Off the Vine

An Oracle 19c & MySQL Project

By Charlie Evert
DSS 640

Introduction

Background

Off the Vine is a fictional company that sells Tomatoes and Strawberries worldwide for several moths every year. As an internationally sought-after brand, Off the Vine must harness the power of data analysis to set KPI objectives with. It has several data sources, including its sales journal, its product catalog, its location catalog (where its products are sold), its customer list (sourced from each transaction) and its payment processing services.

The following write-up will summarize ways that Off the Vine can normalize its enterprise's data and integrate it into Oracle 19c so that it may effectively capitalize upon the power of data driven insights.

Scope

The data sources that Off the Vine uses have many common entities, all of which can be easily normalized through the use of Primary and Foreign keys. Off the Vine was careful to name tables and columns with the same value with the same name, as well, which simplifies the normalization process. The goal of this project is to consolidate the data from each of its sources into a cohesive data warehouse that can seamlessly integrate into Off the Vine's analytical needs.

Off the Vine's full process of data normalization and integration will be covered in this report, from its schema reconciliation to its star schema (with pertinent metadata included). Several queries, groupings, views and applications will also be explored in this report.

Scope limitations include access to data sources (the majority of the data was simulated), the absence of a live connection to transaction data and the inability for data to be seamlessly integrated with Oracle 19c (instead of needing to be converted from an Excel file).

Logical Design

Individual Schemas

Sales Journal

SALES(Trans_ID, Prod_ID, Location_ID, Empl_ID, Cust_ID, Payment_Type, Price, Qty, Total)

TRANSACTIONDATE(Trans_ID, Date)

Product Catalog

PRODUCT(Prod_ID, Prod_Name, Prod_Type)

PRESICIDE(Pest_Manu, Prod_ID)

Location Catalog

LOCATION(Location_ID, Neighborhood, State/Province, Country)

Customer List

CUSTOMER(Customer_ID, First_Name, Last_Name, DOB)

Payment Processing

Payment_Type(Payment_Type, Decription)

Reconciled Schema

Due to the fact that this data is fictionally simulated, and was ideally constructed, no aspects need extensive reconciliation. Each separate table, taken from Excel, can be seamlessly reconciled by way of foreign keys (as demonstrated below, they are referred to by _FK). Additionally, I designated primary keys with the suffix _PK.

Off the Vine Enterprise Data Warehouse

SALES(Trans_ID_PK, Prod_ID, Location_ID, Empl_ID, Cust_ID, Payment_Type, Price, Qty, Total)

TRANSACTIONDATE(Trans_ID_PK_FK, Date)

PRODUCT(Prod_ID_PK_FK, Prod_Name, Prod_Type)

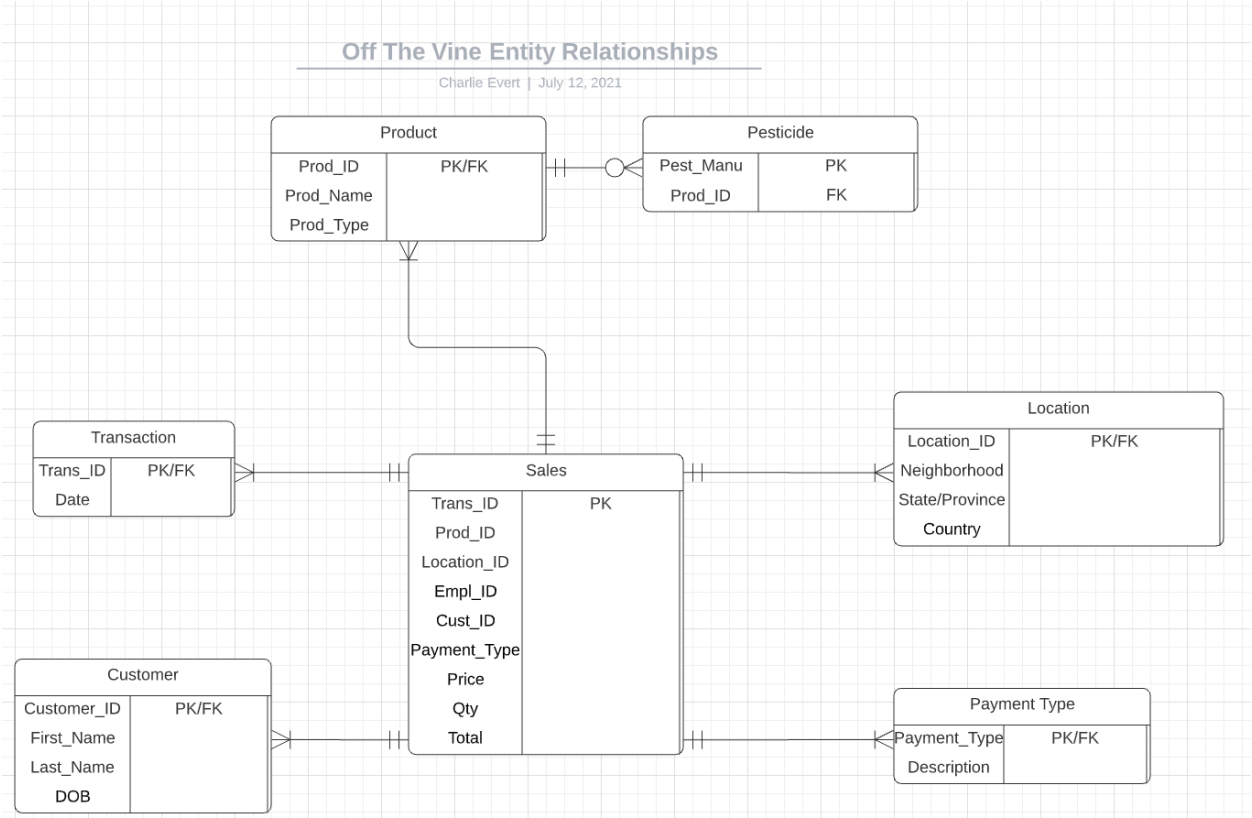
PRESICIDE(Pest_Manu_PK, Prod_ID_FK)

LOCATION(Location_ID_PK_FK, Neighborhood, State/Province, Country)

CUSTOMER(Customer_ID_PK_FK, First_Name, Last_Name, DOB)

Payment_Type(Payment_Type_PK_FK, Decription)

Pertinent Meta Data & Final Conceptual/Fact Schemata



The above Entity Relationship Diagram illustrates the final star/fact schema used for Off the Vine enterprise's database. Most relationships illustrated are one to many relationships between the central table in the star schema (sales) and its branched tables; this is due to the fact that each table is interconnected, and helps to describe some aspect of a simulated ledger.

