University of Pittsburgh
School of Computing and Information

INFSCI 2711: Advanced Database Management System
Spring 2020

Final Project Report

Submitted By:

Name	Email Address
Aishwarya Jakka	aij12@pitt.edu
Shruti Gupta	shg104@pitt.edu
Piu Mallick	pim16@pitt.edu
Reshma Sara Pothen	rep83@pitt.edu
Soham Bhatnagar	sob38@pitt.edu
Kenny Wu	kew143@pitt.edu
Kwesi Randolph Aguillera	kra40@pitt.edu

Contents

Online Retail Datawarehouse

Introduction

A data warehouse is a system that pulls together data from many different sources within an organization for reporting and analysis. In more comprehensive terms, a data warehouse is a consolidated view of either a physical or logical data repository collected from various systems. The primary focus of a data warehouse is to provide a correlation between data from existing systems, i.e., product inventory stored in one system purchase orders for a specific customer, stored in another system. Our project is based on the online retail dataset. It tries to analyze the data set and answer some of the basic questions that are needed for the market - for example, the performance of each product. For our analysis, we used three different types of databases, MySQL, Neo4j and MongoDB.

Sub-Teams

Teams responsible for exploring various areas in the project:

- **Coordination**
  - Aishwarya Jakka
- **Initial Data Cleaning**
  - Kwesi Randolph Aguillera
- **Further Data Cleaning**
  - Respective Teams
- **SQL-DB**
  - Shruti Gupta
  - Aishwarya Jakka
- **MongoDB**
  - Piu Mallick
  - Reshma Sara Pothen
- **Neo4j**
  - Soham Bhatnagar
  - Kenny Wu
  - Kwesi Randolph Aguillera
  - Aishwarya Jakka
- **Front-end & Back-end integration**
  - Kenny Wu
  - Kwesi Randolph Aguillera
- **Documentation**
  - Piu Mallick
  - Reshma Sara Pothen
  - Shruti Gupta

Dataset Description

This **Online Retail II** data set (the original dataset is a **excel file**) contains all the transactions occurring for a **UK-based** and registered, non-store online retail between **01/12/2009 (1st Dec 2009)** and **09/12/2011 (9th Dec 2011)**. The company mainly sells **unique all-occasion giftware**. Many customers of the company are wholesalers.

Attribute Information

The dataset has the following attributes:

- **InvoiceNo**: Invoice number. Nominal. A 6-digit integral number uniquely assigned to each transaction. If this code starts with the letter 'c', it indicates a cancellation.
- **StockCode**: Product (item) code. Nominal. A 5-digit integral number uniquely assigned to each distinct product.
- **Description**: Product (item) name. Nominal.
- **Quantity**: The quantities of each product (item) per transaction. Numeric.
- **InvoiceDate**: Invoice date and time. Numeric. The day and time when a transaction was generated.
- **UnitPrice**: Unit price. Numeric. Product price per unit in sterling (Â£).
- **CustomerID**: Customer number. Nominal. A 5-digit integral number uniquely assigned to each customer.
- **Country**: Country name. Nominal. The name of the country where a customer resides.

Github Link

The Github link for the project repository: Click here

Assumptions

- An item can have multiple descriptions for the same StockCode.
- We are assuming that a top product is the product that is most quantity purchased by the customer.
- We are assuming that the currency is the same across all the countries.

Initial Data-Cleaning

Following are the initial set of steps to clean the data:

- Removed records with blank CustomerIDs.
- Removed records weird StockCodes.

- Corrected Description where possible.
- Removed records where Description cannot be corrected.
- Removed zero Price items.
- Removed "Unspecified" country records.
- Changed Negative quantity records to positive quantity values.
- Separate Datetime into Date and Time columns.
- Calculate SubTotal for each record.

Following are the column stats for the year **2009-2010**:

- No. of records before cleaning: 525461
- No. of records after cleaning: 413084
- InvoiceNo Range: 489437 – 528618 / C489449 – C538168
- StockCode Range: 10002 – 90208 / 10123C – 90214Z & SP1002
- Quantity Range: 1 – 19,152
- Date Range: Dec 2009 – Dec 2010
- Time Range: 7:01 AM – 9:52 PM
- UnitPrice Range: $0.08 - $295.00
- CustomerID Range: 12346 – 18287
- No. of Countries: 40
- SubTotal Range: $0.06 - $15,818.40

Following are the column stats for the year **2010-2011**:

- No. of records before cleaning: 541910
- No. of records after cleaning: 300766
- InvoiceNo Range: 536365 – 564272 / C536383 – C564276
- StockCode Range: 10002 – 90208 / 10123C – 90214Z
- Quantity Range: 1 – 80,995
- Date Range: Dec 2010 – Dec 2001
- Time Range: 7:35 AM – 8:38 PM
- UnitPrice Range: $0.08 - $649.50
- CustomerID Range: 12347 – 18287
- No. of Countries: 40
- SubTotal Range: $0.06 - $168,469.60

The **initial data cleaning** was done manually in **Microsoft Excel**.
Post initial data cleaning, some extra cleaning was done in order to meet various database needs, which will be discussed later in this report.

Possible Aggregation Queries

Following are the probable statistical and aggregation queries that the owner/manager of the store may want to view in order to get the growth report of their store:

i. What time of the day (which hour of the day) is the sale maximum per country?

ii. What is the annual TotalSales per product?

iii. What is the top product per year?

iv. What is the top product per country?

v. Which item is sold below a certain threshold value? Or, what are the under-performed products based on the average sales last year (the year denotes either 2009 or 2010)?

vi. Which customer spends the most (per country/overall)?

vii. What is the best-selling month per country? (Given the year range, 2009-2011)

viii. What is the best-selling product per month? (Given the year range, 2009-2011)

ix. What is the change in TotalSales per country per year (Trend of Sales)?

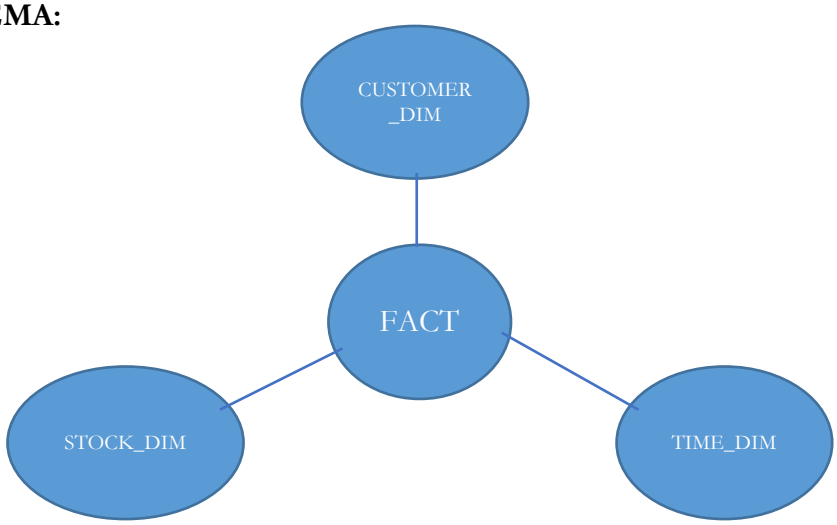
x. What is the average spending of a customer per country? (TotalSales/Number of customers)

xi. What is the frequently purchased item per customer?

STAR Schema: Design

Based on the queries (which we want to ask the database) stated above, we have designed the **DIMENSIONS** and **MEASURES**, which we popularly call as a **STAR schema**. Hence, first comes the aggregate queries, and based on the queries, we would like to build our aggregated/analytical database.

DIMENSIONS: Customer, Stock, Time
MEASURES: Quantity, Sales
STAR SCHEMA:



Hence, the **tables** (with all the **attributes**) we would be considering are as follows:

- **FACT** (*CustomerID, Year, Month, Day, Hour, Minute, StockCode, Quantity, TotalSales*)
- **CUSTOMER_DIM** (*CustomerID, Country*)
- **STOCK_DIM** (*StockCode, Description, UnitPrice*)
- **TIME_DIM** (*Year, Month, Day, Hour, Minute*)

However, the above table definitions may change depending on the databases we would be using.
Also, some tables may be excluded (not used) in some databases depending on the data retrieval demands.

Databases

We have chosen 3 databases, namely **MySQL, MongoDB** and **Neo4j** for building our **Online Retail Datawarehouse**. We would be integrating each of the databases with front-end, which we will discuss later in the document.

Please note:

1. All the data-warehouse design, implementation and execution are done on local instance(s). The project is not hosted on Cloud system.
2. Operational Database is loaded into both SQL and NoSQL databases.

MySQL DB

- **Creating the ODB (Operational Database)**
  - **Extra data-cleaning for MySQL DB:**
    Steps followed to clean the **Cleaned2009-2010.xlsx** and **Cleaned 2010-2011.xlsx.** (actual data files) from the data folder:
    - Renamed column "**Customer ID**" to "**Customer_ID**".
    - Changed format of the **InvoiceDate** from **MM/DD/YY** to **YYYY-MM-DD.**
    - Changed format of the **InvoiceTime** from **HH:MM AM/PM** to **HH:MM:SS** (**24 hour format**).
    - Changed format of the **SubTotal** to remove the "," formatting of numeric value.
    - Saved the file as "**online_retail_odb_2009_2010.xlsx**".
    - Then saved file as "**online_retail_odb_2009_2010.csv**".
    - The above steps are repeated for the other file (**2010-2011**) as well.

- **Steps followed to import the data in phpMyAdmin:**
  - Created **SQL** insert scripts.
  - Inserted the script in batches of **25000** as this was the limit for insertion.
  - The **ODB** creation process took a significant amount of time as import functionality didn't work with the csv (with phpMyAdmin).
  - Tried MySQL workbench to insert the script. All the insertions were done in one go.

- **Creating the ADB (Analytical Database)**
  - **Steps followed to create the ADB, relevant tables and furnish data**
    - Created a new database '**online_retail_adb**'.
    - Once the **Analytical Database** was created, we were ready to create the table specified in the schema – specifically, the 3 dimension tables – '**customerdim**', '**stockdim**' and '**timedim**' along with the '**fact**' table.
    - For each of these tables, we used the insert scripts to move data from the **ODB** to the relevant **ADB** table.
    - For the **FACT** table, we extracted the relevant fields from the **ODB**. Note that even though the **ODB** had '**SubTotal**' as a field, it was renamed '**TotalSales**' in the **FACT** table.
    - The '**customerdim**' table required the '**Customer_ID**' and '**Country**' fields. This was done using a '**groupby**' operator. To remove duplicates in **CustomerID**, **DISTINCT** operator was used. Finally, we used insertion scripts to insert data into '**customerdim'** table.
    - The '**stockdim'** table required the '**StockCode**', '**Description'** and '**Unit_Price**' fields. There was different unit price for single StockCode, so we took the maximum unit price. Insertion scripts were used to insert data into '**stockdim'** table.

- The '**timedim**' table required the '**Day**', '**Month**', '**Year**', '**Hour**' and '**Minute**' fields. Distinct rows were inserted from **ODB**.

- **Challenge Faced:**
  Since the database is too big, it took time to insert data.

- **Aggregation Queries**

  - **What time of the day is the sale maximum per country? (Query 1)**
    This query required three joins, making the query very expensive. It took around 5 minutes to run.
    In order to improve the run time, we created a materialized view named '**mview_sum_sale_country**'. Using the materialized view, run time decreased drastically.

    Sample Output:

hour	totalsale	Country
8	549.26	Czech Republic
8	2668.23	Israel
9	949.82	Korea
10	1143.60	Brazil
10	772.20	European Community
10	3777.92	Finland
10	6340.25	Greece
10	14613.56	Japan
10	1693.88	Lebanon
10	98203.17	Netherlands
10	18908.16	Spain
10	32646.31	Switzerland
10	1956.84	Thailand
11	488.21	Bahrain
11	10894.51	Channel Islands
11	20535.32	Denmark
11	52308.67	France
11	14.75	Saudi Arabia
11	3949.32	Singapore
11	536.41	West Indies
12	8072.75	Austria
12	9731.55	Belgium
12	9677.66	Cyprus
12	106500.20	EIRE
12	57966.78	Germany
12	4676.51	Lithuania
12	1002.31	RSA
12	13988.03	Sweden

- **Annual total sales of per product? (Query 2)**

  This query required three joins, making the query very expensive. It took around 7 minutes to run.

  In order to improve the run time, we created a materialized view named '**mview_totalsale_product**'. Using the materialized view, run time decreased drastically.

  Sample Output:

stockCode	Year	description	Totalsale
10002	2009	INFLATABLE POLITICAL GLOBE	181.05
10002	2010	INFLATABLE POLITICAL GLOBE	6941.27
10002	2011	INFLATABLE POLITICAL GLOBE	330.65
10080	2009	GROOVY CACTUS INFLATABLE	3.40
10080	2010	GROOVY CACTUS INFLATABLE	6.80
10080	2011	GROOVY CACTUS INFLATABLE	94.13
10109	2009	BENDY COLOUR PENCILS	1.68
10120	2009	DOGGY RUBBER	22.26
10120	2010	DOGGY RUBBER	83.58
10120	2011	DOGGY RUBBER	28.56
10123C	2009	HEARTS WRAPPING TAPE	78.18
10123C	2010	HEARTS WRAPPING TAPE	161.12
10123C	2011	HEARTS WRAPPING TAPE	0.65
10123G	2010	ARMY CAMO WRAPPING TAPE	223.42
10124A	2010	SPOTS ON RED BOOKCOVER ...	21.00
10124A	2011	SPOTS ON RED BOOKCOVER ...	3.78
10124G	2010	ARMY CAMO BOOKCOVER TAPE	8.40
10124G	2011	ARMY CAMO BOOKCOVER TAPE	5.04
10125	2009	MINI FUNKY DESIGN TAPES	105.40
10125	2010	MINI FUNKY DESIGN TAPES	594.05
10125	2011	MINI FUNKY DESIGN TAPES	727.65
10133	2009	COLOURING PENCILS BROWN...	130.05
10133	2010	COLOURING PENCILS BROWN...	720.70
10133	2011	COLOURING PENCILS BROWN...	781.02

- **Top product per year? (Query 3)**

  This query required three joins, making the query very expensive. It took around 5 minutes to run

  In order to improve the run time, we created a materialized view named '**mview_totalsale_product**'. Using the materialized view, run time decreased drastically.

  Sample Output:

stockCode	description	Year	totalSale
22423	REGENCY CAKESTAND 3 TIER	2010	164296.35
23843	PAPER CRAFT , LITTLE BIRDIE	2011	336939.20
85123A	WHITE HANGING HEART T-LIGHT HOLDER	2009	16637.66

- **Top product per country? (Query 4)**
  This query required three joins, making the query very expensive. It took around 5 minutes to run
  In order to improve the run time, we created a materialized view named '**mview_totalsale_product**'. Using the materialized view, run time decreased drastically.
  Sample Output:

stockCode	Country	Total
23843	United Kingdom	161990
37410	Denmark	25164
21088	France	14304
35961	Netherlands	6385
16033	Sweden	5760
84991	EIRE	5268
22328	Japan	3622
22670	Spain	2916
22326	Germany	2570
22492	Australia	1980
21731	Belgium	1296
51008	Channel Islands	900
22554	Switzerland	768
16008	Norway	576
84997D	Finland	568
37370	Canada	504
51014A	Italy	432
20725	Portugal	415
21355	USA	408
22047	Cyprus	400
21918	Austria	288
22339	Singapore	288
84568	Austria	288
84568	United Arab E...	288
84598	Austria	288
84598	United Arab E...	288
85017A	Thailand	288

- **Which item is sold below the threshold value? Or, what are the under-performed products based on the average sales last year (the year denotes either 2009 or 2010)? (Query 5)**
  For this query, threshold for a stock each year is devised by the average sale in the previous year. So, for any stock if there is sales record available for only one year, it is not considered. Also, if there are no record available for two consecutive years, the next available year is considered. For example, if for a stock we have data for the year of 2017 but no record for 2018 and then we have record for 2019, the records for 2019 is compared against 2017.
  This query required three joins, making the query very expensive. It took around 7 minutes to run
  In order to improve the run time, we created a materialized view named '**mview_quantity_product_year**'. Using the materialized view, run time decreased drastically.

Sample Output:

stockcode	description	difference
84255A	PINK GLASS COCKTAIL LAMP	0
90011B	BLACK CRYSTAL DROP EARRINGS	0
90111	BLUE HIBISCUS HAIR CLIP	0
90125B	AOUA BERTIE GLASS BEAD BAG CHARM	0
90141E	ORANGE PE DANT TRIPLE SHELL NECKLAC	0
90177E	DROP DIAMANTE EARRINGS GREEN	0
35980B	RED TOP SCANDINAVIAN HEART	0
46775A	BUBBLE GUM CHUNKY KNITTED THROW	0
79340P	PURPLE ORCHID FLOWER LIGHTS	0
79413	GLASS WINE GLASS DECORATIONS	0
21496	WOODLAND CREATURES WRAP	0
84206C	CHAMPAGNE TRAY BLANK CARD	0
90034	WHITE SILVER NECKLACE SHELL GLASS	0
90043	COPPER AND BRASS BAG CHARM	0
90060D	FIRE POLISHED GLASS NECKL BRONZE	0
90067B	BROWN VINTAGE VICTORIAN EARRINGS	0
90134	OLD ROSE COMBO BEAD NECKLACE	0
84617	NEW BAROQUE BLACK BOXES	0
90168	2 DAISIES HAIR COMB	0
90177D	DROP DIAMANTE EARRINGS PURPLE	0
21081	SET/20 POSIES PAPER NAPKINS	0
47569	ENGLISH ROSE DESIGN SHOPPING BAG	0
72708	SCULPTED ROUND IVORY CANDLE	0
90131	PINK/AMETHYST/GOLD NECKLACE	1
90139	FLOWER BROOCH 4 ASSORTED COLOURS	1

- **Which customer spends the most per country? (query 6)**
  This query required three joins, making the query very expensive. It took around 5 minutes to run.
  In order to improve the run time, we created a materialized view named '**mview_customer_total_country'**. Using the materialized view, run time decreased drastically.

