**Data Analysis Document**

For

**The Kambi Grill**

**DWH Modeling Process**

Author : Sayani Roy

Contents:

Page No.

1. Introduction ........................................................................................... 03

2. Data Preparation .................................................................................... 04

3. Data Execution ......................................................................................... 05

4. Data Modeling ......................................................................................... 06

5. Conclusion ................................................................................................ 35

1. Introduction

The purpose of this document is to design data models to analyse business domains for Kambi Grill. Here, we will use different data tables to extract data and will discuss several business possibilities.

The scope of this document is to find answers to the below questions:

* Gross generated revenue by campaign/store/customer/membership type.
* How long until the first customer orders?
* What items are the most/least popular?
* What is our gross per sale?
* What items should we remove due to customer preferences?
* How can we improve targeting of marketing campaigns?
* How long does it take for an order to be processed/delivered?
* How long does it take for customers to add their first preference?
* How long does it take for customers to improve their membership?
* How many campaigns does it take to improve a customer's membership?
* Which store sees the most campaign action?
* Can we create new "menu meals" to improve sales?
* We want to measure the value of membership types as members return to ensure we have the correct levels.

1. Data Preparation :

A brief idea about the data along with the assumptions is described below for each domain. My assumptions are made in **bold**.

1. **Members Domain**:

This domain is having 4 data tables with one audit table.

* **Membership\_type:**

There are **3 types** of membership available for Kambi Grill, i.e. ‘**Silver**’, ‘**Gold**’, ‘**Platinum**’. Silver is the basic membership and Platinum is most costly one.

* **Customer:**

Customers will be registered with a membership which can be upgraded accordingly. With membership change new customer id will be generated. **The combination of first name, last name and loyalty number will be unique and unchanged forever**.

* **Preference:**

Two main components are describing a preference. Preference name will be ingredients and **preference type will describe whether the ingredient needs to be added or removed**.

* **Customer\_preference:**

This table contains all the mapping data for customer and their preference. **Customers can update their preferences**.

1. **Marketing Domain:**

This domain is having 2 data tables with one audit table.

* **Store:**

This table contains all the store data along with their address and tax details.

* **Campaign:**

This table describes all the campaigns and their status; also the name of memberships and stores which will be tagged for each campaign.

1. **Order Domain:**

This domain is having 4 data tables with one audit table.

* **Order:**

Order table is documenting all the order placed along with customer, used campaign, store details and the key to track order status. The sub total is the "true" value of the order; the total is the "final" value of the order, after taxes and discounts have been applied.

* **Order\_list:**

This table contains all order details like what are the items have been order and what the sale price is for that. **One order can have multiple items which will be tracked in this table**.

* **Order\_Status:**

There are 3 types of order status available, i.e. ‘submitted’, ‘in progress’, ‘delivered’ respectively as per the progress. **Updated\_date is timestamp field which will be updated when any changes made in the order\_status\_name field**.

* **item:**

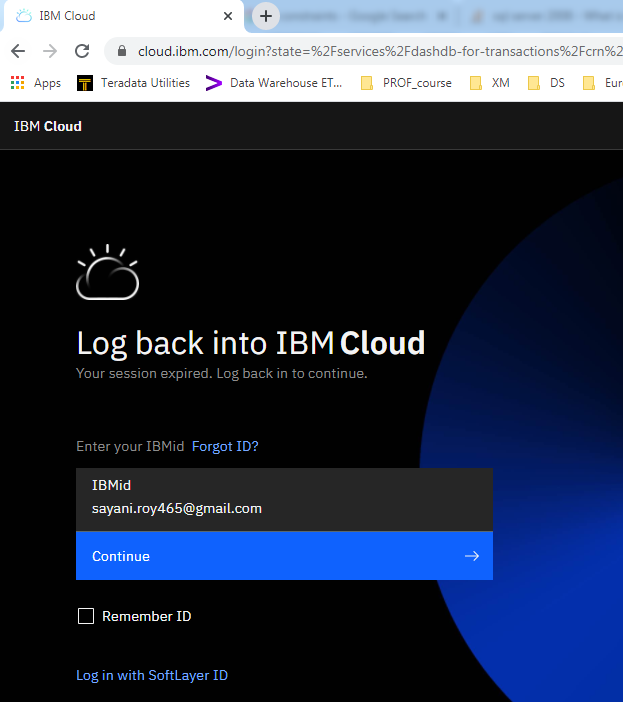
This table has all the details of items along with cost price, sale price and item availability.