Please see Appendix A for constraints.

The following is pertinent metadata:

- Trans_ID, Prod_ID, Location_ID, Empl_ID, Cust_ID, & Qty are all of the Integer (INT) data type.
- Prod_Name, Prod_Type, First_Name, Last_Name, Neighborhood, State/Province, Country & Description are all String (VARCHAR) data types.
- DOB & Date are both Date and Time (DATETIME) data types.

- Price and Total are both Float (NUMERIC) data types.

Data Staging

To start, tables were created via the CREATE TABLE function with the data types listed above (as Pertinent Meta Data).

Then, data sourced from Excel was imported into Oracle 19c via the 'Actions…' button in Oracle 19c.

This data did not need any transformation due to it being randomized and simulated appropriately; there were no missing values to worry about. Values were simulated using various RANDBETWEEN functions in Excel.

Implementation

Aggregate Queries

The following aggregate queries help to explain the data via aggregations. The number of transactions, as well as the maximum/minimum amount of money spent in one transaction were analyzed. There is a wide range of values, ranging from $.50 to $20 over the course of 500 transactions. See appendix B for outputs for each aggregate query.

```
SELECT COUNT(Trans_ID) AS "Number of Transactions"
FROM SALES;
```

The picture above illustrates a COUNT query, which returns N. N is useful for determining how many sales it took to run out of inventory (or simply the number of sales between periods). This can help to set benchmarks for subsequent periods.

```
SELECT MAX(Total) AS "Maximum Money Spent in One Transaction"
FROM SALES;
```

The picture above illustrates a MAX query, which returns the maximum total value of a transaction. This helps identify which customers to incentivize for repeat visits (so that they spend as much as they can).

```
SELECT MIN(Total) AS "Minimum Money Spent in One Transaction"
FROM SALES;
```

The picture above illustrates a MIN query, which returns the minimum total value of a transaction. This is helpful in order for analytics professionals to help incentivize lower paying customers to buy more, via upselling and promotions.

Join Queries

The following join queries join multiple tables and display key disaggregated values. Each provides a handy joining of attributes between tables that can display data derived from key business operations. Outputs are visible in Appendix C.

```
SELECT VINECUSTOMER.First_Name AS First, VINECUSTOMER.Last_Name AS Last, SALES.Total
FROM SALES
INNER JOIN VINECUSTOMER ON SALES.Cust_ID=VINECUSTOMER.Customer_ID;
```

The above query would help managers at Off the Vine to identify customers that spend money multiple times on their products. These customers are valuable, and data indicating who they are is helpful for future advertising.

```
SELECT DISTINCT(LOCATION.Neighborhood), LOCATION.State_Province AS "State or Province", LOCATION.Country, SALES.Total
FROM LOCATION
INNER JOIN SALES ON LOCATION.Location_ID=SALES.Location_ID;
```

This query demonstrates each neighborhood's distinct total sales volumes. This is helpful in determining which towns would be best to export to (since towns with higher sales volume have the potential to garner more sales volume through increased product availability). It also lists the country of the town, which is helpful in determining exports as well since it makes sense to export more tomatoes and strawberries to countries have higher demand for them (in this case, Canada's demand is superior to the United States).

Groupings

The following Grouping queries are helpful to Off the Vine's analysts in identifying the relative performance of key attributes. This is helpful to managers in that ill-performing or bottlenecking operations identified via grouping can be trimmed or discontinued to increase the bottom line. Appendix D details the outputs of the following groupings.

```
SELECT COUNT(Cust_ID), Payment_Type
FROM SALES
GROUP BY Payment_Type
ORDER BY COUNT(Cust_ID) DESC;
```

The above grouping demonstrates a breakdown of how many customers used each payment type. In this case, few people used 2nd payment method (card) whereas significantly more used checks. One could conclude that, for whatever reason, processes must be optimized for the facilitating of payment via checks for Off the Vine's strawberries and tomatoes; perhaps local customs are responsible and the majority Canadian consumer enjoys the act of check writing. Managers should address this, given the bottlenecking associated with spending a few minutes writing a check instead of swiping a card.

```
SELECT
    COUNT(sales.cust_id),
    sales.location_id,
    system.location.neighborhood,
    system.location.country
FROM
        sales
    INNER JOIN system.location ON sales.location_id = system.location.location_id
GROUP BY
    sales.location_id,
    system.location.neighborhood,
    system.location.country
ORDER BY
    COUNT(sales.cust_id) DESC
```

The grouping query above demonstrates the count of customers that bought from Off the Vine per neighborhood. This is similar to the second join query in the previous section, but is superior given its ease of use. This query is, like its predecessor, useful for identifying target markets as well as underperforming markets. Managers should use this data to adjust exports to each neighborhood accordingly.