Sample Output:

customer_id	totalsale	Country
12347	3223.59	Iceland
12355	818.01	Bahrain
12359	7318.08	Cyprus
12409	23736.11	Switzerland
12415	120873.87	Australia
12429	5841.81	Austria
12431	10136.67	Belgium
12433	18466.34	Norway
12446	1002.31	RSA
12454	28820.60	Spain
12469	3070.54	Thailand
12471	33080.28	Germany
12565	14.75	Saudi Arabia
12594	4125.21	Italy
12607	3159.02	USA
12653	4160.83	Israel
12664	4568.46	Finland
12744	10040.16	Singapore
12753	16153.27	Japan
12758	10002.28	Portugal
12764	1693.88	Lebanon
12767	949.82	Korea
12769	1143.60	Brazil
12779	5236.27	Poland
12781	901.76	Czech Rep...
13902	37671.26	Denmark
14156	268978.91	EIRE

- **Best-selling month (per country) [2009-2011] (Query 7)**
  This query required three joins, making the query very expensive. It took around 5 minutes to run.
  In order to improve the run time, we created a materialized view named '**mview_sale_month**'. Using the materialized view, run time decreased drastically.

Sample Output:

month	totalsale	Country
1	77551.86	EIRE
1	4798.35	Greece
1	1693.88	Lebanon
2	549.26	Czech Republic
2	23803.92	Denmark
2	2199.54	United Arab Emirates
3	14.75	Saudi Arabia
4	1143.60	Brazil
4	16288.58	Sweden
5	907.01	Bahrain
6	26982.41	Australia
6	1956.84	Thailand
7	616.80	European Community
7	3272.96	Finland
7	3381.73	Israel
7	3949.32	Singapore
8	2047.61	Malta
8	112.57	Nigeria
8	536.41	West Indies
9	8055.54	Channel Islands
10	834.89	Canada
10	7356.04	Italy
10	1002.31	RSA

- **Best-selling product per month [2009-2011] (query 8)**
  This query required three joins, making the query very expensive. It took around 5 minutes to run.
  In order to improve the run time, we created a materialized view named '**mview_quantity_product'**. Using the materialized view, run time decreased drastically.

Sample Output:

stockCode	description	maxtotal	month
20993	JAZZ HEARTS MEMO PAD	9489	1
37410	BLACK AND WHITE PAISLEY FLOWER MUG	19248	2
21099	SET/6 STRAWBERRY PAPER CUPS	13105	3
84077	WORLD WAR 2 GLIDERS ASSTD DESIGNS	13536	4
84077	WORLD WAR 2 GLIDERS ASSTD DESIGNS	8704	5
85099B	JUMBO BAG RED RETROSPOT	7058	6
85123A	WHITE HANGING HEART T-LIGHT HOLDER	6639	7
84879	ASSORTED COLOUR BIRD ORNAMENT	8886	8
17003	BROCADE RING PURSE	15826	9
84077	WORLD WAR 2 GLIDERS ASSTD DESIGNS	13063	10
84879	ASSORTED COLOUR BIRD ORNAMENT	14246	11
23843	PAPER CRAFT , LITTLE BIRDIE	161990	12

- **What is the change in total sales per country per year (trend of sales)? (Query 9)**
  This query required three joins, making the query very expensive. It took around 5 minutes to run.
  In order to improve the run time, we created a materialized view named '**mview_yearsale_country'**. Using the materialized view, run time decreased drastically.

  Sample Output:

sub_2010_2009	sub_2011_2010	country
33326.31	77122.37	Australia
13166.53	-4996.61	Austria
25604.90	8427.82	Belgium
24055.03	-10011.70	Channel Islands
7018.56	-157.47	Cyprus
51966.04	-34140.81	Denmark
313425.22	-144514.91	EIRE
6569.26	4319.55	Finland
138405.70	-6843.18	France
179877.70	-39303.31	Germany
12954.92	-11801.59	Greece
13457.91	-3219.70	Italy
14693.66	14470.24	Japan
247607.58	-69847.55	Netherlands
12850.94	9562.35	Norway
3225.37	281.61	Poland
18187.44	-2883.41	Portugal
40264.75	-2795.69	Spain
53673.45	-29869.78	Sweden
42139.06	4690.77	Switzerland
6024.07	-4715.15	United Arab E...
6584255.08	-1878346.37	United Kingdom

- **Average spending of a customer in that country (Query 10)**
  This query required both '**CustomerID**' and '**Country**' from the '**Customerdim**' table. The data was grouped by '**Country**' and the average of '**TotalSales**' was calculated. The results were then sorted and displayed.
  This query required three joins, making the query very expensive. It took around 5 minutes to run.
  In order to improve the run time, we created a materialized view named '**mview_customer_total_country**'. Using the materialized view, run time decreased drastically.

Sample Output:

Avg_Spending	country
0.0001	Australia
0.0005	Austria
0.0016	Bahrain
0.0005	Belgium
0.0014	Brazil
0.0020	Canada
0.0003	Channel Islands
0.0004	Cyprus
0.0011	Czech Republic
0.0002	Denmark
0.0000	EIRE
0.0009	European Com...
0.0006	Finland
0.0003	France
0.0003	Germany
0.0003	Greece
0.0003	Iceland
0.0005	Israel
0.0006	Italy
0.0002	Japan
0.0016	Korea
0.0006	Lebanon
0.0002	Lithuania
0.0006	Malta
0.0000	Netherlands
0.0089	Nigeria
0.0003	Norway
0.0008	Poland

- **Frequently purchased item per customer? (Query 11)**
  This query used same table, so materialized view was not required.

Sample Output:

Customer_ID	stockCode	stockcount
17850	82494L	125
14911	22423	85
17841	21935	68
13089	84949	48
15005	21232	47
13767	21314	45
17377	21622	45
14680	20725	43
14606	20749	40
15039	71053	40
12471	21232	38
15311	15056BL	38
15311	15056N	38
12748	21034	37
17243	22112	35
17675	85123A	34
12681	21731	33
15078	21931	33
17511	85099B	32
17511	85123A	32
16422	20974	31
16422	22090	31
16549	85123A	31

- **Materialized queries:**

  For materialized queries, we first created the relevant table, i.e. table used for optimizing the query, and then inserted the data into the table for further usage.

  - '**mview_sum_sale_country**': This materialized query was used to get the **TotalSales per Hour per Country**. This query is further used in query for finding 'What time of the day is the sale maximum per country?'

  - '**mview_totalsale_product**': This materialized query was used to get the **TotalSales per StockCode per Year** and the respective description of **StockCode**.

  - '**mview_quantity_product_year**': This materialized query was used to get the **TotalQuantity per StockCode per Year** with respective description of **StockCode**.

  - '**mview_customer_total_country**': This materialized query was used to get the **TotalSales per Customer_ID per Country**.

  - '**mview_sale_month**': This materialized query was used to get the **TotalSales per month per country.**

- '**mview_quantity_product**': This materialized query was used to get the **TotalQuantity per StockCode per Month** with description based on StockCode.
- '**mview_yearsale_country'**: This materialized query was used to get the **TotalSales per Year per Country**.
- '**mview_product_country**': This materialized query was used to get the **TotalQuantity per Country per StockCode**.

- **Advantages of MySQL:**

  - MySQL can store a relational schema of this database. There are many internal functions in MySQL. Relational model is easier to understand, hierarchy and other models.
  - The common SQL language makes it very convenient to operate a relational database. Abundant integrity (entity integrity, referential integrity, and user-defined integrity) greatly reduces the probability of data redundancy and data inconsistency.
  - All the aggregation queries (as designed to query the Online Retail Datawarehouse) are executable in MySQL.
  - It is possible to run every type of query (as stated above) by applying multiple joins.

- **Disadvantages of MySQL:**

  - It is not skillful to write large data into database, and it will increase overhead cost when executing multi-table association queries, especially for such a large database.
  - Execution time of the queries are very long.
  - MySQL is expensive when multiple joins are used.

MongoDB

- **Creating the ODB (Operational Database)**

  - **Extra data-cleaning for MongoDB:**

    Steps followed to clean the **Cleaned2009-2010.xlsx** and **Cleaned 2010-2011.xlsx.** (actual data files) from the **data** folder:

    - Changed the **file type** to **csv**.
    - Loaded the **csv files** to **Jupyter Notebook** (**Python**) and converted them to dataframes.
    - Concatenated two dataframes using the '**append'** function.
    - The attribute '**Customer ID**' is renames to '**CustomerID**'.
    - The attributes '**InvoiveDate**' and '**InvoiceTime**' are then concatenated and renamed to '**InvoiceDateTime**'. Later, they are split to **Year** (**YYYY**), **Month** (**MM**), **Day** (**DD**), **Hour** (**HH**) and **Minute** (**MM**).
    - The dataframe is then exported to the data folder as '**Online_Retail_DB.csv**'.

  - **Steps followed to import the csv file to MongoDB:**
    - Go to the **terminal** and login to **MongoDB** (by typing '**mongo**' to stay in the same terminal or '**mongod**' to switch to a new terminal).
      Pre-requisite: Prior installation of MongoDB in the system and the **mongodb service** to be running.
    - Switch to or create a new database – '**odb**' in MongoDB with the help of the following command in the terminal:
    - In a separate terminal, navigate to the path where the cleaned data file is present. The **MongoDB import command** is executed outside the mongo shell - in a normal terminal, with the help of the following command.
    - At the end of this process, the relevant documents were successfully inserted. This is shown below.

```
C:\Program Files\MongoDB\Server\4.2\bin>mongoimport --db odb --collection retail --type csv --headerline --file Online_Retail_Data.csv
2020-04-09T17:44:45.157-0400    connected to: mongodb://localhost/
2020-04-09T17:44:48.157-0400    [###.....................] odb.retail   10.9MB/79.0MB (13.8%)
2020-04-09T17:44:51.159-0400    [######..................] odb.retail   21.8MB/79.0MB (27.6%)
2020-04-09T17:44:54.158-0400    [#########...............] odb.retail   32.8MB/79.0MB (41.5%)
2020-04-09T17:44:57.158-0400    [############............] odb.retail   43.9MB/79.0MB (55.5%)
2020-04-09T17:45:00.159-0400    [###############........] odb.retail   55.0MB/79.0MB (69.5%)
2020-04-09T17:45:03.158-0400    [##################.....] odb.retail   65.9MB/79.0MB (83.4%)
2020-04-09T17:45:06.158-0400    [#####################..] odb.retail   77.3MB/79.0MB (97.8%)
2020-04-09T17:45:06.655-0400    [########################] odb.retail   79.0MB/79.0MB (100.0%)
2020-04-09T17:45:06.655-0400    713850 document(s) imported successfully. 0 document(s) failed to import.
```

- **Creating the ADB (Analytical Database)**

  - **Steps followed to create the ADB, relevant tables and furnish data**
    - Created a new database '**adb**' in **MongoDB**.

Specified '**adb**' as the current database using the following command: `use adb`

- Once the **Analytical Database** was created, we were ready to create the table specified in the schema – specifically, **3-dimension** tables (called **collections** in **MongoDB**) - **customer_dim**, **stock_dim** and **time_dim** along with the **FACT** table.
- For each of these collections, we used the **.getSiblingDB** function to move data from the **ODB** to the relevant **ADB** table.
- For the **FACT** collection:
  We extracted the relevant fields from the **ODB**. Note that even though the ODB had '**SubTotal**' as a field, it was renamed '**TotalSales**' in the **FACT** table. The '**forEach**' function is used to loop through the retail collection for all records.
- For the **customer_dim** collection:
  The **customer_dim** collection required the '**CustomerID**' and '**Country**' fields. This was done using a '**$group**' operator. To remove duplicates in CustomerID, the '**$match**' operator was used. Finally, as with the **FACT** table data insertion, '**forEach**' was used to loop through the retail collection in the **ODB** and insert every record.
- For the **stock_dim** collection:
  The **stock_dim** collection required the '**StockCode**', '**Description**' and '**MaxUnitPrice**' fields. Initially, an index was created for '**StockCode**' to allow faster query execution. This was followed by an aggregation query on the retail collection. Note that **Price** was changed to **MaxUnitPrice**.
- For the **time_dim** collection:
  The **time_dim** collection necessitated a two-step approach to insertion. Initially, the data for the '**Day**', '**Month**', '**Year**', '**Hour**' and '**Minute**' fields were fetched, using the **forEach** function, from the retail collection and inserted into a temporary table '**time_dim_temp**'. Once done, the data was aggregated into '**Day**', '**Month**', '**Year**', '**Hour**' and '**Minute**' into the final '**time_dim**' collection.

At the end of this process, all four collections were populated. The relevant statistics are displayed below.

```
MongoDB Enterprise > db.fact.count()
713850
MongoDB Enterprise > db.customer_dim.count()
5723
MongoDB Enterprise > db.stock_dim.count()
4630
MongoDB Enterprise > db.time_dim.count()
36297
MongoDB Enterprise >
```

Please note: The **time_dim** collection was excluded (not deleted, only not used) for the **STAR schema** of the **Mongo DB** data-warehouse. The time related data (required in the aggregation queries) was fetched from the **FACT** collection itself for faster data retrieval. Joining data (using **$lookup** function) is quite expensive in **MongoDB**.

1.1.1 Aggregation Queries

- **What time of the day is the sale maximum per country? (Query 1)**

We performed this query both with and without an index. Without index, this was the longest running query – taking a **whopping 16 minutes**. With an index, it took a much more reasonable **45 seconds**.

Sample Output Data for Query 1 (when executed from MongoDB terminal)


```
{ "_id" : "Australia", "MaxSalePerHourPerCountry" : 32760.46, "Country" : "Australia", "Hour" : 11 }
{ "_id" : "Austria", "MaxSalePerHourPerCountry" : 8072.75, "Country" : "Austria", "Hour" : 11 }
{ "_id" : "Bahrain", "MaxSalePerHourPerCountry" : 488.21, "Country" : "Bahrain", "Hour" : 17 }
{ "_id" : "Belgium", "MaxSalePerHourPerCountry" : 9731.55, "Country" : "Belgium", "Hour" : 13 }
{ "_id" : "Brazil", "MaxSalePerHourPerCountry" : 1143.6, "Country" : "Brazil", "Hour" : 12 }
{ "_id" : "Canada", "MaxSalePerHourPerCountry" : 834.89, "Country" : "Canada", "Hour" : 9 }
{ "_id" : "Channel Islands", "MaxSalePerHourPerCountry" : 10894.51, "Country" : "Channel Islands", "Hour" : 17 }
{ "_id" : "Cyprus", "MaxSalePerHourPerCountry" : 9677.66, "Country" : "Cyprus", "Hour" : 16 }
{ "_id" : "Czech Republic", "MaxSalePerHourPerCountry" : 549.26, "Country" : "Czech Republic", "Hour" : 9 }
{ "_id" : "Denmark", "MaxSalePerHourPerCountry" : 13984.92, "Country" : "Denmark", "Hour" : 18 }
{ "_id" : "EIRE", "MaxSalePerHourPerCountry" : 97638.1, "Country" : "EIRE", "Hour" : 7 }
{ "_id" : "European Community", "MaxSalePerHourPerCountry" : 772.2, "Country" : "European Community", "Hour" : 12 }
{ "_id" : "Finland", "MaxSalePerHourPerCountry" : 3777.92, "Country" : "Finland", "Hour" : 13 }
{ "_id" : "France", "MaxSalePerHourPerCountry" : 50655.55, "Country" : "France", "Hour" : 14 }
{ "_id" : "Germany", "MaxSalePerHourPerCountry" : 57966.78, "Country" : "Germany", "Hour" : 13 }
{ "_id" : "Greece", "MaxSalePerHourPerCountry" : 6340.25, "Country" : "Greece", "Hour" : 9 }
{ "_id" : "Iceland", "MaxSalePerHourPerCountry" : 1777.61, "Country" : "Iceland", "Hour" : 15 }
{ "_id" : "Israel", "MaxSalePerHourPerCountry" : 2668.23, "Country" : "Israel", "Hour" : 10 }
{ "_id" : "Italy", "MaxSalePerHourPerCountry" : 5702.94, "Country" : "Italy", "Hour" : 19 }
{ "_id" : "Japan", "MaxSalePerHourPerCountry" : 9672.210000000001, "Country" : "Japan", "Hour" : 11 }
```


- **Annual Total Sales Per Product? (Query 2)**

This query required a grouping of '**Year**', '**Stock**' and sum of '**TotalSales**' to produce '**TotalAnnualSales**'. Once the result set was obtained, '**$sort**' was used to sort both '**Stock**' and '**Year**' in an ascending order. This took **a very minimal 2 seconds to execute**.