I have prepared **dummy** data for data visualization. Below is the list of .csv files prepared by me for each corresponding table.

|  |  |
| --- | --- |
| **Table Name** | **Source File Name** |
| membership\_type | membership\_type.csv |
| customer | customer.csv |
| customer\_preference | customer\_preference.csv |
| preference | preference.csv |
| campaign | campaign.csv |
| store | store.csv |
| order | order.csv |
| order\_list | order\_list.csv |
| order\_status | order\_status.csv |
| item | item.csv |

1. Data Execution :

I have used IBM cloud to build the data warehouse where the queries have been executed to analysis the dummy data.



1. Data Modeling :

Below are the query statements along with output for dummy data visualizations.

**Problem 1: Gross generated revenue by campaign/ store/ customer/ membership type.**

Gross revenues are calculated on basis of campaign, store, customer, membership type separately.

Gross generated revenue by **store**:

Store table and Order table have been joined based on store id to track the total revenue for each store. Order placing date should be greater than or same as store creation date.

SELECT

str.store\_id,

str.store\_name,

COALESCE(SUM(ord.total),0) revenue

FROM store str

LEFT JOIN order ord

ON ord.store\_id = str.store\_id

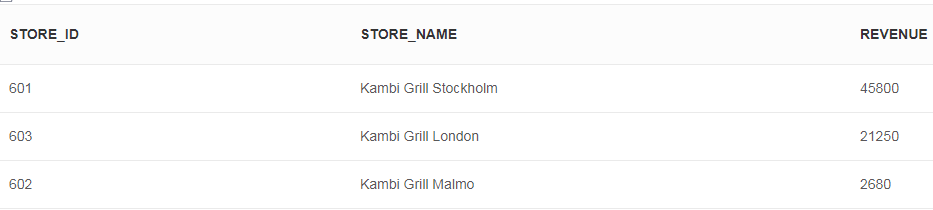
AND ord.created\_date >= str.created\_date

GROUP BY str.store\_id,str.store\_name

ORDER BY revenue DESC

;

Output with dummy data :



Gross generated revenue by **campaign:**

Campaign table and Order table have been joined based on Campaign id to track the total revenue for each Campaign. Order placing date should be greater than or same as Campaign creation date. Also the Campaign should not be expired.

SELECT

camp.campaign\_id,

camp.campaign\_name,

COALESCE(SUM(ord.total),0) revenue

FROM campaign camp

LEFT JOIN order ord

ON ord.campaign\_id = camp.campaign\_id

AND ord.created\_date >= camp.created\_date

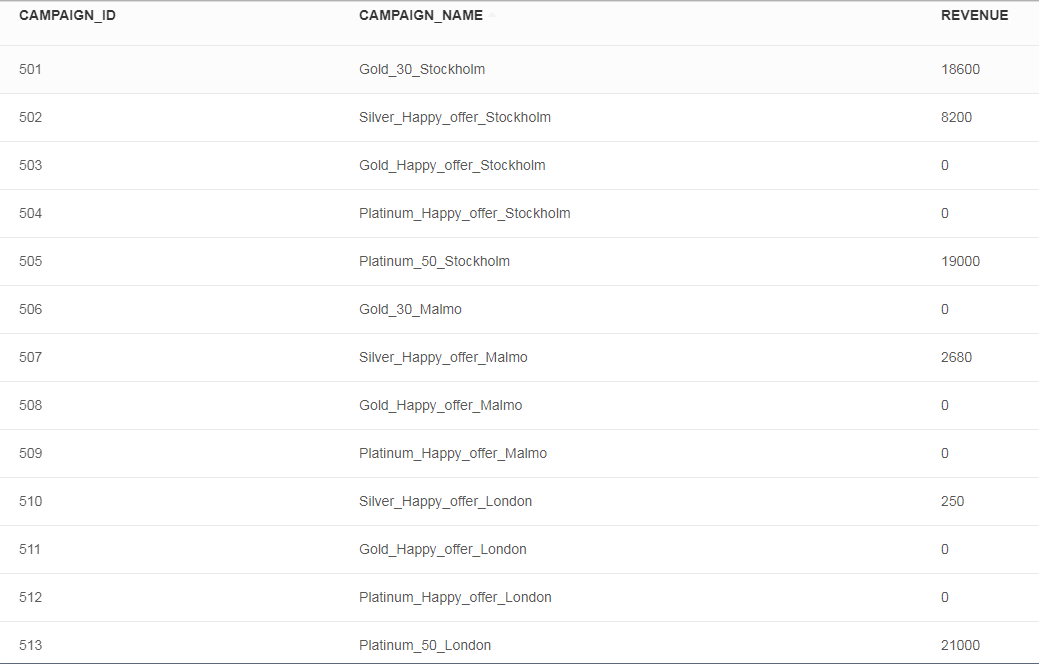
AND UPPER(camp.is\_expired) = 'NO'