Materialized Views

Views are helpful for creating repeatable, easy to read outputs that, if used with a dynamic data set, can be up to date after every time a view is selected. The following views provide broad overlooks of attributes between tables, and each view can be integral towards managerial decision-making. Appendices E demonstrate outputs.

```
Create or Replace View neighborhood AS _
SELECT
    system.product.prod_name,
    system.location.neighborhood,
    system.location.country,
    system.vinecustomer.first_name,
    system.vinecustomer.last_name
FROM
        system.location
    INNER JOIN system.sales ON system.sales.location_id = system.location.location_id
    INNER JOIN system.product ON system.sales.prod_id = system.product.prod_id
    INNER JOIN system.vinecustomer ON system.sales.cust_id = system.vinecustomer.customer_id;

SELECT * FROM Sales_Journal
ORDER BY neighborhood DESC;
```

The above view, “neighborhood”, displays each sale broken down into which product was sold, where, and to whom. This is helpful in determining repeat customers and what they bought by volume, so that exports can be adequate to maximize sales.

```
CREATE OR REPLACE VIEW Above_Average_Totals AS
SELECT Trans_ID, Total
FROM SALES
WHERE Total > (SELECT AVG(Total) FROM SALES);

SELECT * FROM Above_Average_Totals
```

Above_Average_Totals view, pictured above, demonstrates a repeatable view that calls all transaction IDs (along with their associated totals) that were above average. Above average transactions may illuminate trends regarding particular time frames that lead to higher-than-average totals; the variables creating such periods could then be further analyzed to determine how to incentivize customers to have larger totals.

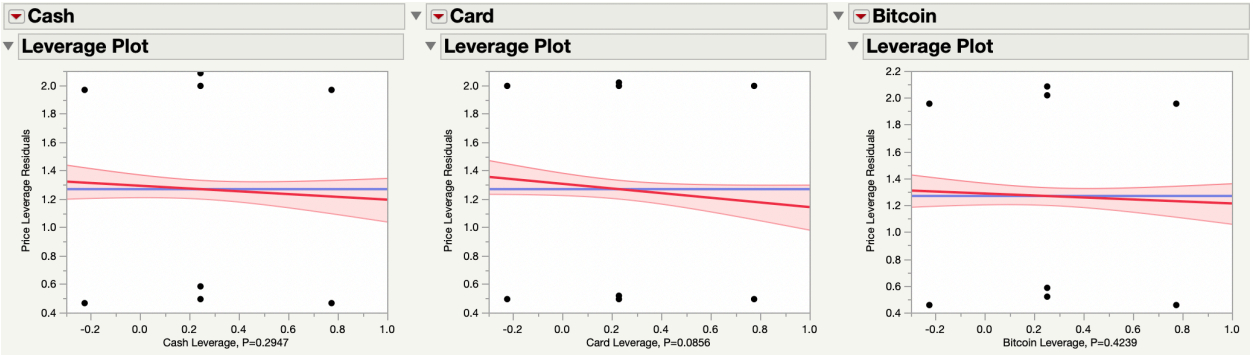
```
CREATE OR REPLACE VIEW STDEV_TOTAL AS
SELECT STDDEV(Total) AS "Standard Deviation"
FROM SALES;

SELECT * FROM STDEV_TOTAL;
```

The STDEV_TOTAL view above shows the overall standard deviation of totals (per transaction). This standard deviation of approximately $6 works well with the Above_Average_Totals view above to demonstrate exactly how rare a given total would be. Being able to call this function rapidly helps analysts to see if standard deviations (and variations) are changing given new data. If standard deviations suddenly change, this is either a cause of concern or evidence that a marketing decision to spur increase sales per transaction is working.

Business Analytics Applications

Artificially Intelligent Forecasts via JMP

Artificial intelligence can help to predict revenues for each location or employee, among other predictive analytics. After dummy coding Cash, Card and Bitcoin and forming multivariate models that correlate with revenues (per transaction), it is clear that there is little difference amongst payment methods in predicting revenues. Please see below for JMP's artificially intelligent output (which is reflective of random inputs being used).



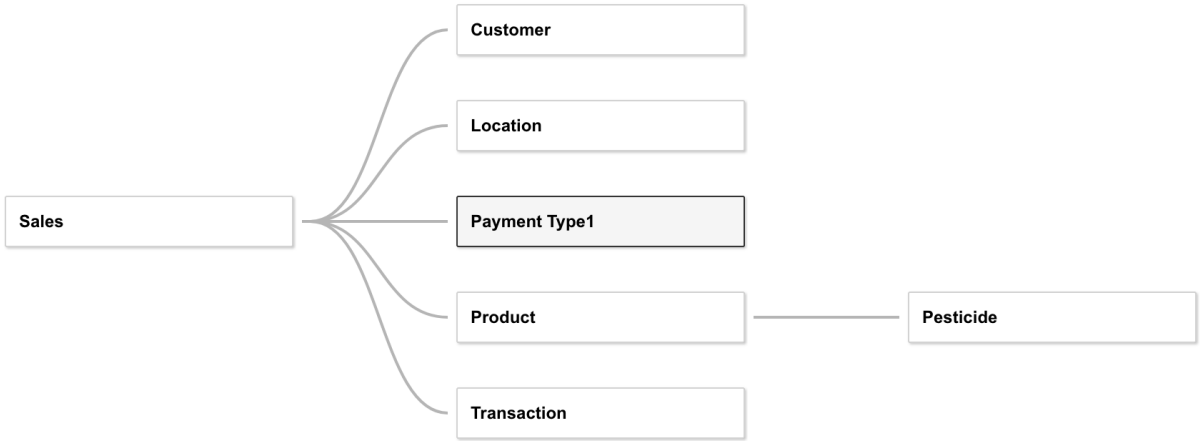
Below is JMP's prediction estimate for revenue given payment types. From this equation, we can conclude that Bitcoin payments typically lead to the highest amount of sales revenues whereas card payments lead to the lowest amount of sales revenues.