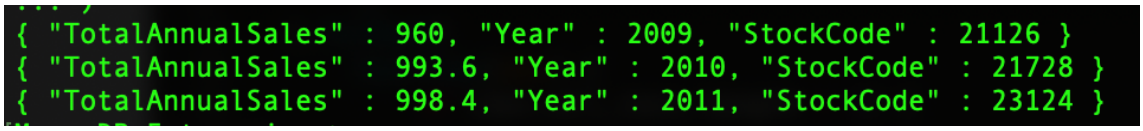
Sample Output Data for Query 2 (when executed from MongoDB terminal)


```
{ "TotalAnnualSales" : 181.04999999999998, "Year" : 2009, "Stock" : 10002, "Description" : "INFLATABLE POLITICAL GLOBE" }
{ "TotalAnnualSales" : 6941.2699999999995, "Year" : 2010, "Stock" : 10002, "Description" : "INFLATABLE POLITICAL GLOBE" }
{ "TotalAnnualSales" : 330.65, "Year" : 2011, "Stock" : 10002, "Description" : "INFLATABLE POLITICAL GLOBE" }
{ "TotalAnnualSales" : 3.4, "Year" : 2009, "Stock" : 10080, "Description" : "GROOVY CACTUS INFLATABLE" }
{ "TotalAnnualSales" : 6.8, "Year" : 2010, "Stock" : 10080, "Description" : "GROOVY CACTUS INFLATABLE" }
{ "TotalAnnualSales" : 94.13, "Year" : 2011, "Stock" : 10080, "Description" : "GROOVY CACTUS INFLATABLE" }
{ "TotalAnnualSales" : 1.68, "Year" : 2009, "Stock" : 10109, "Description" : "BENDY COLOUR PENCILS" }
{ "TotalAnnualSales" : 22.259999999999998, "Year" : 2009, "Stock" : 10120, "Description" : "DOGGY RUBBER" }
{ "TotalAnnualSales" : 83.58, "Year" : 2010, "Stock" : 10120, "Description" : "DOGGY RUBBER" }
{ "TotalAnnualSales" : 28.56, "Year" : 2011, "Stock" : 10120, "Description" : "DOGGY RUBBER" }
{ "TotalAnnualSales" : 105.4, "Year" : 2009, "Stock" : 10125, "Description" : "MINI FUNKY DESIGN TAPES" }
{ "TotalAnnualSales" : 594.05, "Year" : 2010, "Stock" : 10125, "Description" : "MINI FUNKY DESIGN TAPES" }
{ "TotalAnnualSales" : 727.65, "Year" : 2011, "Stock" : 10125, "Description" : "MINI FUNKY DESIGN TAPES" }
{ "TotalAnnualSales" : 130.05, "Year" : 2009, "Stock" : 10133, "Description" : "COLOURING PENCILS BROWN TUBE" }
{ "TotalAnnualSales" : 720.7, "Year" : 2010, "Stock" : 10133, "Description" : "COLOURING PENCILS BROWN TUBE" }
{ "TotalAnnualSales" : 781.02, "Year" : 2011, "Stock" : 10133, "Description" : "COLOURING PENCILS BROWN TUBE" }
{ "TotalAnnualSales" : 52.5, "Year" : 2009, "Stock" : 10134, "Description" : "COLOURING PENCILS BROWN TUBE" }
{ "TotalAnnualSales" : 663.55, "Year" : 2010, "Stock" : 10134, "Description" : "COLOURING PENCILS BROWN TUBE" }
{ "TotalAnnualSales" : 133.75, "Year" : 2009, "Stock" : 10135, "Description" : "COLOURING PENCILS BROWN TUBE" }
{ "TotalAnnualSales" : 2294.55, "Year" : 2010, "Stock" : 10135, "Description" : "COLOURING PENCILS BROWN TUBE" }
```


- **Top Product Per Year? (Query 3)**

This query required a grouping of '**Year**', '**Stock**' and sum of '**TotalSales**' to produce '**TotalAnnualSales**'. The results were then sorted by '**Year**'. Execution time was once again, **minimal at 3 seconds**.

Sample Output Data for Query 3 (when executed from MongoDB terminal)



```
{ "TotalAnnualSales" : 960, "Year" : 2009, "StockCode" : 21126 }
{ "TotalAnnualSales" : 993.6, "Year" : 2010, "StockCode" : 21728 }
{ "TotalAnnualSales" : 998.4, "Year" : 2011, "StockCode" : 23124 }
```

- **Top Product Per Country? (Query 4)**

This query required a grouping of '**Country**', '**Stock**' and sum of '**TotalSales**' to produce '**TotalAnnualSales**'. The results were then sorted by '**Country**'. Execution time was **slightly longer at 52 seconds**.

Sample Output Data for Query 4 (when executed from MongoDB terminal)

```
{ "TotalSalePerStock" : 176955.78999999998, "StockCode" : 10002, "Country" : "United Kingdom" }
{ "TotalSalePerStock" : 13398, "StockCode" : 10002, "Country" : "Germany" }
{ "TotalSalePerStock" : 11859.75, "StockCode" : 10002, "Country" : "EIRE" }
{ "TotalSalePerStock" : 8873.3, "StockCode" : 10002, "Country" : "Netherlands" }
{ "TotalSalePerStock" : 5068.88, "StockCode" : 10002, "Country" : "France" }
{ "TotalSalePerStock" : 4406.4, "StockCode" : 10002, "Country" : "Spain" }
{ "TotalSalePerStock" : 2655, "StockCode" : 16235, "Country" : "Channel Islands" }
{ "TotalSalePerStock" : 2464.2000000000003, "StockCode" : 10002, "Country" : "Australia" }
{ "TotalSalePerStock" : 2094.16, "StockCode" : 20665, "Country" : "Finland" }
{ "TotalSalePerStock" : 2001, "StockCode" : 10002, "Country" : "Sweden" }
{ "TotalSalePerStock" : 1994.4, "StockCode" : 10002, "Country" : "Belgium" }
{ "TotalSalePerStock" : 1828.8, "StockCode" : 10002, "Country" : "Switzerland" }
{ "TotalSalePerStock" : 993.6, "StockCode" : 10002, "Country" : "Denmark" }
{ "TotalSalePerStock" : 949.65, "StockCode" : 10133, "Country" : "Cyprus" }
{ "TotalSalePerStock" : 927, "StockCode" : 20681, "Country" : "Singapore" }
{ "TotalSalePerStock" : 789.75, "StockCode" : 10002, "Country" : "Japan" }
{ "TotalSalePerStock" : 700.8, "StockCode" : 10133, "Country" : "Israel" }
{ "TotalSalePerStock" : 664.75, "StockCode" : 10133, "Country" : "Portugal" }
{ "TotalSalePerStock" : 654, "StockCode" : 11001, "Country" : "Austria" }
{ "TotalSalePerStock" : 591.9, "StockCode" : 10125, "Country" : "Norway" }
```

- **Which item is sold below the threshold value? Or, what are the under-performed products based on the average sales last year (the year denotes either 2009 or 2010)? (Query 5)**

For this query, threshold for a stock in a given year is devised by the average sale in the previous year. So, for any stock if there is sales record available for only one year, it is not considered. Also, if there are no record available for two consecutive years, the next available year is considered. For example, if for a stock we have data for the year of 2017 but no record for 2018 and then we have record for 2019, the records for 2019 is compared against 2017.

This query needed a total of three aggregation pipeline stages in MongoDB all run against the fact table. First is to do a grouping based on StockCode and Year to find out the total Quantity sold per stock per year and the total number of sales happened per year per stock. For faster result we can create an index based on StockCode and Year. In the next stage we do a sort based on StockCode and Year. In the third and final stage we do a group based on StockCode to get, for each stock, the total quantity of a stock sold per year and total number of sales for that stock in that year. We get the result of per year sales record of a stock as an array sorted by year under that stock.

The rest of the logic is done in JavaScript while iterating the resultant cursor, i.e. to find out the average sale for the previous year and if the current year's average exceeds that average or not. The execution time of this query was approximately **3 seconds**.

Sample Output Data for Query 5 (when executed from MongoDB terminal)

```
for StockCode:21265 previous year: 2009's average: 8.333333333333334 and current year 2010's average is 15.166666666666666
for StockCode:21265 previous year: 2009's average: 15.166666666666666 and current year 2011's average is 7.636363636363637
for StockCode:21839 previous year: 2009's average: 2.1666666666666665 and current year 2010's average is 2.4444444444444446
for StockCode:21839 previous year: 2009's average: 2.4444444444444446 and current year 2011's average is 1
for StockCode:22413 previous year: 2010's average: 7.105960264900662 and current year 2011's average is 12.04149377593361
for StockCode:22987 previous year: 2010's average: 25 and current year 2011's average is 25
for StockCode:85165 previous year: 2009's average: 2 and current year 2010's average is 1.4
for StockCode:84526 previous year: 2009's average: 4.25 and current year 2010's average is 2.4814814814814814
for StockCode:90038C previous year: 2010's average: 3.5 and current year 2011's average is 6
for StockCode:62096A previous year: 2010's average: 5.326530612244898 and current year 2011's average is 17.6
for StockCode:90099 previous year: 2009's average: 1 and current year 2010's average is 2.6666666666666665
for StockCode:90099 previous year: 2009's average: 2.6666666666666665 and current year 2011's average is 1.5833333333333333
for StockCode:22582 previous year: 2010's average: 7.683417085427136 and current year 2011's average is 18.361702127659573
for StockCode:37485 previous year: 2009's average: 3.4166666666666665 and current year 2010's average is 2
```

- **Which customer spends the most (per country or overall) (Query 6)**

This is a **two-part query**. We first attempted to find out which customer spends most overall. This query required a grouping of '**CustomerID**' and '**TotalSales**'. An interesting aspect of this query is that initially, the execution, after running for over 10 minutes, produced a '**TimeOut Error**'. We remedied this by setting the '**allowDiskUse**' **flag** to **true**. The execution took **8 seconds**. For the second part, we grouped together '**CustomerID**' and '**TotalSales**' along with '**Country**'. Per the learnings of the first part, we left the '**allowDiskUse**' flag at true. The execution took **7 seconds**.

Sample Output Data for Query 6 (overall) (when executed from MongoDB terminal)

```
{ "Sale" : 425317.46, "CustomerID" : 14646 }
MongoDB Enterprise >
```

Sample Output Data for Query 6 (per country) (when executed from MongoDB terminal)

```
{ "TotalSalePerCustomer" : 425317.46, "CustomerID" : 14646, "Country" : "Netherlands" }
{ "TotalSalePerCustomer" : 269835.11, "CustomerID" : 18102, "Country" : "United Kingdom" }
{ "TotalSalePerCustomer" : 254087.63999999998, "CustomerID" : 14911, "Country" : "EIRE" }
{ "TotalSalePerCustomer" : 246199.81, "CustomerID" : 14156, "Country" : "EIRE" }
{ "TotalSalePerCustomer" : 175603.81, "CustomerID" : 13694, "Country" : "United Kingdom" }
{ "TotalSalePerCustomer" : 153471.11000000002, "CustomerID" : 17511, "Country" : "United Kingdom" }
{ "TotalSalePerCustomer" : 115836.47, "CustomerID" : 12415, "Country" : "Australia" }
{ "TotalSalePerCustomer" : 110703.35, "CustomerID" : 15061, "Country" : "United Kingdom" }
{ "TotalSalePerCustomer" : 108992.53, "CustomerID" : 16684, "Country" : "United Kingdom" }
{ "TotalSalePerCustomer" : 108836.25, "CustomerID" : 13089, "Country" : "United Kingdom" }
{ "TotalSalePerCustomer" : 105906.42, "CustomerID" : 15311, "Country" : "United Kingdom" }
{ "TotalSalePerCustomer" : 85950.87, "CustomerID" : 12931, "Country" : "United Kingdom" }
{ "TotalSalePerCustomer" : 83420.09999999999, "CustomerID" : 14298, "Country" : "United Kingdom" }
{ "TotalSalePerCustomer" : 74347.95, "CustomerID" : 17450, "Country" : "United Kingdom" }
{ "TotalSalePerCustomer" : 74223.01, "CustomerID" : 16029, "Country" : "United Kingdom" }
{ "TotalSalePerCustomer" : 61730, "CustomerID" : 15769, "Country" : "United Kingdom" }
{ "TotalSalePerCustomer" : 60472.469999999994, "CustomerID" : 13798, "Country" : "United Kingdom" }
{ "TotalSalePerCustomer" : 59590.15, "CustomerID" : 17841, "Country" : "United Kingdom" }
{ "TotalSalePerCustomer" : 56303.71, "CustomerID" : 16422, "Country" : "United Kingdom" }
{ "TotalSalePerCustomer" : 55547.48, "CustomerID" : 17850, "Country" : "United Kingdom" }
```

- **Best-selling month (per country) [2009-2011] (Query 7)**

This query utilizes the **'$unwind**' operator that allows us to examine each record in turn. Using that, we were able to extract the '**Country'** values and group by '**Country**', '**Month**' and sum of '**TotalSales**'. This data was then grouped separately to identify '**MaxSalePerHourPerCountry**'. Execution, due to sheer volume of calculation, was a bit **longer at 50 seconds**.

Sample Output Data for Query 7 (when executed from MongoDB terminal)


```
{ "_id" : "Australia", "MaxSalePerHourPerCountry" : 26982.41, "Country" : "Australia", "Month" : 3 }
{ "_id" : "Austria", "MaxSalePerHourPerCountry" : 5327.06, "Country" : "Austria", "Month" : 2 }
{ "_id" : "Bahrain", "MaxSalePerHourPerCountry" : 907.01, "Country" : "Bahrain", "Month" : 5 }
{ "_id" : "Belgium", "MaxSalePerHourPerCountry" : 10312.34, "Country" : "Belgium", "Month" : 2 }
{ "_id" : "Brazil", "MaxSalePerHourPerCountry" : 1143.6, "Country" : "Brazil", "Month" : 9 }
{ "_id" : "Canada", "MaxSalePerHourPerCountry" : 834.89, "Country" : "Canada", "Month" : 10 }
{ "_id" : "Channel Islands", "MaxSalePerHourPerCountry" : 8055.54, "Country" : "Channel Islands", "Month" : 7 }
{ "_id" : "Cyprus", "MaxSalePerHourPerCountry" : 5528.15, "Country" : "Cyprus", "Month" : 7 }
{ "_id" : "Czech Republic", "MaxSalePerHourPerCountry" : 549.26, "Country" : "Czech Republic", "Month" : 2 }
{ "_id" : "Denmark", "MaxSalePerHourPerCountry" : 10785.92, "Country" : "Denmark", "Month" : 1 }
{ "_id" : "EIRE", "MaxSalePerHourPerCountry" : 73913.08, "Country" : "EIRE", "Month" : 8 }
{ "_id" : "European Community", "MaxSalePerHourPerCountry" : 616.8, "Country" : "European Community", "Month" : 6 }
{ "_id" : "Finland", "MaxSalePerHourPerCountry" : 3272.96, "Country" : "Finland", "Month" : 3 }
{ "_id" : "France", "MaxSalePerHourPerCountry" : 44161.13, "Country" : "France", "Month" : 3 }
{ "_id" : "Germany", "MaxSalePerHourPerCountry" : 47467.78, "Country" : "Germany", "Month" : 2 }
{ "_id" : "Greece", "MaxSalePerHourPerCountry" : 4798.35, "Country" : "Greece", "Month" : 7 }
{ "_id" : "Iceland", "MaxSalePerHourPerCountry" : 936.61, "Country" : "Iceland", "Month" : 4 }
{ "_id" : "Israel", "MaxSalePerHourPerCountry" : 3381.73, "Country" : "Israel", "Month" : 10 }
{ "_id" : "Italy", "MaxSalePerHourPerCountry" : 7356.04, "Country" : "Italy", "Month" : 4 }
{ "_id" : "Japan", "MaxSalePerHourPerCountry" : 6551.25, "Country" : "Japan", "Month" : 11 }
```


- **Best-selling product per month [2009-2011] (Query 8)**

This query required a grouping of '**Year**', '**Stock**' and sum of '**TotalSales**' to produce '**TotalAnnualSales**'. The results were then sorted by '**Year'**. Execution time was once again, **minimal at 3 seconds**.

Sample Output Data for Query 8 (when executed from MongoDB terminal)


```
{ "Month" : 1, "StockCode" : 10138, "StockDescription" : "ASSORTED COLOUR JUMBO PEN", "Quantity" : 9312 }
{ "Month" : 2, "StockCode" : 10138, "StockDescription" : "ASSORTED COLOUR JUMBO PEN", "Quantity" : 19152 }
{ "Month" : 3, "StockCode" : 10120, "StockDescription" : "DOGGY RUBBER", "Quantity" : 12960 }
{ "Month" : 4, "StockCode" : 10002, "StockDescription" : "INFLATABLE POLITICAL GLOBE", "Quantity" : 1728 }
{ "Month" : 5, "StockCode" : 10002, "StockDescription" : "INFLATABLE POLITICAL GLOBE", "Quantity" : 5000 }
{ "Month" : 6, "StockCode" : 10002, "StockDescription" : "INFLATABLE POLITICAL GLOBE", "Quantity" : 3500 }
{ "Month" : 7, "StockCode" : 15036, "StockDescription" : "ASSORTED COLOURS SILK FAN", "Quantity" : 3186 }
{ "Month" : 8, "StockCode" : 10002, "StockDescription" : "INFLATABLE POLITICAL GLOBE", "Quantity" : 7128 }
{ "Month" : 9, "StockCode" : 15036, "StockDescription" : "ASSORTED COLOURS SILK FAN", "Quantity" : 7128 }
{ "Month" : 10, "StockCode" : 11001, "StockDescription" : "ASSTD DESIGN RACING CAR PEN", "Quantity" : 4800 }
{ "Month" : 11, "StockCode" : 16033, "StockDescription" : "MINI HIGHLIGHTER PENS", "Quantity" : 9360 }
{ "Month" : 12, "StockCode" : 10138, "StockDescription" : "ASSORTED COLOUR JUMBO PEN", "Quantity" : 80995 }
MongoDB Enterprise >
```


- **What is the change in total sales per country per year (trend of sales)? (Query 9)**

For this query, for any stock if there is sales record available for only one year, it is not considered. Also, if there are no record available for two consecutive years, the next available year is considered. For example, if for a stock we have data for the year of 2017 but no record for 2018 and then we have record for 2019, the records for 2019 is compared against 2017.

