway.
                var receipt = new Receipt
                {
                    Id = Convert.ToInt32(row[1]),
                    Customer = new Customer
                    {
                        FirstName = row[3].ToString(),
                        Id = row[2].ToString(),
                        Surname = row[4].ToString()
                    },
                    Staff = new Staff
                    {
                        OfficeId = Convert.ToInt32(row[8]),
                        Id = row[5].ToString(),
                        FirstName = row[6].ToString(),
                        Surname = row[7].ToString()
                    },
                    SaleDate = (DateTime)row[0],
                    Items = new Dictionary<int, Item>
                {
                    {
                        Convert.ToInt32(row[11]), new Item
                        {
                            Id = Convert.ToInt32(row[11]),
                            Description = row[12].ToString(),
                            Price = (double) row[14],
                            Quantity = Convert.ToInt32(row[13])
                        }
                    }
                },
                    Office = new Office
                    {
                        Id = Convert.ToInt32(row[8]),
                        OfficeLocation = ParseEnum<Location>(row[9].ToString())
                    }
                };
```

```
57                  // Check if receipts either is empty or doesn't have a receipt
58                  // with the same receipt id already, if so add the receipt to
59                  // receipts.
60                  if (receipts.Count == 0 || !receipts.ContainsKey(receipt.Id))
61                  {
62                      AddToReceipts(receipts, receipt);
63                      Console.WriteLine($@"Added new receipt: {receipt.Id}");
64                  }
65                  //
66                  else if (receipts.ContainsKey(receipt.Id))
67                  {
68                      var existingReceipt = receipts[receipt.Id];
69
70                      // Be warned; below is a fucking mess.
71
72                      // Check all staff properties match the existing receipt properties.
73                      if (existingReceipt.Staff.Id != receipt.Staff.Id
74                          || existingReceipt.Staff.FirstName != receipt.Staff.FirstName
75                          || existingReceipt.Staff.Surname != receipt.Staff.Surname)
76                      {
77                          AddToReceipts(invalidReceipts, receipt);
78                          Console.WriteLine($"Staff Id mismatch on receipt: {existingReceipt.Id}, " +
79                                            $"Staff Ids: {existingReceipt.Staff.Id} and {receipt.Staff.Id}, " +
80                                            $"Staff Names: {existingReceipt.Staff.FirstName} and
                                                  {receipt.Staff.FirstName}, " +
81                                            $"Staff Surnames: {existingReceipt.Staff.Surname} and
                                                  {receipt.Staff.Surname}, ");
82                          continue;
83                      }
84
85                      // Check all office properties match the existing receipt properties.
86                      if (existingReceipt.Office.Id != receipt.Office.Id
87                          || existingReceipt.Office.OfficeLocation != receipt.Office.OfficeLocation)
88                      {
89                          AddToReceipts(invalidReceipts, receipt);
90                          Console.WriteLine($"Office Id mismatch on receipt: {existingReceipt.Id}, " +
91                                            $"Office Ids: {existingReceipt.Office.Id} and {receipt.Office.Id}, " +
92                                            $"Office Locations: {existingReceipt.Office.OfficeLocation} and
                                                  {receipt.Office.OfficeLocation}");
93                          continue;
94                      }
95
96                      // Check all date properties match the existing receipt properties.
97                      if (existingReceipt.SaleDate != receipt.SaleDate)
98                      {
99                          AddToReceipts(invalidReceipts, receipt);
100                         Console.WriteLine($"Sale date mismatch on receipt: {existingReceipt.Id}, " +
101                                           $"Dates: {existingReceipt.SaleDate} and {receipt.SaleDate}");
102                         continue;
103                     }
104
105
106
107
108
109
110
111
112
113
114
```

```
115                     // Check all customer properties match the existing receipt properties.
116                     if (existingReceipt.Customer.Id != receipt.Customer.Id
117                         || existingReceipt.Customer.FirstName != receipt.Customer.FirstName
118                         || existingReceipt.Customer.Surname != receipt.Customer.Surname)
119                     {
120                         AddToReceipts(invalidReceipts, receipt);
121                         Console.WriteLine($"Customer mismatch on receipt: {existingReceipt.Id}, " +
122                                           $"Customers Ids: {existingReceipt.Customer.Id} and {receipt.Customer.Id}, "
                                                +
123                                           $"Customers Names: {existingReceipt.Customer.FirstName} and
                                                {receipt.Customer.FirstName}, " +
124                                           $"Customers Surnames: {existingReceipt.Customer.Surname} and
                                                {receipt.Customer.Surname}, ");
125                         continue;
126                     }
127
128                     // If not item mismatches exist, we need to determine if the item
129                     // exists in both the compared receipts.
130                     var key = receipt.Items.First().Key;
131                     var value = receipt.Items.First().Value;
132
133                     // Check if the item exists in the existing receipt already.
134                     if (existingReceipt.Items.ContainsKey(receipt.Items.First().Key))
135                     {
136                         var quantity = receipt.Items.First().Value.Quantity;
137                         Console.WriteLine($"Item exists in current receipt with Id: {key}, " +
138                                           $"updated item quantity from: {existingReceipt.Items[key].Quantity} " +
139                                           $"to new quantity: {existingReceipt.Items[key].Quantity + quantity}");
140
141                         // Update quantity on item in existing receipt.
142                         existingReceipt.Items[key].Quantity += quantity;
143                         receipts[existingReceipt.Id] = existingReceipt;
144                     }
145                     else
146                     {
147                         // Update the receipt with the new item added.
148                         existingReceipt.Items.Add(key, value);
149                         Console.WriteLine($@"Added item with Id: {value.Id} to existing receipt: {receipt.Id}");
150                     }
151                 }
152             }
153             counter++;
154         }
155
156     return new Tuple<Dictionary<int, Receipt>, Dictionary<int, Receipt>, int>(receipts, invalidReceipts,
157         counter);
158 }
```

7.2 C# Code Excerpt: SQL Generation

```
public string GenerateSQL(Dictionary<int, Receipt> receipts)
{
    var output = new StringBuilder();

    foreach(var receipt in receipts)
    {
        var itemTotalPrices = new Dictionary<int, double>();

        foreach (var item in receipt.Value.Items)
        {
            var totalItemPrice = item.Value.Price * item.Value.Quantity;
            itemTotalPrices.Add(item.Key, totalItemPrice);
        };

        var total = itemTotalPrices.Sum(x => x.Value);
        var discountTotal = (itemTotalPrices.Count >= 5) ? total * 0.95 : 0;

        var sql = new StringBuilder($@"


    INSERT INTO [Receipt]
    VALUES('{receipt.Value.SaleDate.ToString("G",CultureInfo.CreateSpecificCulture("en-us"))}',
            {receipt.Key},
            '{receipt.Value.Customer.Id}',
            '{receipt.Value.Staff.Id}',
            {total},
            {discountTotal});");

        foreach (var item in itemTotalPrices)
        {
            sql.Append($@"

    INSERT INTO [ReceiptItem]
    VALUES( {receipt.Key},
            {item.Key},
            {receipt.Value.Items.Select(x => x.Value).Where(y => y.Id == item.Key).FirstOrDefault().Quantity},
            {receipt.Value.Items.Select(x => x.Value).Where(y => y.Id == item.Key).FirstOrDefault().Price});");
        }

        output.Append(sql);
    };

    return output.ToString();
}
```