Prediction Expression

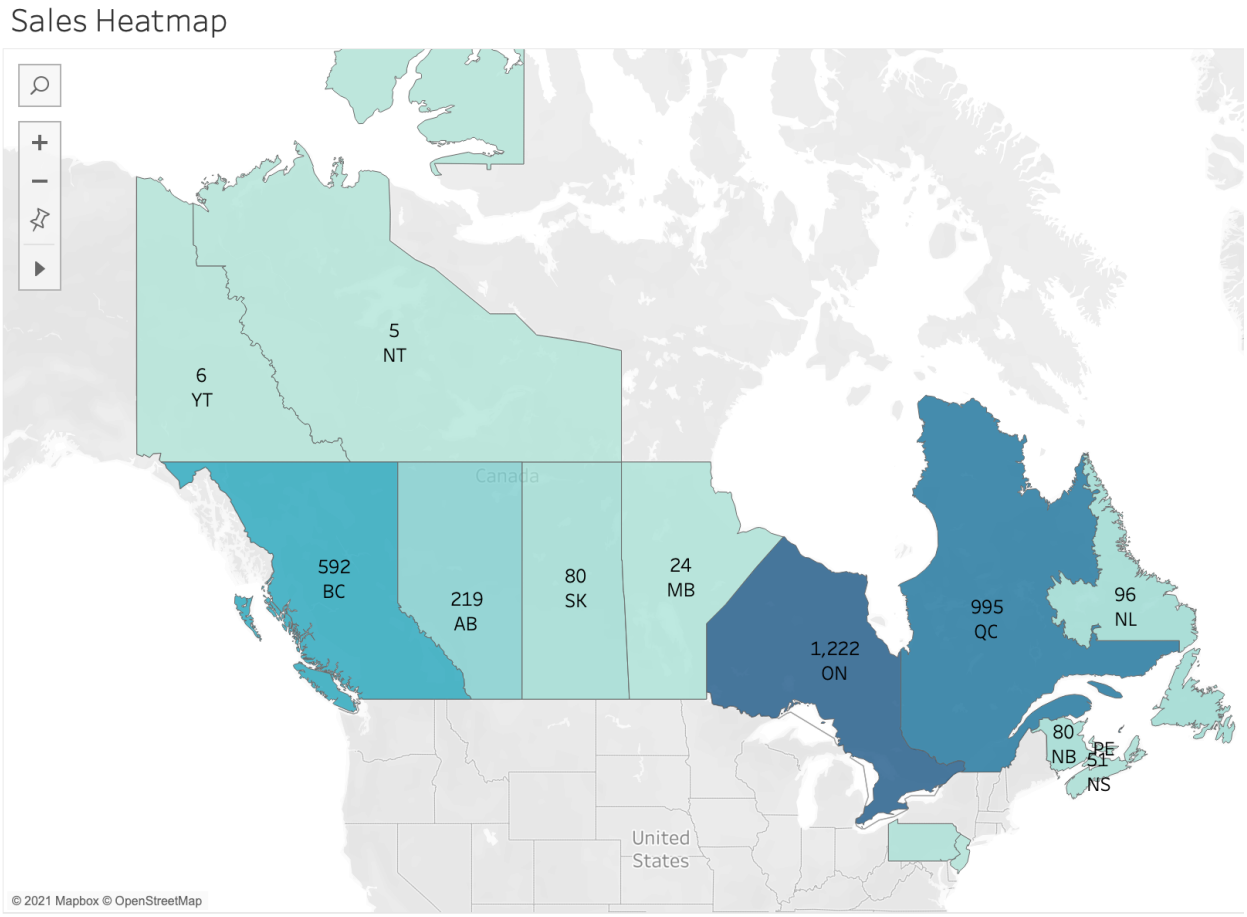
$$1.347826087 + -0.097826087 \bullet \text{Cash} + -0.163615561 \bullet \text{Card} + -0.074016563 \bullet \text{Bitcoin}$$

Niche Identification via Tableau

IoT can help to identify customer trends, including finding particular locations with people more likely to purchase from Off the Vine, helping further distribution and the profitability of advertising. IoT applies sales revenue numbers since cash registers and payment processing systems automatically upload this information online, making the data easily accessible to Off the Vine. Please see below for how easily a star schema is assembled in Tableau for the purpose of analysis given clean, normalized enterprise data.



Once data is assembled into a star schema, attributes that aren't connected in a single table (but are connected in a data warehouse by foreign keys) can lead to easily identifying trends. Using Geographic Information Systems in Tableau derived from Off the Vine's data, I constructed a heatmap viewable below that demonstrates a high concentration of total sales as being on the Canadian coasts, with very few in the United States.



From this application, it can be gleamed that Coastal Canadians really love Off the Vine's strawberries and tomatoes; analytics, here, demonstrates that advertising to regions that aren't Quebec, British Columbia or Ontario is not important to revenues. Thus, these other regions should not be sold to if advertising costs are equivalent in each region. As an aside, these findings are likely due to the fact that most neighborhoods gathered for the purpose of random sampling were in these regions; the data itself follows a normal distribution reminiscent of simulation.

Conclusion

This project was great in bridging gaps between various data managerial and analytical tools that I have frequently worked with. I learned that it can be quite simple to build complete databases at the enterprise level through the use of Oracle and MySQL.

Some short comings of this project include:

- Data was simulated, and therefore insights from this data are difficult to come by (other than illustrating the central limit theorem)
- Data was static and unchanging, so changes in queries (especially views) cannot be observed and their utility is therefore lessened
- I found difficulty in regulating referential integrity constraint, so I kept the data as akin to a star schema as possible to minimize any errors

In future projects, I will:

- Source real data with the potential for actionable insights
- Work with live data sources
- Explore more obscure aggregative functions

Overall, I felt that this project's practicum helped me grasp database management far more effectively than watching lectures/videos and reading due to the hands-on nature of database management at the enterprise level.

Appendix A: Constraints

DSS 630.sql | Welcome Page | DSS 630 | LOCATION

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | D

Actions...

	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION
1	LOCATION_PK	Primary Key	(null)

DSS 630.sql | Welcome Page | DSS 630 | PRODUCT

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | De

Actions...

	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION
1	PRODUCT_PK	Primary_Key	(null)

DSS 630.sql | Welcome Page | DSS 630 | PESTICIDE

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | De

Actions...

	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION
1	PESTICIDE_PK	Primary_Key	(null)
2	PEST_FK	Foreign_Key	(null)

DSS 630.sql | Welcome Page | DSS 630 | PAYMENT_TYPE

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies |

Actions...