This query needed a total of five aggregation pipeline stages in MongoDB. First is to do a lookup between the **FACT** collection and the **customer_dim** collection based on **CustomerID** to get the **Country**, since Country is stored with the **CustomerID** in the **customer_dim** table. Then do a **grouping** aggregation based on **Year** and **Country** to find out the total sale per year per country for the second stage. The third stage is a **replaceRoot** to flatten out the structure. The fourth one, a **sort** based on Country and Year so that the records for per Country sales is sorted

by Year. The fifth and final stage is one more **group** aggregate on Country to find out all the sales record available for that Country per year. The 'per year sales record of a country' is grouped into an array sorted by year under that country in the result.
The rest of the logic is done in **JavaScript** while iterating the resultant cursor, i.e. to **find out the sale for the previous year and to compare current year's sales against that.** Surprisingly, despite of so many execution steps, the execution time of the query was 43 seconds.

Sample Output Data for Query 9 (when executed from MongoDB terminal)

```
For Poland sale increase between years 2010 and 2009 is: 3225.37
For Poland sale increase between years 2011 and 2010 is: 281.6100000000001
For RSA sale increase between years 2011 and 2010 is: 70.88
For Lebanon sale record available for only year: 2011 and sale was: 1693.88
For France sale increase between years 2010 and 2009 is: 135099.46
For France sale increase between years 2011 and 2010 is: -3536.9400000000023
For Lithuania sale record available for only year: 2010 and sale was: 6473.04
For Malta sale increase between years 2011 and 2010 is: -1862.5700000000002
For Denmark sale increase between years 2010 and 2009 is: 34163.04
For Denmark sale increase between years 2011 and 2010 is: -18796.510000000002
For Bahrain sale increase between years 2011 and 2010 is: -387.17
```

- **Average spending of customers in a country? (Query 10)**

This query required both '**CustomerID**' and '**Country**' from the '**Customer_dim**' collection. The data was grouped by '**Country**' and the average of '**TotalSales**' was calculated. The results were then sorted and displayed. This query took slightly longer to execute at **51 seconds**.

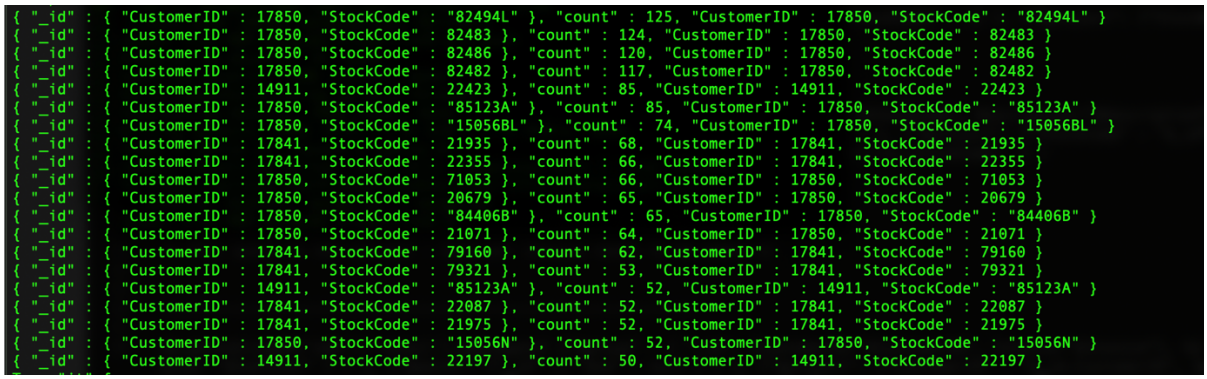
Sample Output Data for Query 10 (when executed from MongoDB terminal)

```
{ "AvgSpending" : 107.2836067146283, "Country" : "Netherlands" }
{ "AvgSpending" : 75.83203893996755, "Country" : "Australia" }
{ "AvgSpending" : 65.50067833698031, "Country" : "Japan" }
{ "AvgSpending" : 63.698314893617024, "Country" : "Sweden" }
{ "AvgSpending" : 56.49427835051546, "Country" : "Denmark" }
{ "AvgSpending" : 40.48451612903226, "Country" : "Singapore" }
{ "AvgSpending" : 40.40184210526316, "Country" : "Thailand" }
{ "AvgSpending" : 37.64177777777778, "Country" : "Lebanon" }
{ "AvgSpending" : 36.5557615610035, "Country" : "EIRE" }
{ "AvgSpending" : 34.6151871657754, "Country" : "Lithuania" }
{ "AvgSpending" : 33.907429718875505, "Country" : "Israel" }
{ "AvgSpending" : 32.73196893063584, "Country" : "Switzerland" }
{ "AvgSpending" : 32.205714285714286, "Country" : "Czech Republic" }
{ "AvgSpending" : 29.585158665581776, "Country" : "Norway" }
{ "AvgSpending" : 29.363720598717038, "Country" : "Channel Islands" }
{ "AvgSpending" : 28.64732564679415, "Country" : "Spain" }
{ "AvgSpending" : 25.587640449438204, "Country" : "Greece" }
{ "AvgSpending" : 24.809038461538464, "Country" : "Austria" }
{ "AvgSpending" : 24.56420572916667, "Country" : "Finland" }
{ "AvgSpending" : 24.40498382703439, "Country" : "France" }
```

- **What is the frequently purchased item per customer? (Query 11)**

This query required a grouping of '**CustomerID**' and '**StockCode**' along with a '**$sum**' variable in place to count the number of occurrences of an item. The results were then sorted by '**count**'. Execution time was once again, minimal at 3 seconds.

Sample Output Data for Query 11 (when executed from MongoDB terminal)



Please note: The execution times of all the above stated queries may vary slightly depending on the system configuration. However, when the queries were executed on different systems, the difference in execution-time was barely few seconds.
The above queries were executed on the following system configuration:
Processor: 2.7 GHz Quad-Core Intel Core i7 | Memory: 16 GB | MacOS

- **Challenges of MongoDB**

  - It has a steep learning curve on how the query and aggregation and aggregation pipeline and stages work along with various accumulation functions available in different aggregation stages and in the query (**find**) process. **Also, learning about memory utilization in mongodb in various stages and how indexes plays their part on that has been enlightening**.

  - The main challenge in **ODB** database was to split the **InvoiceDate** and **InvoiceTime** into individual **Year**, **Month**, **Day**, **Hour**, **Minute** (as the Date and Time were already split in the initial part of the data cleaning). Hence, we imported the **csv file to Python** to do the date and time splitting.

  - An interesting challenge in MongoDB is that, unlike SQL, there is no direct option to insert files in a collection by selecting them from another. So, while creating the ADB tables, we had to read the data record by record and utilize the '**forEach**' function to insert them. However, we realized later that there is a command called '**mongoexport**' which can export columns from a collection (of one database) to a collection (of another database) very easily. (from odb to adb in our case)

  - While we did create the '**Time_Dim**' collection, we did not need to utilize it in any **MongoDB** query execution.

- While processing data from more than one collection, indices were needed to fasten query execution. Without creating an index for the '**Stock_Dim**' collection, for instance, several queries would have taken far longer to execute.
- For queries 4, 6, 7, 8, 9, 10 and 11, we set the '**allowDiskUse**' parameter to **true**. This enables writing to temporary files. When set to true, most aggregation stages can write data to the **_tmp subdirectory** in the **dbPath directory**. This vastly sped up the execution of queries **– from over 10 minutes** (and an **ExceededMemory Error**) to **8 seconds** in the case of **Query 6**.

- **Advantages of MongoDB**

  - The primary advantage of **MongoDB** is its **speed**. MongoDB can very quickly query and return results from very large data sets. Insertion of over **700,000** records took relatively minimal time. This presents major advantages in terms of processing time but also the volumes the database can handle. The **FACT** collection insertion took **5 minutes**, '**Customer_dim**', **6 minutes** and '**Stock_dim**', **4 minutes**. '**Time_dim**' was split due to the creation of the '**time_dim_temp**' collection. This took **4 minutes and 53 seconds**. The creation of the final '**time_dim**' collection only took **13 seconds**.

  - Additionally, we appended two cleaned csv files to a single file. This was because each import command in MongoDB (one for each csv file) took around 40 seconds. So, importing 2 csv files would have taken around 80 seconds. After combining the two files into one file, the import command took around 45 seconds, which means we saved around 35 seconds. While this seems rather trivial, it does speak to MongoDB's ability to handle large volumes of data.

  - While Relational Databases would require views to speedup query execution, this was not required for MongoDB.

  - While sending data to the **User Interface** (**UI**), we are spared from doing further transformation of data in **MongoDB**.

1.1.2 Disadvantages of MongoDB

- **Multiple joins(lookup)** is not advisable in **MongoDB** as it is meant to be a database where **data needs to be denormalized as much as possible**. Each join reduces performance of the entire operation. **Indexes** help them make better, however, if we need queries that needs us to join more than a few collections, **MongoDB is not advisable**.

- In **MongoDB**, the methods available in an **aggregation stage** or in a **find operation** and their behavior is **limited** and **quite rigid**. Also, since the output of MongoDB is in JSON format, there are limits on the operation and calculations that can be achieved using only MongoDB

scripts. For example, **the queries #5 and #9 required extensive calculation on the output of the aggregation which was not possible only through MongoDB scripts. So, some of it was delegated to the caller function.**

Neo4j

- **ODB Creation (Operational Database)**
  - **Creating CSVs**
    - We must change the column "**Customer ID**" to "**CustomerID**" in both Excel Files.
    - We must save the files in a **CSV format**.
  - **Importing the Data**
    - We must store both the **CSV files** in the import folder of the **Database**.
    - We must use the following queries to import the data and store it into nodes with the label **ODB**.
    - Each entry in the **CSV** is converted into a **node** with the label **ODB** with each column as a property.

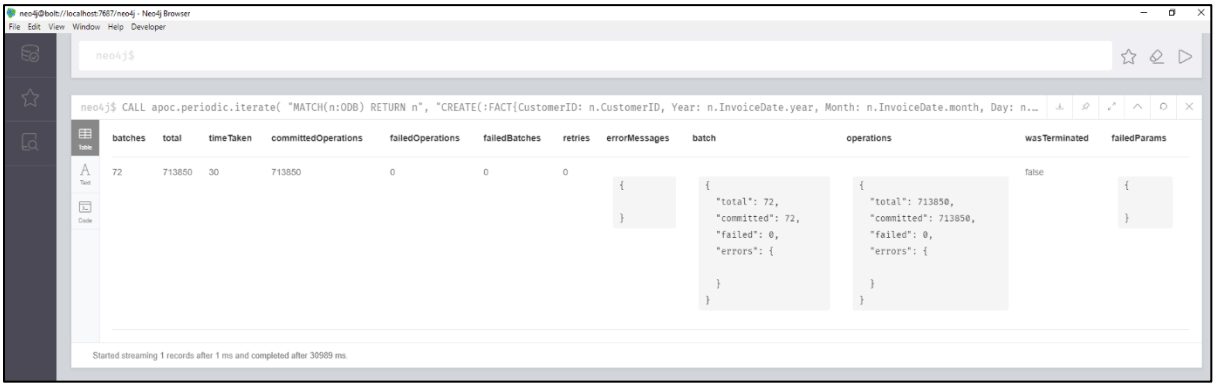
- **Cleaning the ODB**

  - We have converted the values of **InvoiceTime** and **InvoiceDate** into a format which is understood by Neo4j.

- **ADB Creation (Analytical Database)**

  - **Adjustment to the Star Schema by adding TimeID**
    - We have added **TimeID** to the **FACT** table and the **TIMEDIM** table.
    - This has been done as **Neo4j** does not handle the creation of a relation using multiple properties. If this issue is not fixed, it would lead to a very inefficient query.
    - Without **TimeID**, the query would run for 6+ hours if the **dbms.memory.heap.max_size** configuration is changed from 1GB to 10GB in the config file.
    - This was unacceptable and for the query to run in a timely manner we added **TimeID**.
    - The **TimeID** property is not used in anything except the creation of the relation between the **FACT** table and the **TIMEDIM** table.
    - Since we do not use the **TIMEDIM** table, we do not need it and hence it can be removed.

  - **Creating FACT**



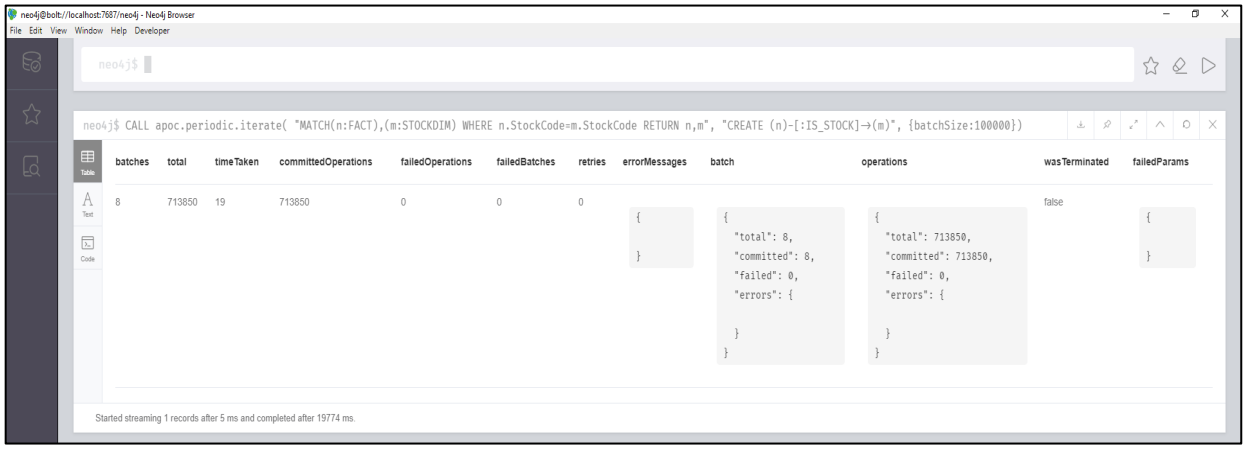
- **Creating CUSTOMERDIM**

- **Creating TIMEDIM**

- **Creating STOCKDIM**

- **Creating the Relationships between the tables**
  - **Creating Relationship between FACT and CUSTOMERDIM**

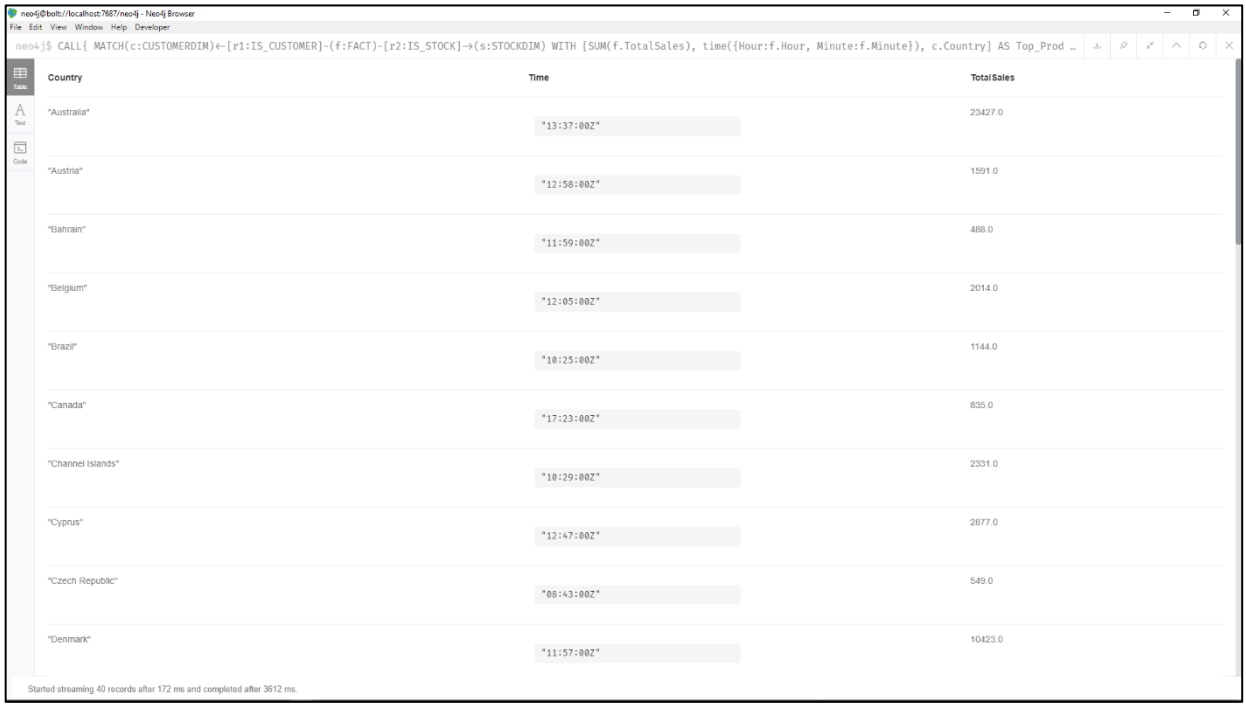
  - **Creating Relationship between FACT and STOCKDIM**



  - **Creating Relationship between FACT and TIMEDIM**

- **Aggregation Queries**
  - **What time of the day (which hour of the day) is the sale maximum per country? (Query 1)**

Sample Output



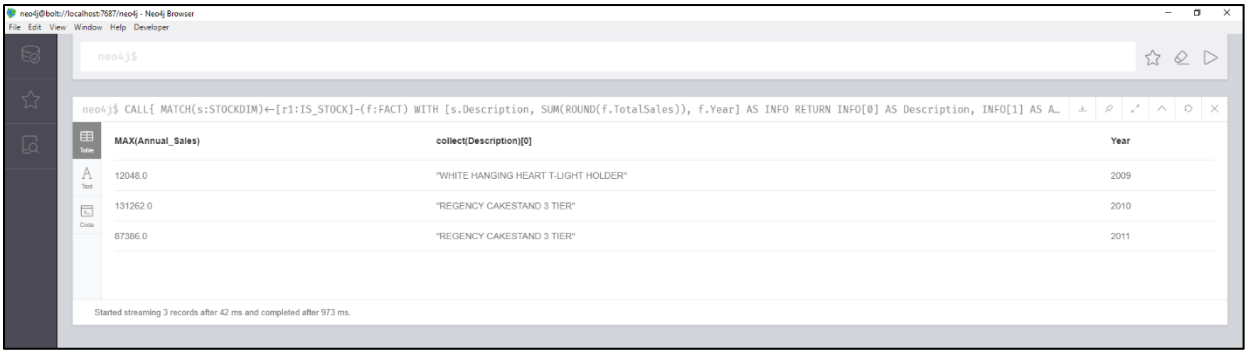
- **What is the annual TotalSales per product? (Query 2)**

Sample Output

Description	Annual_Sales	Year
" DOLLY GIRL BEAKER"	2118.0	2011
" I LOVE LONDON MINI RUCKSACK"	1094.0	2011
" NINE DRAWER OFFICE TIDY"	660.0	2011
" SPACEBOY BABY GIFT SET"	6247.0	2011
" TRELLIS COAT RACK"	904.0	2011
"12 COLOURED PARTY BALLOONS"	1356.0	2010
"12 COLOURED PARTY BALLOONS"	1044.0	2011
"12 DAISY PEGS IN WOOD BOX"	31.0	2009
"12 DAISY PEGS IN WOOD BOX"	967.0	2010
"12 DAISY PEGS IN WOOD BOX"	422.0	2011
"12 EGG HOUSE PAINTED WOOD"	294.0	2009
"12 EGG HOUSE PAINTED WOOD"	2243.0	2010
"12 EGG HOUSE PAINTED WOOD"	1135.0	2011
"12 HANGING EGGS HAND PAINTED"	41.0	2011
"12 IVORY ROSE PEG PLACE SETTINGS"	70.0	2009
"12 IVORY ROSE PEG PLACE SETTINGS"	1929.0	2010
"12 IVORY ROSE PEG PLACE SETTINGS"	623.0	2011
"12 MESSAGE CARDS WITH ENVELOPES"	2539.0	2010
"12 MESSAGE CARDS WITH ENVELOPES"	1610.0	2011

- **What is the top product per year? (Query 3)**

Sample Output



MAX(Annual_Sales)	collect(Description)[0]	Year
12048.0	"WHITE HANGING HEART T-LIGHT HOLDER"	2009
131262.0	"REGENCY CAKESTAND 3 TIER"	2010
87386.0	"REGENCY CAKESTAND 3 TIER"	2011

- **What is the top product per country? (Query 4)**

Sample Output

- **Which customer spends most per country? (Query 6)**

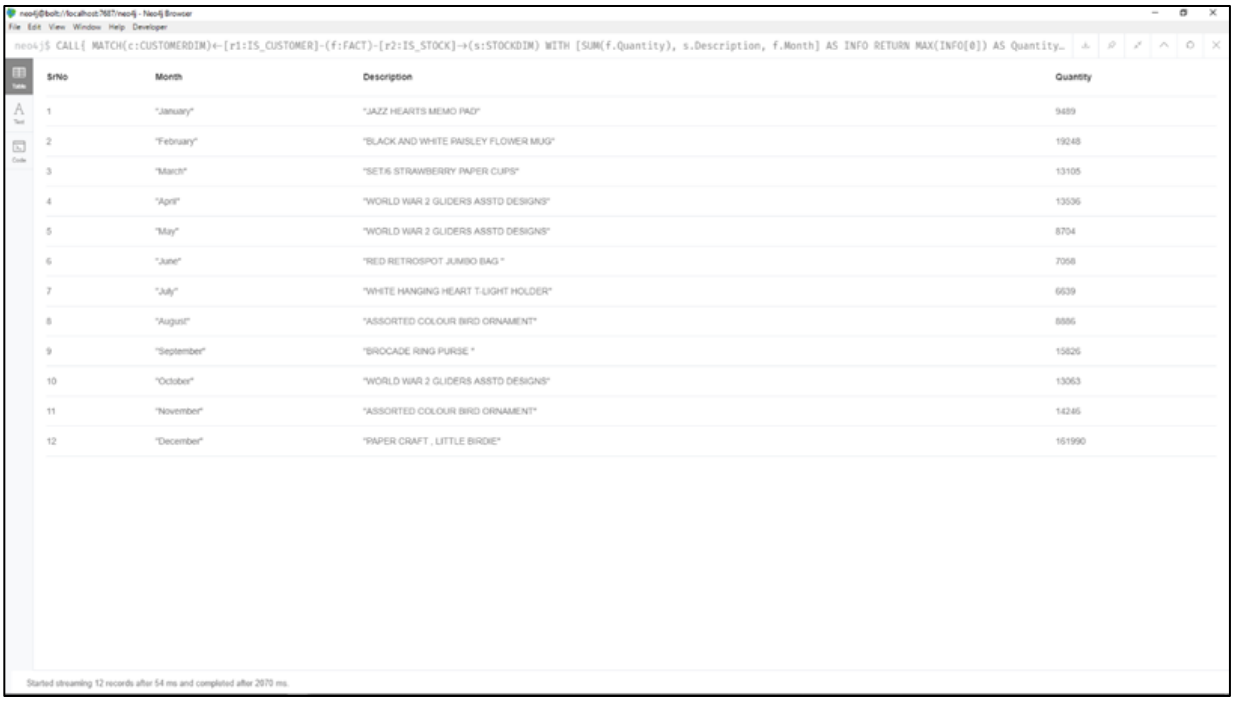
Sample Output

- **What is the best-selling month per country through the years 2009- 2011? (Query 7)**

Sample Output

- **What is the best-selling product per month? (Given the year range, 2009-2011) (Query 8)**

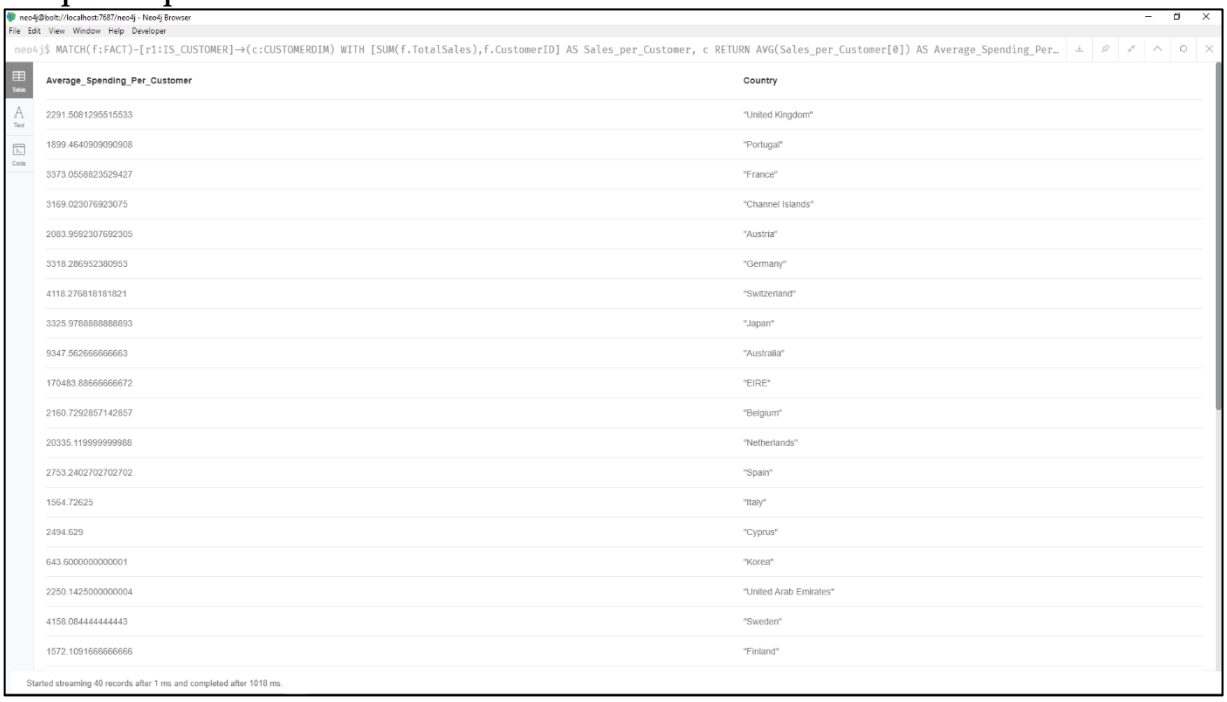
Sample Output



SrNo	Month	Description	Quantity
1	"January"	"JAZZ HEARTS MEMO PAD"	9489
2	"February"	"BLACK AND WHITE PAISLEY FLOWER MUG"	19248
3	"March"	"SET/6 STRAWBERRY PAPER CUPS"	13105
4	"April"	"WORLD WAR 2 GLIDERS ASSTD DESIGNS"	13536
5	"May"	"WORLD WAR 2 GLIDERS ASSTD DESIGNS"	8704
6	"June"	"RED RETROSPOT JUMBO BAG "	7058
7	"July"	"WHITE HANGING HEART T-LIGHT HOLDER"	6639
8	"August"	"ASSORTED COLOUR BIRD ORNAMENT"	8886
9	"September"	"BROCADE RING PURSE "	15826
10	"October"	"WORLD WAR 2 GLIDERS ASSTD DESIGNS"	13063
11	"November"	"ASSORTED COLOUR BIRD ORNAMENT"	14246
12	"December"	"PAPER CRAFT , LITTLE BIRDIE"	161990

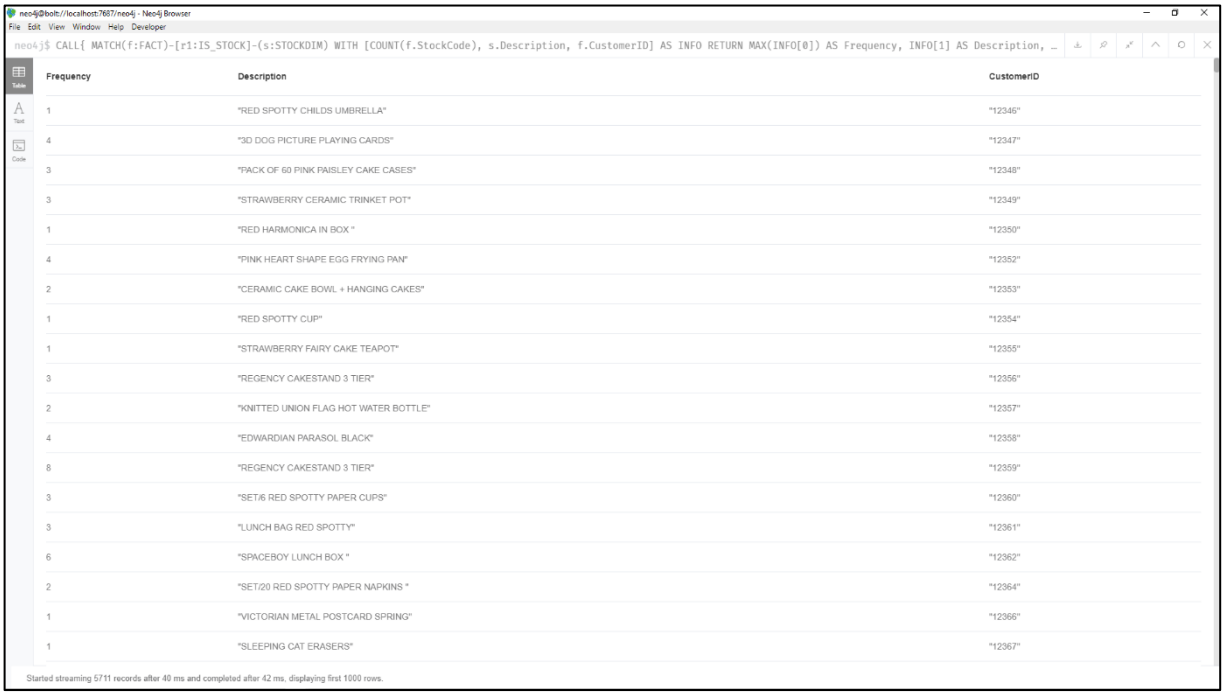
- **What is the average spending of a customer per country? (Query 10)**

Sample Output



Average_Spending_Per_Customer	Country
2291.5081295515533	"United Kingdom"
1899.4640909090908	"Portugal"
3373.0558823529427	"France"
3169.023076923075	"Channel Islands"
2083.9592307692305	"Austria"
3318.286952380953	"Germany"
4118.275818181821	"Switzerland"
3325.9788888888893	"Japan"
9347.562666666663	"Australia"
170483.88666666672	"EIRE"
2160.7292857142857	"Belgium"
20335.119999999988	"Netherlands"
2753.2402702702702	"Spain"
1564.72625	"Italy"
2494.629	"Cyprus"
643.6000000000001	"Korea"
2250.1425000000004	"United Arab Emirates"
4158.084444444443	"Sweden"
1572.1091666666666	"Finland"

- **What is the most frequently purchased item per customer? (Query 11)**



Frequency	Description	CustomerID
1	"RED SPOTTY CHILDS UMBRELLA"	"12346"
4	"3D DOG PICTURE PLAYING CARDS"	"12347"
3	"PACK OF 60 PINK PAISLEY CAKE CASES"	"12348"
3	"STRAWBERRY CERAMIC TRINKET POT"	"12349"
1	"RED HARMONICA IN BOX "	"12350"
4	"PINK HEART SHAPE EGG FRYING PAN"	"12352"
2	"CERAMIC CAKE BOWL + HANGING CAKES"	"12353"
1	"RED SPOTTY CUP"	"12354"
1	"STRAWBERRY FAIRY CAKE TEAPOT"	"12355"
3	"REGENCY CAKESTAND 3 TIER"	"12356"
2	"KNITTED UNION FLAG HOT WATER BOTTLE"	"12357"
4	"EDWARDIAN PARASOL BLACK"	"12358"
8	"REGENCY CAKESTAND 3 TIER"	"12359"
3	"SET/6 RED SPOTTY PAPER CUPS"	"12360"
3	"LUNCH BAG RED SPOTTY"	"12361"
6	"SPACEBOY LUNCH BOX "	"12362"
2	"SET/20 RED SPOTTY PAPER NAPKINS "	"12364"
1	"VICTORIAN METAL POSTCARD SPRING"	"12366"
1	"SLEEPING CAT ERASERS"	"12367"

Started streaming 5711 records after 40 ms and completed after 42 ms, displaying first 1000 rows.

- **Explanations**
  - **Creating ODB**
    - In the query we use to import the information from the CSVs and load into the ODB we use `:auto USING PERIODIC COMMIT 1000`.
    - This tells Neo4j to commit changes every 1000 rows of the CSV.
    - Using this we don't have to import the entire CSV file in batches manually as the database does it for us.
  - **Cleaning InvoiceDate and InvoiceTime**
    - When we load the CSV into the ODB the values of InvoiceDate and InvoiceTime are strings and we need to convert them into a format that is used by Neo4j.
    - For this, we use the DATE() and TIME() functions for InvoiceDate and InvoiceTime respectively with some string manipulation for InvoiceDate
    - We must use apoc.periodic.iterate because Neo4j cannot compute the conversion of InvoiceTime with the hardware constraints.
  - **Creating Tables in ADB**
    - We create the tables from the ODB using apoc.periodic.iterate() function
    - This function allows us to select the information we need from the ODB in the inner query and create new nodes using the outer query in batches so that the system isn't overloaded.
  - **Queries 1, 3, 4, 6, 7, 8 and 11**
    - In these queries, we use a Neo4j function "CALL". This function allows us to take the returned values from a query and perform a second RETURN function on them.
    - All these queries function using a similar technique. We use the query in the CALL to order the results using aggregations or the ORDER BY function.
    - We can then use the COLLECT() function or any other aggregation we want to get our results.
    - For example, in Aggregation query 4 we need to find the Top product per country.
    - So, first, we RETURN the SUM of Total Sales AS Annual Sales, Description and the Year, we ORDER BY Annual Sales in the descending order and then by Year. This will order the results with the highest Annual Sales first
    - In the RETURN of the CALL, we can use COLLECT() to combine all rows with the same Description and display the first value.
    - Since we had ordered them from the largest value of Annual Sale to smallest. The first value in the COLLECT() would be the description of the item which sold most in that year.
    - We also use MAX of the Annual Sales to display the value for the respective Description
    - Since we have not aggregated Year, we get all the values for it
    - This gives us the result we need.