	CONSTRAINT_...	CONSTRAINT_TYPE	SEARCH_CONDITION	R_OWNER
1	PAYMENT_TYPE_PK	Primary_Key	(null)	(null)

DSS 630.sql | Welcome Page | DSS 630 | VINECUSTOMER

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies |

Actions...

	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	R_OWNER
1	CUSTOMER_PK	Primary_Key	(null)	(null)

DSS 630.sql | Welcome Page | DSS 630 | TRANSACTION_DATE

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies |

Actions...

	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	R_OWNER
1	TRANS_DATE_PK	Primary_Key	(null)	(null)

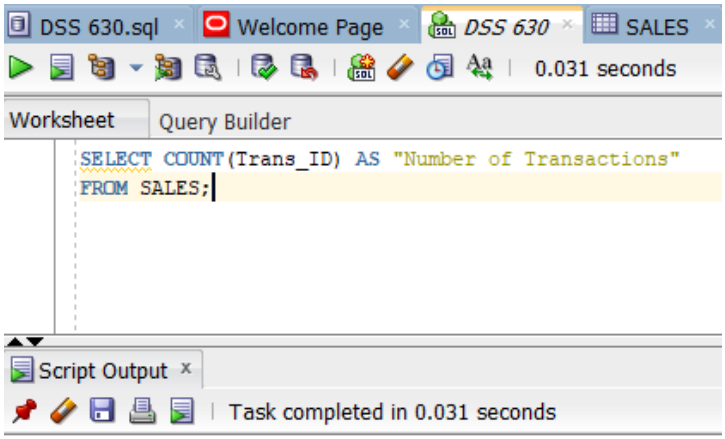
DSS 630.sql | Welcome Page | DSS 630 | SALES

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

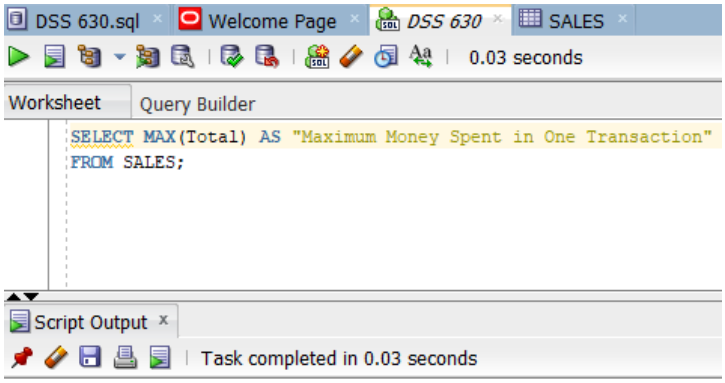
Actions...

	CONSTRAINT_NAME	CONSTRAINT_TYPE	SEARCH_CONDITION	R_OWNER	R_TABLE_NAME	R_CONSTRAINT_NA
1	CUST_FK	Foreign_Key	(null)	SYSTEM	VINECUSTOMER	CUSTOMER_PK
2	LOCATION_FK	Foreign_Key	(null)	SYSTEM	LOCATION	LOCATION_PK
3	PMT_TYPE_FK	Foreign_Key	(null)	SYSTEM	PAYMENT_TYPE	PAYMENT_TYPE_PK
4	PROD_FK	Foreign_Key	(null)	SYSTEM	PRODUCT	PRODUCT_PK
5	SALES_PK	Primary_Key	(null)	(null)	(null)	(null)
6	TRANS_DATE_FK	Foreign_Key	(null)	SYSTEM	TRANSACTION_DATE	TRANS_DATE_PK

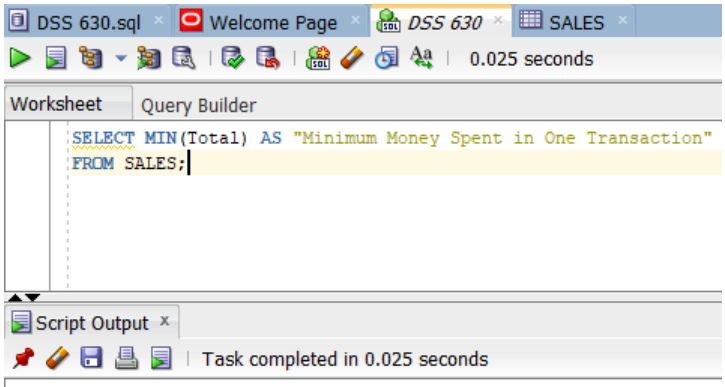
Appendix B: Aggregate Queries




```
Number of Transactions
----------------------
                   500
```