- **Important Notes**
  - **Query 5 and 9**
    - Both the queries require some conditional filtering after aggregation.
    - In order to have multiple aggregations, we need to use the CALL function in Neo4j.
    - CALL allows us to perform more aggregations on the RETURN values of a query. However, this can only be done using a RETURN statement.
    - Due to this, we cannot use any conditional filters or conditions.
    - This is one of the drawbacks of Neo4j we have found.
    - However, in a practical setting where we can use other languages, this issue can be worked around.

- **Advantages of Neo4j**
  - By creating relationships between the nodes and effectively converting out database into a graph, thus we are avoiding the need to define joins in the query saving time.
  - Neo4j is databases do not have a structure and hence is very flexible in terms if changes to the database structure.
  - Because each node does not need to have a structure neo4j can deal with missing data or when there is no definite number of entries in the table.

- **Disadvantages of Neo4j**
  - There are no nested aggregations in a straightforward manner they must be done using the CALL function.
  - Cloning nodes is a very slow process and must be done using the APOC plugin for many entries.
  - Creating Relationships between nodes using multiple properties is a very slow and memory intensive process.
  - The call function does not allow any additional conditional filters.
  - There is no Group By but there are aggregations which are not as flexible as they don't allow aggregation based on a category.
  - There is no way to create a View in Neo4j.

Front-End:

When choosing the front end and backend integration technologies of all three databases, our initial choice was to go with PHP as the server-side scripting that would serve information to the frontend simple HTML and CSS styled webpage. But during the process of building the connections between all three databases, we found it more challenging and yielded less than desirable results. Thus, our final choice was to go with building a JavaScript run-time environment to execute JavaScript code on the server-side using NodeJS, which was able to achieve a connection between front-end and backend smoothly.

- **Integration technologies**
  - Front-end
    - HTML - HyperText markup as the backbone of the web application
    - CSS - Cascading style sheet to style the HTML
    - JavaScript – Scripting and event listener to create/achieve dynamic single page application
  - Back-end
    - NodeJS - JavaScript runtime environment as the server-side engine, enables connection to all three databases, SQL, MongoDB and Neo4j
    - ExpressJS – NodeJS framework to enable server-side routing and information/result serving
    - EJS – JavaScript templating engine to pre-format query results to be served to the front-end
- **Challenges**

  Some of the challenges we faced when building the connection between all three databases and the front-end to show the results include, different connection style for databases that proved to be somewhat challenging during the initial learning stage especially for the database we're not familiar with, query processing speed in certain database and queries are significantly longer than others, which in term prevented the presentation of result instantaneously, so since we're comparing the query processing for different types of database, we included the query runtime that would show how long it takes (in seconds) for each query to produce the result.

Comparison

Notable Features	**Databases**		
	SQL DB	**Mongo DB**	**Neo4j**
Data Loading	Data loading took time	Has mongoimport functionality that loaded data quite fast	Has automated batch loading
Large Data Handling	It does handle but takes time	Handles large data of any kind very well.	Vertical data is not handled well, Horizontal data is handled well
Joining Tables/Collections/Nodes	Required	Possible using lookups, but not recommended.	Permanent Relationships between nodes

Index Creation	Not implemented due to resource constraints	While not required, is encouraged for faster execution	Single and composite indexes can be created on node properties
View Materialization	Required	Not required	Is not required
Query Execution Speed	Slow without view materialization	With indices in place, queries execute quite fast – in the order of a few seconds	Fast, but only for horizontally large data.

Detailed Comparison between RDBMS and NoSQL Approaches

MySQL

For **RDBMS**, we have used **MySQL** query language. For this, we used **PhpMyAdmin** and **MySQL Workbench**. We used both to create the database and loading the dataset. We created insert scripts to load data in these two frameworks. In **PhpMyAdmin**, the dataset took time to load as it has the limit to insert rows of 25000 at a time. In MySQL Workbench, there was no limit so, all the insert scripts were inserted at a time. In **MySQL**, we had to define the data type explicitly. If we are inserting name, we have to mention the column name as well the data type of it along with the length of it.
For example, **name VARCHAR (255)**.
MySQL has different type of joins, which help in getting specific results from the tables. Joins are expensive, so in order to make the queries optimized, views and materialized views can be created.
MySQL can be used where the data is in rows and column where each column is of same datatype.

MongoDB

MongoDB is a **document-oriented NoSQL database** used for **high volume data storage**. Instead of using tables and rows as in the traditional relational databases, **MongoDB** makes use of **collections** and **documents**. **Documents** consist of **key-value pairs** which are the basic unit of data in **MongoDB**. **Collections** contain sets of documents and functions which are equivalent of relational database tables. Each document can be different with a varying number of fields. The size and content of each document can be different from each other. The document structure is more in line with how developers construct their classes and objects in their respective programming languages. The rows (or documents as called in MongoDB) doesn't need to have a schema defined beforehand. Instead, the fields can be created on the fly.

The **data model** available within **MongoDB** allows easy representation of hierarchical relationships, to store arrays, and other more complex structures more easily. Additionally, the MongoDB environments are very scalable, easily allowing for hundreds of thousands of clusters. Another key difference is the

concept of 'Joins'. While relational databases have this functionality, MongoDB is intended for data in a single collection and thus, 'Joins' are not supported as such in MongoDB. One can use Embedded Documents instead but they do not support the complete functionality. Indices can be and should be created in MongoDB – this allows for a much faster query execution by speeding up the performance of searches. Any field in MongoDB can be indexed.

Relational databases are known for enforcing data integrity. This is not an explicit requirement in MongoDB. RDBMS also requires that data be normalized first so that it can prevent orphan records and duplicates. Normalizing data then has the requirement of more tables, which will then result in more table joins, thus requiring more keys and indexes. As databases start to grow, performance can start becoming an issue. Again, this is not an explicit requirement in MongoDB. MongoDB is flexible and does not need the data to be normalized first.

MongoDB can provide high availability with replica sets. A replica set consists of two or more mongo DB instances. Each replica set member may act in the role of the primary or secondary replica at any time. The primary replica is the main server which interacts with the client and performs all the read/write operations. The Secondary replicas maintain a copy of the data of the primary using built-in replication. When a primary replica fails, the replica set automatically switches over to the secondary and then it becomes the primary server.

Finally, MongoDB uses the concept of sharding to scale horizontally by splitting data across multiple MongoDB instances. MongoDB can run over multiple servers, balancing the load and/or duplicating data to keep the system up and running in case of hardware failure.

MongoDB does very well in specific scenarios or requirements. One such example is a content or catalogue management system. Since MongoDB can incorporate any kind of data – text, images, tweets and more, it becomes easy to store all data together in one collection for easy storage and retrieval. Another example is mainframe offloading. Mainframe offloading is the process of replicating commonly accessed mainframe data to an Operational Data Layer (ODL) built on MongoDB, against which operations are redirected from consuming applications. The existing mainframes are left untouched. Redirecting queries away from the mainframe to the ODL significantly reduces costs. It also allows for more agility and scalability.

Neo4j

Neo4j is a Graph-oriented NoSQL database. Rather than using tables and rows to store the data like, No4j uses Nodes. These nodes consist of a node Label and node Properties this is the primary way of storing information. A SQL table can be stored in Neo4j by using the table name as the Labels and the column names as the Properties.

One major disadvantage of relational databases is that they need to use joins in their queries. In Neo4j Nodes can be linked to each other using Relationships avoiding the need for expensive join operations in the queries. This means that multiple Nodes with different Labels can be used in the same query very easily if there is a Relation between the Nodes. Relations can also have Properties like Nodes this allows for even more detailed information about the relation between the nodes to be stored without the need for a separate table.

The data model in Neo4j allows graphs or data with many relations to be stored very easily. It also allows us to create very complicated structures easily. Unlike RDBMS, Neo4j every node does not need to have the same structure which allows us a lot of flexibility in terms of missing data or when there is no definite number of entries in the table. However, Neo4j allows the creation of indexes and constraints based on the nodes Labels, Properties, and Relations allowing integrity constraints to be created when required like a Relational Database. Unlike an RDBMS, Neo4j has inbuilt Spatial functions allowing the database to store and use latitude and longitude values in their nodes.

Due to the nature of Neo4j being a graphical database it lends itself very well in a situation in which the data can be expressed in a graph. For example, a Social Network where each node can be a person or a page and is linked to others via relationships. Neo4j can also be used in a system where the data has been normalized into multiple tables and needs to be joined frequently. For example, the database of a multinational company, here we can use relationships to create a permanent SQL like join between the tables. This can speed up the queries which require joins but do not have a lot of aggregation.

Conclusion

In a real-life scenario with so many options available at hand, one has to consider various aspects before settling down on to which database he/she should use for building his/her application. Cost is one of the important factors that one should always keep in mind. Second, there should be a trade-off between data accuracy, data accessibility and availability and high speed.
With all the elaborate discussions and comparisons done so far, a conclusion can be drawn that Relational Database (MySQL DB in our case) works best for the Online Retail Datawarehouse. One of the main reasons lies in the fact that all the probable analytical / aggregate queries that could be asked by the store manager to check the growth of the store were answered only by RDBMS. That is, data accuracy comes to play while checking the growth of the online store rather than other important aspects. The other two NoSQL databases failed to produce any results for two queries, which could be thought of as important queries.
However, speed was taken into consideration as materialized views were implemented.

APPENDIX A

RELATIONAL DATABASE (MySQL)

ODB Creation

```
CREATE DATABASE online_retail;

USE online_retail;
```

```
CREATE TABLE IF NOT EXISTS ONLINE_RETAIL_ODB(
Invoice varchar(15),
StockCode varchar(15),
Description varchar(100),
Quantity int(11),
InvoiceDate date,
InvoiceTime time(0),
Price numeric(10,2),
Customer_ID varchar(15),
Country varchar(100),
SubTotal numeric(10,2)
);
```

ADB creation

Creating the FACT Table (From ODB to ADB)

```
CREATE TABLE IF NOT EXISTS fact
  (
     customer_id VARCHAR(15),
     stockcode   VARCHAR(15),
     years       INT(4),
     month       INT(2),
     day         INT(2),
     hour        INT(2),
     minute      INT(2),
     quantity    VARCHAR(100),
     totalsale   VARCHAR(100)
  )
```

```
INSERT INTO `fact`
            (`customer_id`,
             `stockcode`,
             `years`,
             `month`,
             `day`,
             `hour`,
             `minute`,
             `quantity`,
             `totalsale`)
SELECT customer_id,
       stockcode,
       Year(invoicedate),
       Month(invoicedate),
       Day(invoicedate),
       Hour(invoicetime),
       Minute(invoicetime),
       quantity,
       Sum(subtotal)
FROM   online_retail_odb
GROUP  BY customer_id,
          stockcode,
          invoicedate,
          invoicetime,
          quantity
```

Creating the CustomerDim table (From ODB to ADB)

```
USE retail_database;

CREATE TABLE IF NOT EXISTS customer_dim_table
  (
     customer_id VARCHAR(100),
     country     VARCHAR(100)
  );

INSERT INTO `customer_dim_table`
            (`customer_id`,
             `country`)
SELECT customer_id,
       country
FROM   online_retail_odb
GROUP  BY customer_id,
          country
```

Creating the StockDim Collection (From ODB to ADB)

```
CREATE TABLE IF NOT EXISTS stock_dim_table
  (
     stockcode   VARCHAR(15),
     description VARCHAR(100),
     unitprice   VARCHAR(100)
  )
```

```
INSERT INTO `stock_dim_table`
            (`stockcode`,
             `description`,
             `unitprice`)
SELECT stockcode,
       description,
       Max(price)
FROM   online_retail_odb
GROUP  BY stockcode,
          description
```

Creating the TimeDim Collection (From ODB to ADB)

```
CREATE TABLE IF NOT EXISTS time_dim_table
  (
     years  INT(4),
     month  INT(2),
     day    INT(2),
     hour   INT(2),
     minute INT(2)
  )
```

```
INSERT INTO `time_dim_table`
            (`years`,
             `month`,
             `day`,
             `hour`,
             `minute`)
SELECT DISTINCT Year(invoicedate),
                Month(invoicedate),
                Day(invoicedate),
                Hour(invoicetime),
                Minute(invoicetime)
FROM   online_retail_odb
```

Aggregation Queries

1. What time of the day is the sale maximum per country?

SQL Query:

```
SELECT a.hour,
       a.total AS max_total_sale,
       a.country
FROM   (SELECT fact.hour           AS Hour,
               Sum(fact.totalsale) AS total,
               customerdim.country AS Country
        FROM   fact
               JOIN customerdim
                 ON fact.customer_id = customerdim.customer_id
        GROUP  BY fact.hour,
                  customerdim.country) AS a
       INNER JOIN (SELECT Max(total) AS maxtotal,
                          country
                   FROM   (SELECT fact.hour           AS Hour,
                                  Sum(fact.totalsale) AS total,
                                  customerdim.country AS Country
                           FROM   fact
                                  JOIN customerdim
                                    ON fact.customer_id =
                                       customerdim.customer_id
                           GROUP  BY fact.hour,
                                     customerdim.country) t1
                   GROUP  BY country) AS b
               ON a.total = b.maxtotal
                  AND a.country = b.country;
```

Create materialized query:

```
CREATE TABLE mview_sum_sale_country
  (
     hour       INT(2),
     totalsale  DECIMAL(32, 2),
     country    VARCHAR(100)
  );
```

Insert into materialized query:

```
INSERT INTO mview_sum_sale_country
(SELECT fact.hour,
        SUM(fact.totalsale) AS total,
        customerdim.country
 FROM   fact
        join customerdim
          ON fact.customer_id = customerdim.customer_id
 GROUP  BY fact.hour,
           customerdim.country);
```

Query using materialized query

```
SELECT f.hour,
       f.totalsale,
       f.country
FROM   (SELECT Max(totalsale)AS total,
               country
        FROM   mview_sum_sale_country
        GROUP  BY country) AS x
       INNER JOIN mview_sum_sale_country AS f
               ON f.totalsale = x.total
                  AND x.country = f.country;
```

2. Annual Total Sales Per Product?

SQL Query:

```
SELECT fact.stockcode,
       fact.year,
       Sum(fact.totalsale),
       stockdim.description
FROM   fact
       JOIN stockdim
         ON fact.stockcode = stockdim.stockcode
GROUP  BY stockdim.stockcode,
          fact.year,
          stockdim.description;
```

Create materialized query:

```
CREATE TABLE mview_totalsale_product
  (
     stockcode   VARCHAR(15),
     year        INT(4),
     description VARCHAR(100),
     totalsale   NUMERIC(10, 2)
  )
```

Insert into materialized query:

```
INSERT INTO mview_totalsale_product
(SELECT fact.stockcode,
        fact.year,
        stockdim.description,
        SUM(fact.totalsale)
 FROM   fact
        join stockdim
          ON fact.stockcode = stockdim.stockcode
 GROUP  BY stockcode,
           year)
```

Query using materialized query

```
SELECT *
FROM   mview_totalsale_product
```

3. Top Product Per Year

SQL Query:

```
SELECT f.stockcode,
       f.description,
       f.year,
       f.totalsale AS totalSale
FROM   (SELECT year,
               Max(totalsale) AS maxtotal
        FROM   (SELECT stockdim.stockcode,
                       fact.year           AS year,
                       stockdim.description,
                       Sum(fact.totalsale) AS TotalSale
                FROM   fact
                       JOIN stockdim
                         ON fact.stockcode = stockdim.stockcode
                GROUP  BY stockdim.stockcode,
                          fact.year,
                          stockdim.description)t1
```

```
        GROUP  BY year) AS x
       INNER JOIN (SELECT stockdim.stockcode,
                          fact.year,
                          stockdim.description,
                          Sum(fact.totalsale)
                   FROM   fact
                          JOIN stockdim
                            ON fact.stockcode = stockdim.stockcode
                   GROUP  BY stockdim.stockcode,
                             fact.year,
                             stockdim.description)AS f
                  ON f.totalsale = x.maxtotal
                     AND x.year = f.year;
```

Query using materialized query:

```
SELECT f.stockcode,
       f.description,
       f.year,
       f.totalsale AS totalSale
FROM   (SELECT year,
               Max(totalsale) AS maxtotal
        FROM   mview_totalsale_product
        GROUP  BY year) AS x
       INNER JOIN mview_totalsale_product AS f
               ON f.totalsale = x.maxtotal
                  AND x.year = f.year;
```

4. Top Product Per Country

SQL Query:

```
USE retail_database;

SELECT country,
       Max(total)
FROM   (SELECT fact.stockcode      AS StockCode,
               customerdim.country AS country,
               Sum(fact.quantity)  AS total
        FROM   fact
               JOIN customerdim
                 ON fact.customer_id = customerdim.customer_id
        GROUP  BY stockcode,
                  country) AS total
GROUP  BY country
```

Create materialized query:

```
CREATE TABLE mview_product_country
  (
     stockcode VARCHAR(15),
     country   VARCHAR(100),
     quantity  INT(11)
  )
```

Insert into materialized query:

```
INSERT INTO mview_product_country
(SELECT fact.stockcode      AS StockCode,
        customerdim.country AS country,
        SUM(fact.quantity)  AS total
 FROM   fact
        join customerdim
          ON fact.customer_id = customerdim.customer_id
 GROUP  BY stockcode,
           country)
```

Query using materialized query:

```
SELECT f.stockcode,
       f.country,
       f.quantity AS Total
FROM   (SELECT country,
               Max(quantity) AS maxtotal
        FROM   mview_product_country
        GROUP  BY country) AS x
       INNER JOIN mview_product_country AS f
               ON f.quantity = x.maxtotal
                  AND x.country = f.country
ORDER  BY total DESC
```