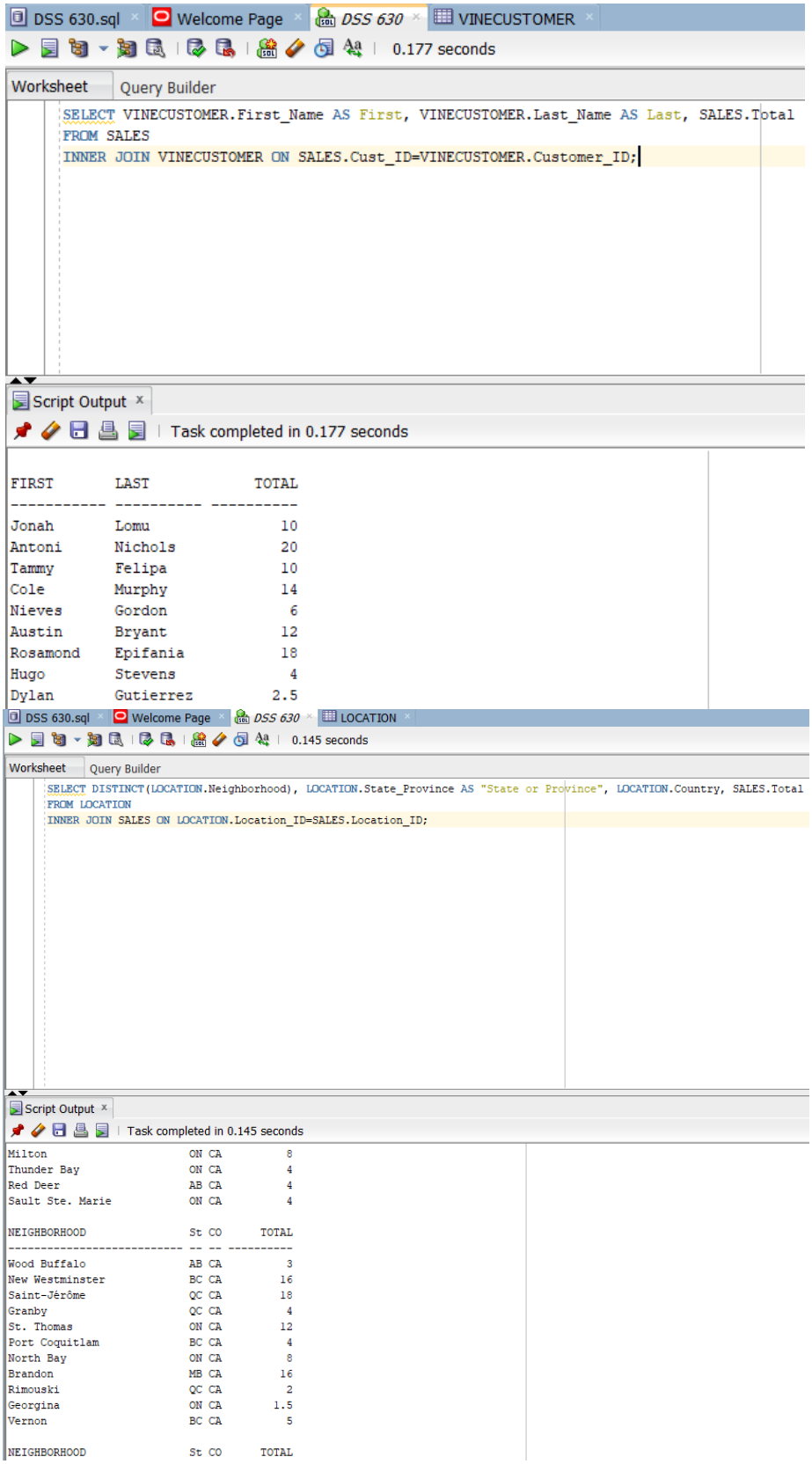



```
Maximum Money Spent in One Transaction
--------------------------------------
                                    20
```

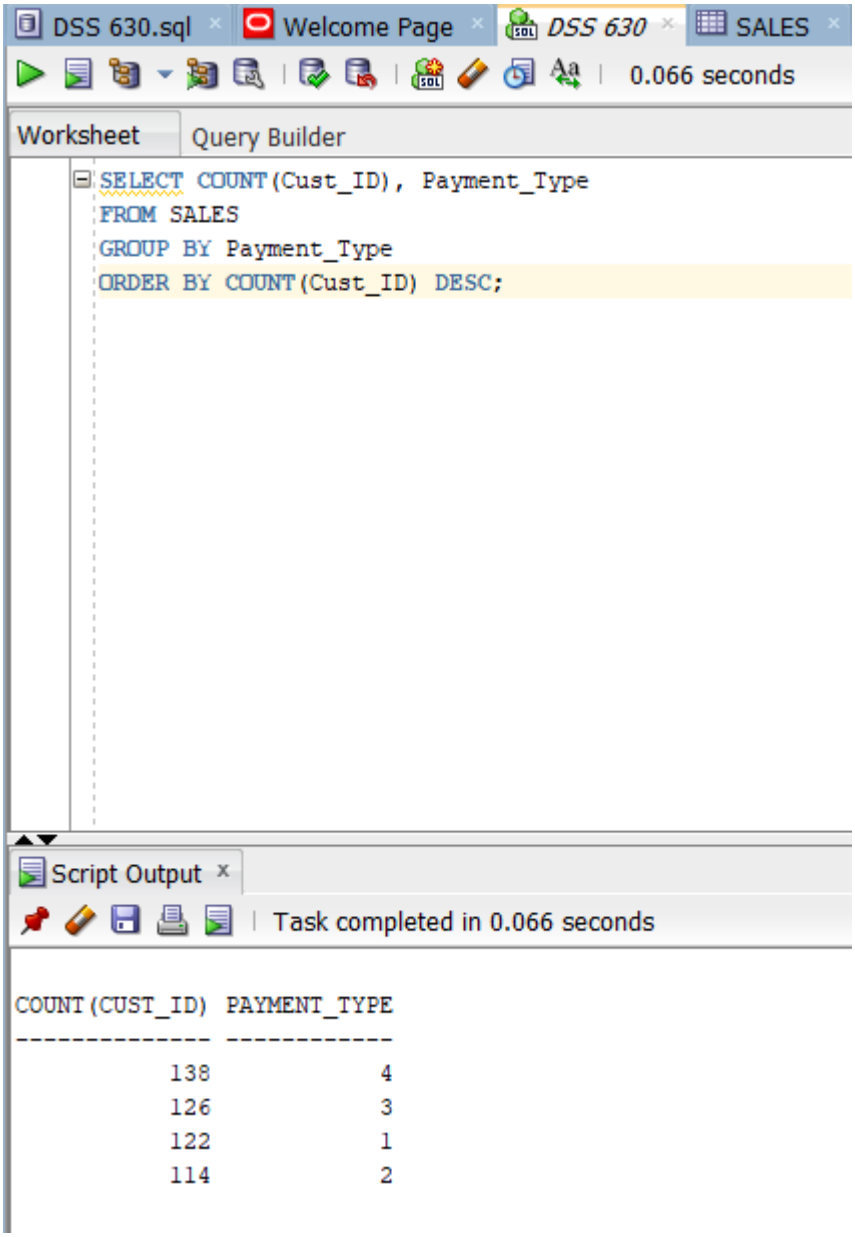



```
Minimum Money Spent in One Transaction
--------------------------------------
                                    .5
```