5. Query 5

Create materialized query:

```
CREATE TABLE mview_quantity_product_year
  (
     stockcode     VARCHAR(15),
     year          INT(4),
     description   VARCHAR(100),
     totalquantity INT(11)
  );
```

Insert into materialized query:

```
INSERT INTO mview_quantity_product_year
(SELECT stockdim.stockcode,
        fact.year,
        stockdim.description,
        SUM(fact.quantity)
 FROM   fact
        join stockdim
          ON fact.stockcode = stockdim.stockcode
 GROUP  BY stockdim.stockcode,
           fact.year,
           stockdim.description);
```

Query using materialized query:

```
SELECT t1.stockcode,
       t1.description,
       ( t1.totalquant_2010 - t2.totalquant_2009 ) AS difference
FROM   (SELECT stockcode,
               year,
               description,
               totalquantity AS totalquant_2010
        FROM   mview_quantity_product_year
        WHERE  year = 2010) t1
       INNER JOIN (SELECT stockcode,
                          year,
                          description,
                          totalquantity AS totalquant_2009
                   FROM   mview_quantity_product_year
                   WHERE  year = 2009) t2
               ON t1.stockcode = t2.stockcode
                  AND t1.description = t2.description
                  AND t2.totalquant_2009 <= t1.totalquant_2010
ORDER  BY `difference` ASC
LIMIT  25;
```

6. **Which customer spends the most per country**

SQL Query:

```
SELECT f.customer_id,
       f.totalsale,
       f.country
FROM   (SELECT Max(totalsale) AS maxtotal,
               country
        FROM   (SELECT fact.customer_id    AS customerID,
                       customerdim.country AS country,
                       Sum(fact.totalsale) AS totalSale
                FROM   fact
                       JOIN customerdim
                         ON fact.customer_id = customerdim.customer_id
                GROUP  BY fact.customer_id,
                          customerdim.country) t1
        GROUP  BY country) AS x
       INNER JOIN (SELECT fact.customer_id    AS customerID,
                          customerdim.country AS country,
                          Sum(fact.totalsale) AS totalSale
                   FROM   fact
                          JOIN customerdim
                            ON fact.customer_id = customerdim.customer_id
                   GROUP  BY fact.customer_id,
                             customerdim.country) AS f
               ON f.totalsale = x.maxtotal
                  AND x.country = f.country;
```

Create materialized query:

```
CREATE TABLE mview_customer_total_country
  (
     customer_id VARCHAR(15),
     country     VARCHAR(100),
     totalsale   NUMERIC(10, 2)
  )
```

Insert into materialized query:

```
INSERT INTO mview_customer_total_country
(SELECT fact.customer_id    AS customerID,
        customerdim.country AS country,
        SUM(fact.totalsale) AS totalSale
 FROM   fact
        join customerdim
          ON fact.customer_id = customerdim.customer_id
 GROUP  BY fact.customer_id,
           customerdim.country)
```

Query using materialized query:

```
SELECT f.customer_id,
       f.totalsale,
       f.country
FROM   (SELECT Max(totalsale)AS maxtotal,
               country
        FROM   mview_customer_total_country
        GROUP  BY country) AS x
       INNER JOIN mview_customer_total_country AS f
               ON f.totalsale = x.maxtotal
                  AND x.country = f.country;
```

7. Best-selling month (per country) [2009-2011]

SQL Query:

```
SELECT month,
       country,
       Max(totalsale)
FROM   (SELECT fact.month          AS Month,
               customerdim.country AS country,
               Sum(fact.totalsale) AS totalSale
        FROM   fact
               JOIN customerdim
                 ON fact.customer_id = customerdim.customer_id
        GROUP  BY month,
                  country) AS a
GROUP  BY country
```

Create materialized query:

```
CREATE TABLE mview_sale_month
  (
     month     INT(2),
     country   VARCHAR(100),
     totalsale NUMERIC(10, 2)
  )
```

Insert into materialized query:

```
INSERT INTO mview_sale_month
(SELECT fact.month           AS Month,
        customerdim.country AS country,
        SUM(fact.totalsale) AS totalSale
 FROM   fact
        join customerdim
          ON fact.customer_id = customerdim.customer_id
 GROUP  BY month,
           country)
```

Query using materialized query:

```
SELECT f.month,
       f.totalsale,
       f.country
FROM   (SELECT Max(totalsale) AS maxtotal,
               country
        FROM   mview_sale_month
        GROUP  BY country) AS x
       INNER JOIN mview_sale_month AS f
               ON f.totalsale = x.maxtotal
                  AND x.country = f.country;
```

8. Best-selling product per month [2009-2011]

SQL Query:

```
SELECT stockcode,
       month,
       Max(totalquantity) AS Quantity
FROM   (SELECT fact.stockcode     AS StockCode,
               fact.month         AS Month,
               Sum(fact.quantity) AS totalQuantity
        FROM   fact
        GROUP  BY fact.month,
                  fact.stockcode) AS a
GROUP  BY stockcode,
          month;
```

Create materialized query:

```
CREATE TABLE mview_quantity_product
  (
     stockcode     VARCHAR(15),
     month         INT(2),
     description   VARCHAR(100),
     totalquantity INT(11)
  );
```

Insert into materialized query:

```
INSERT INTO mview_quantity_product
(SELECT stockdim.stockcode,
        fact.month,
        stockdim.description,
        SUM(fact.quantity)
 FROM   fact
        join stockdim
          ON fact.stockcode = stockdim.stockcode
 GROUP  BY stockdim.stockcode,
           fact.month,
           stockdim.description);
```

Query using materialized query:

```
SELECT t3.stockcode,
       t3.description,
       t3.totalquantity AS maxtotal,
       t3.month
FROM   (SELECT Max(maxtotal) AS total,
               month
        FROM   (SELECT stockcode,
                       description,
                       month,
                       Max(totalquantity) AS maxtotal
                FROM   mview_quantity_product
                GROUP  BY stockcode,
                          description,
                          month)t1
        GROUP  BY month) t2
       INNER JOIN mview quantity product t3
               ON t3.totalquantity = t2.total
                  AND t3.month = t2.month
ORDER  BY t3.month ASC
```

9. What is the change in total sales per country per year?

Create materialized query:

```
CREATE TABLE mview_yearsale_country
  (
     year      INT(4),
     totalsale NUMERIC(10, 2),
     country   VARCHAR(100)
  )
```

Insert into materialized query:

```
INSERT INTO materialized VIEW:
INSERT INTO mview_yearsale_country
            (
                        SELECT   fact.year,
                                 sum(fact.totalsale),
                                 customerdim.country
                        FROM     online_retail_adb.fact
                        JOIN     online_retail_adb.customerdim
                        ON       fact.customer_id=customerdim.customer_id
                        GROUP BY fact.year,
                                 customerdim.country
            )
```

Query using materialized query

```
SELECT a.sub_2010_2009,
       b.sub_2011_2010,
       b.country
FROM   (SELECT ( t1.totalsale - t2.totalsale ) AS sub_2010_2009,
               t1.country
        FROM  (SELECT totalsale,
                      country
               FROM   mview_yearsale_country
               WHERE  year = 2010) t1,
              (SELECT totalsale,
                      country
               FROM   mview_yearsale_country
               WHERE  year = 2009) t2
        WHERE  t1.country = t2.country) AS a
       INNER JOIN (SELECT ( t1.totalsale - t2.totalsale ) AS sub_2011_2010,
                          t1.country
                   FROM  (SELECT totalsale,
                                 country
                          FROM   mview_yearsale_country
                          WHERE  year = 2011) t1,
                         (SELECT totalsale,
                                 country
                          FROM   mview_yearsale_country
                          WHERE  year = 2010) t2
                   WHERE  t1.country = t2.country) AS b
               ON b.country = a.country;
```

10. Average spending of customers in a country.

SQL Query:

```
SELECT ( Sum(totalsale) / Count(customer_id) ) AS Avg_Spending,
       country
FROM   (SELECT fact.customer_id,
               Sum(fact.totalsale) AS TotalSale,
               customerdim.country
        FROM   fact
               JOIN customerdim
                 ON fact.customer_id = customerdim.customer_id
        GROUP  BY fact.customer_id,
                  customerdim.country) AS a
GROUP  BY country
```

Query using materialized query:

```
SELECT ( Count(customer_id) / Sum(totalsale) ) AS Avg_Spending,
       country
FROM   mview_customer_total_country
```

11. Frequently purchased item per customer.

SQL Query:

```
SELECT f.customer_id,
       f.stockcode,
       f.stockcount
FROM   (SELECT Max(stockcount) AS maxstockcount,
               customer_id
        FROM   (SELECT customer_id,
                       stockcode,
                       Count(stockcode) AS stockcount
                FROM   fact
                GROUP  BY customer_id,
                          stockcode) t1
        GROUP  BY customer_id) AS x
       INNER JOIN (SELECT customer_id,
                          stockcode,
                          Count(stockcode) AS stockcount
                   FROM   fact
                   GROUP  BY customer_id,
                             stockcode) AS f
               ON f.stockcount = x.maxstockcount
                  AND f.customer_id = x.customer_id;
```

APPENDIX B

MONGODB

ODB Creation

```
use odb
mongoimport --db odb --collection retail --type csv --headerline --file Online_Retail_Data.csv
```

ADB Creation

```
use adb
db.getSiblingDB('odb')['retail'].find( { },
                                          { CustomerID: 1,
                                            Day: 1,
                                            Month: 1,
                                            Year: 1,
                                            Hour: 1,
                                            Minute: 1,
                                            StockCode: 1,
                                            Quantity: 1,
                                            SubTotal: 1
                                          }
                                       ).forEach(function(rec){db.fact.insert(rec) } )
```

Creating the FACT Collection (From ODB to ADB)

```
db.getSiblingDB('odb')['retail'].find( { },
                                          { CustomerID: 1,
                                            Day: 1,
                                            Month: 1,
                                            Year: 1,
                                            Hour: 1,
                                            Minute: 1,
                                            StockCode: 1,
                                            Quantity: 1,
                                            SubTotal: 1
                                          }
                                       ).forEach(function(rec){db.fact.insert(rec) } )
```

Creating the Customer_Dim Collection (From ODB to ADB)

```
db.getSiblingDB('odb')['retail'].aggregate(
        {"$group" :
                { "_id": {CustomerID:"$CustomerID",
                                Country:"$Country"}
        }
        },
                {"$match":
                        {"_id" :
                                { "$ne" : null }
                        }
                },
                {"$project":
                        {CustomerID : "$_id.CustomerID",
                        Country: "$_id.Country", "_id" : 0}
                }
).forEach( function (rec) { db.customer_dim.insert(rec)  } )
```

Creating the Stock_Dim Collection (From ODB to ADB)

```
db.stock_dim.createIndex( { "StockCode": 1 } )

db.getSiblingDB('odb')['retail'].aggregate (
        [{
            $project: {
                StockCode: '$StockCode',
                Description: '$Description',
                Price: '$Price',
                desc_lec: {
                    $strLenCP: '$Description'
                }
            }
        }, {
            $group: {
                _id: '$StockCode',
                Description: {
                    $first: '$Description'
                },
                MaxUnitPrice: {
                    $max: '$Price'
                },
                max_desc_len: {
                    $max: '$desc_lec'
                }
            }
        }, {
            $project: {
                StockCode: '$_id',
                Description: 1,
                MaxUnitPrice: 1,
                _id: 0
            }
        }]
).forEach( function (rec) {
        db.stock_dim.insert(rec)
})
```

Creating the Time_Dim Collection (From ODB to ADB)

```
db.getSiblingDB('odb')['retail'].find( { },
                                       { Day: 1,
                                           Month: 1,
                                           Year: 1,
                                           Hour: 1,
                                           Minute: 1
                                       }).forEach(function(rec){db.time_dim_temp.insert(rec) } )


db.time_dim_temp.aggregate (
        [{
            $group: {
                _id: {
                    Year: "$Year",
                    Month: "$Month",
                    Day: "$Day",
                    Hour: "$Hour",
                    Minute: "$Minute"
                }
            }
        }
        ]).forEach( function(rec){ db.time_dim.insert(rec) } )
```

Aggregation Queries

12. What time of the day is the sale maximum per country?

```
db.customer_dim.createIndex( { "CustomerID": 1 } )
db.fact.aggregate(
        [{
            $lookup: {
                from: 'customer_dim',
                localField: 'CustomerID',
                foreignField: 'CustomerID',
                as: 'customer_rec'
            }
        }, {
            $unwind: {
                path: '$customer_rec'
            }
        }, {
            $project: {
                Hour: 1,
                Country: '$customer_rec.Country',
                TotalSales: 1,
                _id: 0
            }
        }, {
            $group: {
                _id: {
                    Hour: '$Hour',
                    Country: '$Country'
                },
                TotalSalePerCountry: {
                    $sum: '$TotalSales'
                }
            }
        }, {
            $project: {
                Hour: '$_id.Hour',
                Country: '$_id.Country',
                TotalSalePerCountry: 1,
                rec: '$$ROOT',
                _id: 0
            }
        }, {
            $group: {
                _id: '$Country',
                MaxSalePerHourPerCountry: {
                    $max: '$TotalSalePerCountry'
                },
                rec: {
                    $first: '$rec'
                }
            }
        }, {
            $project: {
                Country: '$_id',
                MaxSalePerHourPerCountry: 1,
                Hour: '$rec._id.Hour'
            }
        }, {
            $sort: {
                Country: 1
            }
        }]
)
```

13. Annual Total Sales Per Product?

```
db.fact.aggregate (
        [{
            $group: {
                _id: {
                    "Year": "$Year",
                    "Stock": "$StockCode"
                },
                TotalAnnualSales: {
                    $sum: "$TotalSales"
                }
            }
        }, {
            $project: {
                Year: "$_id.Year",
                Stock: "$_id.Stock",
                TotalAnnualSales: 1,
                _id: 0
            }
        }, {
            $sort: {
                Stock: 1,
                Year: 1
            }
        }]
)
```

14. Top Product Per Year

```
db.fact.aggregate (
        [{
            $group: {
                _id: {
                    "Year": "$Year",
                    "StockCode": "$StockCode"
                },
                TotalAnnualSales: {
                    $max: {
                        $sum: "$TotalSales"
                    }
                }
            }
        }, {
            $project: {
                TotalAnnualSales: 1,
                stock_doc: "$$ROOT"
            }
        }, {
            $group: {
                _id: "$_id.Year",
                TotalAnnualSales: {
                    $max: "$TotalAnnualSales"
                },
                stock_doc: {
                    $first: "$stock_doc"
                }
            }
        }, {
            $project: {
                Year: "$_id",
                StockCode: "$stock_doc._id.StockCode",
                TotalAnnualSales: 1,
                _id: 0
            }
        }, {
            $sort: {
                Year: 1
            }
        }]
)
```

15. Top Product Per Country

```
db.fact.aggregate (
        [{
            $lookup: {
                from: 'customer_dim',
                localField: 'CustomerID',
                foreignField: 'CustomerID',
                as: 'customer_rec'
            }
        }, {
            $unwind: {
                path: '$customer_rec'
            }
        }, {
            $group: {
                _id: {
                    StockCode: '$StockCode',
                    Country: '$customer_rec.Country'
                },
                TotalSalePsPc: {
                    $sum: '$TotalSales'
                },
                all_sales: {
                    $push: {
                        sale_rec: '$$ROOT'
                    }
                }
            }
        }, {
            $project: {
                StockCode: '$_id.StockCode',
                Country: '$_id.Country',
                TotalSalePsPc: 1,
                rec: '$$ROOT',
                _id: 0
            }
        }, {
            $group: {
                _id: '$Country',
                TotalSalePerStock: {
                    $max: '$TotalSalePsPc'
                },
                rec: {
                    $first: '$rec'
                }
            }
        }, {
            $project: {
                StockCode: '$rec._id.StockCode',
                Country: '$_id',
                TotalSalePerStock: 1,
                _id: 0
            }
        }, {
            $sort: {
                TotalSalePerStock: -1
            }
        }], {
                allowDiskUse: true
        }
)
```

16. Query 5

```
db.fact.aggregate(
   [{
         $group: {
             _id: {
                 StockCode: '$StockCode',
                 Year: '$Year'
             },
             TotalSaleQ: {
                 $sum: '$Quantity'
             },
             count: {
                 $sum: 1
             }
         }
     }, {
         $sort: {
             _id: 1
         }
     }, {
         $group: {
             _id: '$_id.StockCode',
             sale_q_per_year: {
                 $push: {
                     Year: '$_id.Year',
                     TotalSaleQ: '$TotalSaleQ',
                     count: '$count'
                 }
             }
         }
     }]
 ).forEach ( function (stkReport) {
         var stkCd = stkReport._id;
         var prevYr = stkReport.sale_q_per_year[0].Year;
         var prevYrAvg = stkReport.sale_q_per_year[0].TotalSaleQ/stkReport.sale_q_per_year[0].count;
         for (i = 1; i < stkReport.sale_q_per_year.length; i++) {
             var rec = stkReport.sale_q_per_year[i];
             var curYrAvg = rec.TotalSaleQ / rec.count;
             print("for StockCode:" + stkCd + " previous year: " + prevYr + "'s average: " + prevYrAvg + " and current
year " + rec.Year + "'s average is " + curYrAvg);
             prevYrAvg = curYrAvg;
         }
     }
 )
```