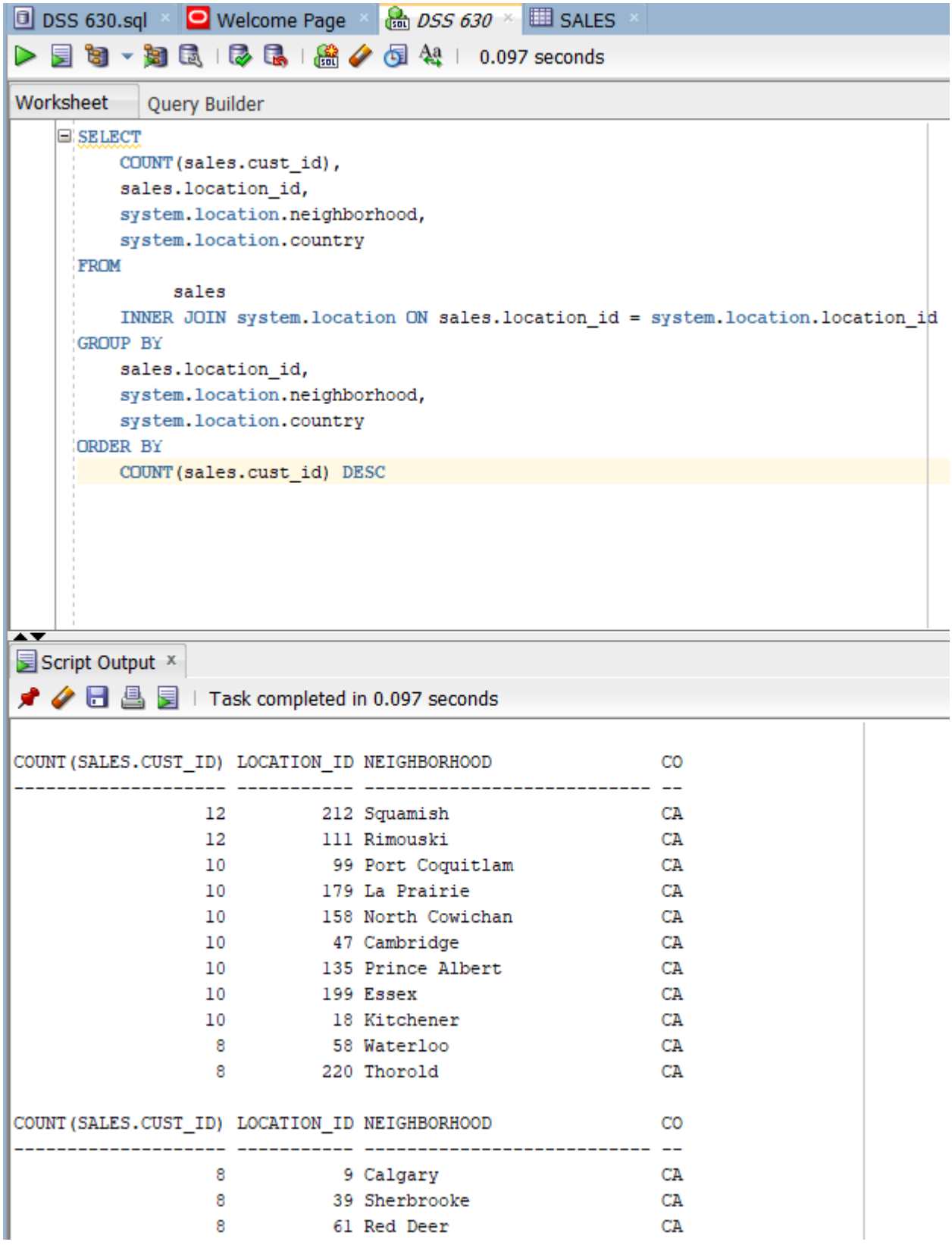
Appendix C: Join Queries



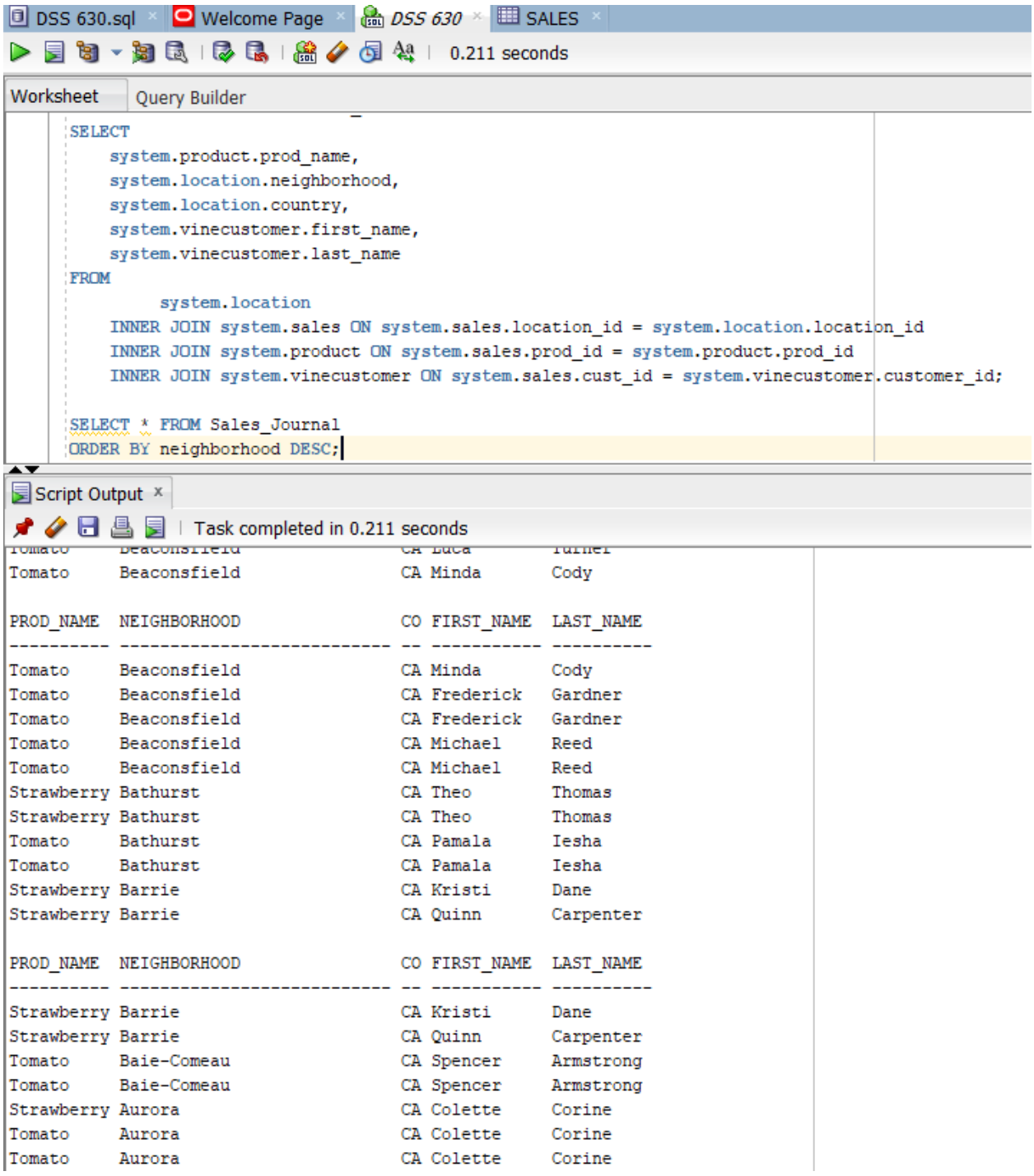
Appendix D1: Groupings