17. Which customer spends the most (per country or overall)

Overall:

```
db.fact.aggregate (
        [{
            $project: {
                CustomerID: '$CustomerID',
                TotalSales: '$TotalSales',
                rec: '$$ROOT',
                _id: 0
            }
        }, {
            $group: {
                _id: '$CustomerID',
                TotalSales: {
                    $sum: '$TotalSales'
                },
                all_sales: {
                    $push: {
                        sale: '$rec'
                    }
                }
            }
        }, {
            $sort: {
                TotalSales: -1
            }
        }, {
            $project: {
                CustomerID: '$_id',
                TotalSales: 1,
                rec: '$$ROOT'
            }
        }, {
            $group: {
                _id: null,
                Max: {
                    $max: '$TotalSales'
                },
                rec: {
                    $first: '$rec'
                }
            }
        }, {
            $project: {
                Sale: '$Max',
                CustomerID: '$rec._id',
                _id: 0
            }
        }], {
                allowDiskUse: true
        }
)
```

Per country:

```
db.fact.aggregate (
        [{
            $lookup: {
                from: 'customer_dim',
                localField: 'CustomerID',
                foreignField: 'CustomerID',
                as: 'customer_rec'
            }
        }, {
            $unwind: {
                path: '$customer_rec'
            }
        }, {
            $group: {
                _id: {
                    CustomerID: '$CustomerID',
                    Country: '$customer_rec.Country'
                },
                TotalSalePerCustomer: {
                    $sum: '$TotalSales'
                },
                all_sales: {
                    $push: {
                        sale_rec: '$$ROOT'
                    }
                }
            }
        }, {
            $sort: {
                TotalSalePerCustomer: -1
            }
        }, {
            $project: {
                CustomerID: '$_id.CustomerID',
                Country: '$_id.Country',
                TotalSalePerCustomer: 1,
                _id: 0
            }
        }, {
            $group: {
                _id: {
                    CustomerID: '$CustomerID',
                    Country: '$Country'
                },
                TotalSalePerCustomer: {
                    $max: '$TotalSalePerCustomer'
                }
            }
        }, {
            $sort: {
                TotalSalePerCustomer: -1
            }
        }, {
            $project: {
                CustomerID: '$_id.CustomerID',
                Country: '$_id.Country',
                TotalSalePerCustomer: 1,
                _id: 0,
            }
        }], {
                allowDiskUse: true
        }
)
```

18. Best-selling month (per country) [2009-2011]

```
db.fact.aggregate (
        [{
            $lookup: {
                from: 'customer_dim',
                localField: 'CustomerID',
                foreignField: 'CustomerID',
                as: 'customer_rec'
            }
        }, {
            $unwind: {
                path: '$customer_rec'
            }
        }, {
            $project: {
                Month: 1,
                Country: '$customer_rec.Country',
                TotalSales: 1,
                _id: 0
            }
        }, {
            $group: {
                _id: {
                    Month: '$Month',
                    Country: '$Country'
                },
                TotalSalePerCountry: {
                    $sum: '$TotalSales'
                }
            }
        }, {
            $project: {
                Month: '$_id.Month',
                Country: '$_id.Country',
                TotalSalePerCountry: 1,
                rec: '$$ROOT',
                _id: 0
            }
        }, {
            $group: {
                _id: '$Country',
                MaxSalePerHourPerCountry: {
                    $max: '$TotalSalePerCountry'
                },
                rec: {
                    $first: '$rec'
                }
            }
        }, {
            $project: {
                Country: '$_id',
                MaxSalePerHourPerCountry: 1,
                Month: '$rec._id.Month'
            }
        }, {
            $sort: {
                Country: 1
            }
        }], {
                allowDiskUse: true
        }
)
```

19. Best-selling product per month [2009-2011]

```
db.fact.aggregate(
[{
    $lookup: {
        from: 'stock_dim',
        localField: 'StockCode',
        foreignField: 'StockCode',
        as: 'StockCode'
    }
}, {
    $group: {
        _id: '$Month',
        max: {
            $max: '$Quantity'
        },
        StockCode: {
            $first: '$StockCode'
        }
    }
}, {
    $unwind: {
        path: '$StockCode'
    }
}, {
    $project: {
        Month: '$_id',
        StockCode: '$StockCode.StockCode',
        StockDescription: '$StockCode.Description',
        Quantity: '$max',
        _id: 0
    }
}, {
    $sort: {
        Month: 1
    }
}], {
         allowDiskUse: true
     }
 )
```

20. What is the change in total sales per country per year?

```
db.fact.aggregate(
     [{
         $lookup: {
             from: 'customer_dim',
             localField: 'CustomerID',
             foreignField: 'CustomerID',
             as: 'customer_rec'
         }
     }, {
         $replaceRoot: {
             newRoot: {
                 $mergeObjects: [{
                         $arrayElemAt: [ '$customer_rec', 0 ]
                     },
                     '$$ROOT'
                 ]
             }
         }
     }, {
         $group: {
             _id: {
                 Country: '$Country',
                 Year: '$Year'
             },
             TotalSalePerCountry: {
                 $sum: '$TotalSales'
             }
         }
     }, {
         $sort: {
             _id: 1
         }
     }, {
         $group: {
             _id: '$_id.Country',
             all_recs: {
                 $push: {
                     Year: '$_id.Year',
                     TotalSaleInYear: '$TotalSalePerCountry'
                 }
             }
         }
     }], {
         allowDiskUse: true
     }
 ).forEach(
     function(doc) {
         if (doc.all_recs.length > 1) {
             var prevYear = doc.all_recs[0].Year;
             var prevYearSale = doc.all_recs[0].TotalSaleInYear;
             for (i = 1; i < doc.all_recs.length; i++) {var curYear = doc.all_recs[i].Year;
                 var curYearSale = doc.all_recs[i].TotalSaleInYear;
                 var diff = curYearSale - prevYearSale;
                 print("For " + doc._id + " sale increase between years " + curYear + " and " + prevYear + " is: " +
diff);
                 prevYearSale = curYearSale;
                 prevYear = curYear;
             }
         } else if (doc.all_recs.length == 1) {
             print("For " + doc._id + " sale record available for only year: " + doc.all_recs[0].Year + " and sale
was: " + doc.all_recs[0].TotalSaleInYear);
         }
     }
 )
```

21. Average spending of customers in a country.

```
db.fact.aggregate(
        [{
            $lookup: {
                from: 'customer_dim',
                localField: 'CustomerID',
                foreignField: 'CustomerID',
                as: 'customer_rec'
            }
        }, {
            $unwind: {
                path: '$customer_rec'
            }
        }, {
            $project: {
                rec: '$$ROOT'
            }
        }, {
            $group: {
                _id: '$rec.customer_rec.Country',
                AvgSpending: {
                    $avg: '$rec.TotalSales'
                },
                rec: {
                    $first: '$rec'
                }
            }
        }, {
            $project: {
                Country: '$_id',
                AvgSpending: 1,
                _id: 0
            }
        }, {
            $sort: {
                AvgSpending: -1
            }
        }], {
                allowDiskUse: true
        }
)
```

22. Frequently purchased item per customer.

```
db.fact.aggregate(
        [{
            $group: {
                _id: {
                    CustomerID: '$CustomerID',
                    StockCode: '$StockCode'
                },
                count: {
                    $sum: 1
                }
            }
        }, {
            $project: {
                CustomerID: '$_id.CustomerID',
                StockCode: '$_id.StockCode',
                count: 1
            }
        }, {
            $sort: {
                count: -1,
                CustomerID: 1
            }
        }]
)
```

APPENDIX C

NEO4J

1. Query for Importing into and Creating ODB

```
a.
:auto USING PERIODIC COMMIT 1000 LOAD CSV WITH HEADERS FROM 'file:///Cleaned_2009_2010.csv' AS test CREATE (:ODB
{Invoice: test.Invoice, StockCode: test.StockCode, Description: test.Description, Quantity: toInteger(test.Quantity),
InvoiceDate: test.InvoiceDate, InvoiceTime: test.InvoiceTime, Price: toFloat(test.Price), CustomerID: test.CustomerID,
Country:test.Country, SubTotal: toFloat(test.SubTotal)})

b.
:auto USING PERIODIC COMMIT 1000 LOAD CSV WITH HEADERS FROM 'file:///Cleaned_2010_2011.csv' AS test CREATE (:ODB
{Invoice: test.Invoice, StockCode: test.StockCode, Description: test.Description, Quantity: toInteger(test.Quantity),
InvoiceDate: test.InvoiceDate, InvoiceTime: test.InvoiceTime, Price: toFloat(test.Price), CustomerID: test.CustomerID,
Country:test.Country, SubTotal: toFloat(test.SubTotal)})
```

2. Query for Cleaning ODB

```
a.
MATCH(m:ODB)
WITH [item in split(m.InvoiceDate, "/") | toInteger(item)] AS dateComponents, m AS m
SET m.InvoiceDate = date({day: dateComponents[1], month: dateComponents[0], year: dateComponents[2]})

b.
CALL apoc.periodic.iterate(
"MATCH(n:ODB) RETURN n",
"SET n.InvoiceTime = time(n.InvoiceTime)",
{batchSize:100000})
```

3. Query for Creating the Tables

a. FACT

```
CALL apoc.periodic.iterate(
"MATCH(n:ODB) RETURN n",
"CREATE(:FACT{CustomerID: n.CustomerID, Year: n.InvoiceDate.year, Month: n.InvoiceDate.month, Day: n.InvoiceDate.day,
Hour: n.InvoiceTime.hour, Minute: n.InvoiceTime.minute, StockCode: n.StockCode, Quantity: n.Quantity, TotalSales:
n.SubTotal, TimeID:datetime({Year: n.InvoiceDate.year, Month: n.InvoiceDate.month, Day: n.InvoiceDate.day, Hour:
n.InvoiceTime.hour, Minute: n.InvoiceTime.minute})})",
{batchSize:10000})
```

b. CUSTOMERDIM

```
CALL apoc.periodic.iterate(
"MATCH(n:ODB)
RETURN n.CustomerID as CustomerID, n.Country as Country
UNION
MATCH(n:ODB)
RETURN n.CustomerID as CustomerID, n.Country as Country",
"CREATE(:CUSTOMERDIM{ CustomerID: CustomerID , Country: Country})",
{batchSize:10000})
```

c. STOCKDIM

```
CALL apoc.periodic.iterate(
"MATCH (n:ODB) RETURN n.StockCode as StockCode, max(n.Description) as Description, max(n.Price) as Price",
"CREATE(:STOCKDIM{StockCode: StockCode, Description: Description, Price: Price})",
{batchSize:10000})
```

d. TIMEDIM

```
CALL apoc.periodic.iterate(
"MATCH(n:ODB)
RETURN n.InvoiceDate.year as Year, n.InvoiceDate.month as Month, n.InvoiceDate.day as Day,n.InvoiceTime.hour as Hour,
n.InvoiceTime.minute as Minute
UNION
MATCH(n:ODB)
RETURN n.InvoiceDate.year as Year, n.InvoiceDate.month as Month, n.InvoiceDate.day as Day,n.InvoiceTime.hour as Hour,
n.InvoiceTime.minute as Minute",
"CREATE(:TIMEDIM{ Year: Year , Month: Month, Day: Day, Hour: Hour, Minute: Minute, TimeID:datetime({Year: Year ,
Month: Month, Day: Day, Hour: Hour, Minute: Minute})})",
{batchSize:10000})
```

4. Query for Creating the Relationship between Tables

a. Creating Relationship between FACT and CUSTOMERDIM

```
CALL apoc.periodic.iterate(
"MATCH(n:FACT),(m:CUSTOMERDIM)
WHERE n.CustomerID=m.CustomerID
RETURN n,m",
"CREATE (n)-[:IS_CUSTOMER]->(m)",
{batchSize:100000})
```

b. Creating Relationship between FACT and STOCKDIM

```
CALL apoc.periodic.iterate(
"MATCH(n:FACT),(m:STOCKDIM)
WHERE n.StockCode=m.StockCode
RETURN n,m",
"CREATE (n)-[:IS_STOCK]->(m)",
{batchSize:100000})
```

c. Creating Relationship between FACT and TIMEDIM

```
CALL apoc.periodic.iterate(
"MATCH(n:FACT),(m:TIMEDIM)
WHERE n.TimeID=m.TimeID
RETURN n,m",
"CREATE (n)-[:IS_TIME]->(m)",
{batchSize:100000})
```

5. Aggregation Queries

a. **What time of the day (which hour of the day) is the sale maximum per country?**

```
CALL{
MATCH(c:CUSTOMERDIM)<-[r1:IS_CUSTOMER]-(f:FACT)-[r2:IS_STOCK]->(s:STOCKDIM)
WITH [SUM(f.TotalSales), time({Hour:f.Hour, Minute:f.Minute}), c.Country] AS Top_Prod
RETURN MAX(Top_Prod[0]) AS TotalSales, Top_Prod[1] AS Time, Top_Prod[2] AS Country
ORDER BY TotalSales DESC, Country
}
RETURN Country, collect(Time)[0] AS Time, ROUND(MAX(TotalSales)) AS TotalSales
ORDER BY Country
```

b. **What is the annual TotalSales per product?**

```
MATCH(s:STOCKDIM)<-[r1:IS_STOCK]-(f:FACT)
RETURN s.Description AS Description, SUM(ROUND(f.TotalSales)) AS Annual_Sales, f.Year AS Year
ORDER BY Description, Year
```

c. **What is the top product per year?**

```
CALL{
MATCH(s:STOCKDIM)<-[r1:IS_STOCK]-(f:FACT)
WITH [s.Description, SUM(ROUND(f.TotalSales)), f.Year] AS INFO
RETURN INFO[0] AS Description, INFO[1] AS Annual_Sales, INFO[2] AS Year
ORDER BY Annual_Sales DESC, Year
}
RETURN MAX(Annual_Sales), collect(Description)[0], Year
ORDER BY Year
```

d. **What is the top product per country?**

```
CALL{
MATCH(c:CUSTOMERDIM)<-[r1:IS_CUSTOMER]-(f:FACT)-[r2:IS_STOCK]->(s:STOCKDIM)
WITH [SUM(f.Quantity), s.Description, c.Country] AS Top_Prod
RETURN MAX(Top_Prod[0]) AS Quantity, Top_Prod[1] AS Description, Top_Prod[2] AS Country
ORDER BY Quantity DESC, Country ASC
}
RETURN MAX(Quantity), collect(Description)[0], Country
ORDER BY Country
```

e. **Which item is sold below a certain threshold value? Or, what are the under-performed products based on the average sales last year?**

f. **Which customer spends the most per country?**

```
CALL{
MATCH(f:FACT)-[r1:IS_CUSTOMER]->(c:CUSTOMERDIM)
WITH [SUM(f.TotalSales), f.CustomerID, c.Country]  AS Sales_per_Customer, c
RETURN MAX(Sales_per_Customer[0]) AS Total_Sale, Sales_per_Customer[1] AS CustomerID, Sales_per_Customer[2] AS
Country, c, Sales_per_Customer
ORDER BY Total_Sale DESC, Country
}
RETURN MAX(Total_Sale), collect(CustomerID)[0], Country
ORDER BY Country
```

g. **What is the best-selling month per country? (Given the year range, 2009-2011)**

```
CALL{
MATCH(c:CUSTOMERDIM)<-[r1:IS_CUSTOMER]-(f:FACT)-[r2:IS_STOCK]->(s:STOCKDIM)
WITH [SUM(f.Quantity), f.Month, c.Country] AS Top_Prod
RETURN MAX(Top_Prod[0]) AS Quantity, Top_Prod[1] AS Month, Top_Prod[2] AS Country
ORDER BY Quantity DESC, Country
}
RETURN Country, ["January", "February", "March", "April", "May", "June", "July", "August", "September", "October",
"November", "December"][collect(Month)[0]-1] AS Month, MAX(Quantity) AS Quantity
ORDER BY Country
```

h. What is the best-selling product per month? (Given the year range, 2009-2011)

```
CALL{
MATCH(c:CUSTOMERDIM)<-[r1:IS_CUSTOMER]-(f:FACT)-[r2:IS_STOCK]->(s:STOCKDIM)
WITH [SUM(f.Quantity), s.Description, f.Month] AS INFO
RETURN MAX(INFO[0]) AS Quantity, INFO[1] AS Description, INFO[2] AS Months
ORDER BY Quantity DESC, Months
}
RETURN Months AS SrNo, ["January", "February", "March", "April", "May", "June", "July", "August", "September",
"October", "November", "December"][Months-1] AS Month, collect(Description)[0] AS Description, MAX(Quantity) AS
Quantity
ORDER BY SrNo
```

i. What is the change in TotalSales per country per year (Trend of Sales)?

j. What is the average spending of a customer per country? (TotalSales/Number of customers)

```
MATCH(f:FACT)-[r1:IS_CUSTOMER]->(c:CUSTOMERDIM)
WITH [SUM(f.TotalSales),f.CustomerID]  AS Sales_per_Customer, c
RETURN AVG(Sales_per_Customer[0]) AS Average_Spending_Per_Customer, c.Country AS Country
```

k. What is the frequently purchased item per customer?

```
CALL{
MATCH(f:FACT)-[r1:IS_STOCK]-(s:STOCKDIM)
WITH [COUNT(f.StockCode), s.Description, f.CustomerID] AS INFO
RETURN MAX(INFO[0]) AS Frequency, INFO[1] AS Description, INFO[2] AS CustomerID
ORDER BY Frequency DESC, CustomerID
}
RETURN MAX(Frequency) AS Frequency, collect(Description)[0] AS Description, CustomerID
ORDER BY CustomerID
```

6. Order of Execution

- Save the .CSV files folder into the import folder of the database
- Run ODB creation query
- Run the ODB cleaning query
- Create the ADB
- Create the relation between the ADB tables