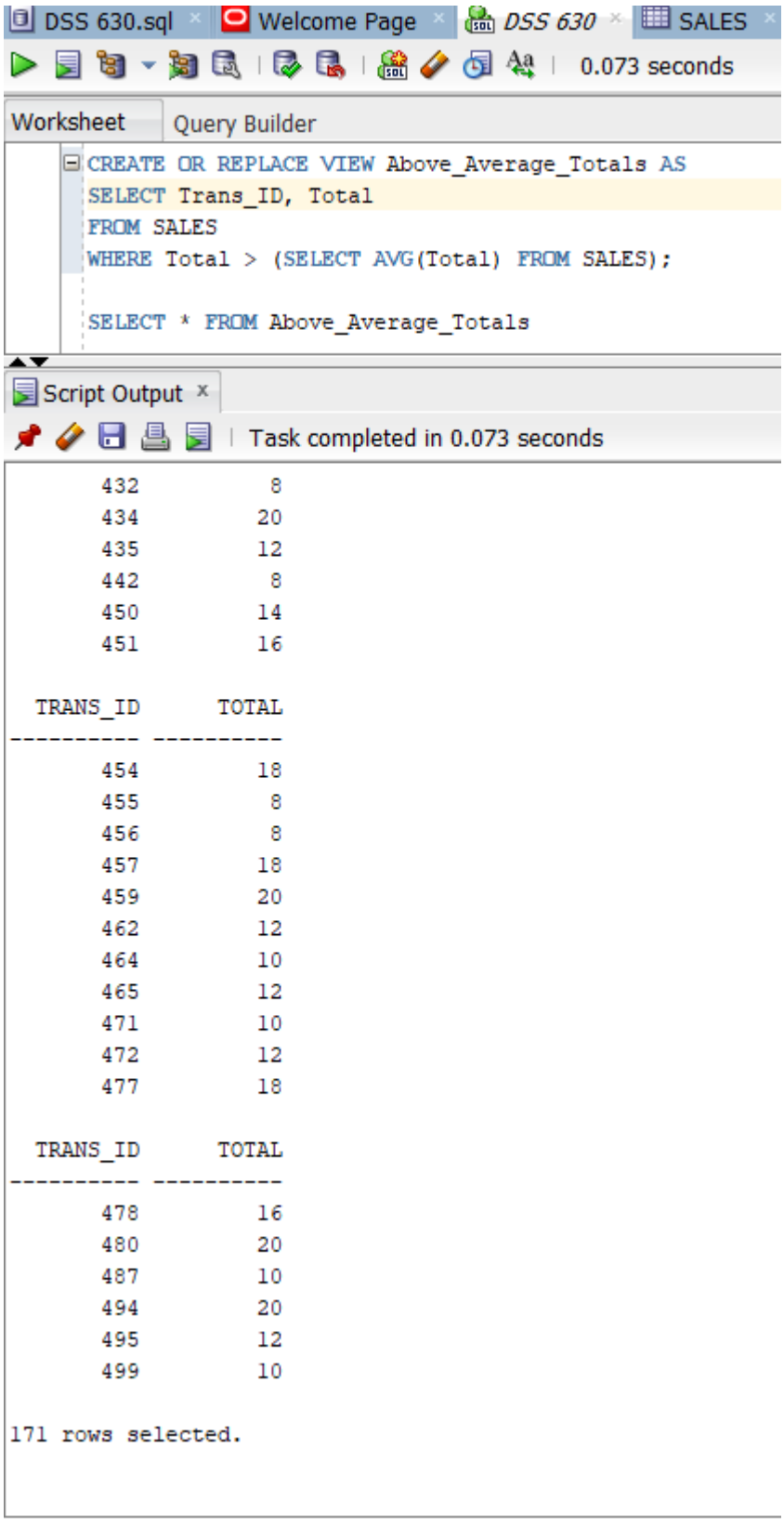
Appendix D2: Groupings



Appendix E1: Materialized Views



Appendix E2: Materialized Views



Appendix E3: Materialized Views