GROUP BY camp.campaign\_id,camp.campaign\_name

ORDER BY camp.campaign\_id

;

Output with dummy data :



Gross generated revenue by **customer:**

Customer table and Order table have been joined based on Customer id to track the total revenue for each Customer. Order placing date should be greater than or same as Customer creation date.

SELECT

cust.first\_name,

cust.last\_name,

cust.loyalty\_number,

COALESCE(SUM(ord.total),0) revenue

FROM customer cust

LEFT JOIN order ord

ON ord.customer\_id = cust.customer\_id

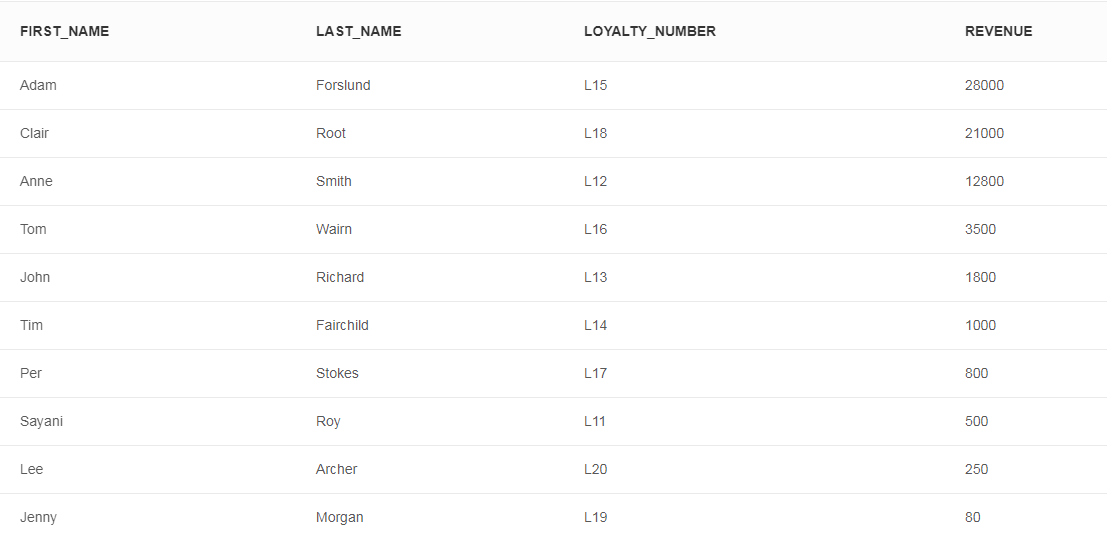
AND ord.created\_date >= cust.created\_date

GROUP BY cust.first\_name,cust.last\_name,cust.loyalty\_number

ORDER BY revenue DESC

;

Output with dummy data :



Gross generated revenue by **membership type:**

Membership type table, Customer table and Order table have been joined based on membership type id and customer id respectively to track the total revenue for each membership type. Order placing date should be greater than or same as Customer creation date.

SELECT

mem.membership\_type\_id,

mem.membership\_name,

COALESCE(SUM(ord.total),0) revenue

FROM membership\_type mem

INNER JOIN customer cust

ON mem.membership\_type\_id = cust.membership\_type\_id

LEFT JOIN order ord

ON ord.customer\_id = cust.customer\_id

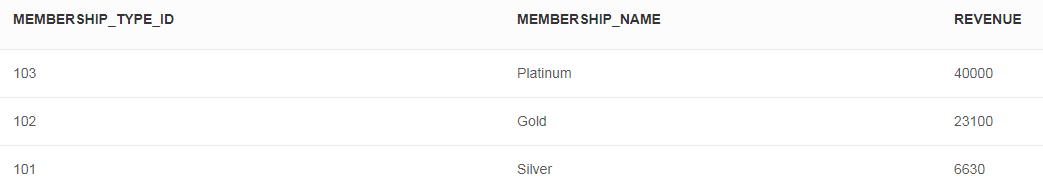
AND ord.created\_date >= cust.created\_date

GROUP BY mem.membership\_type\_id,mem.membership\_name

ORDER BY revenue DESC

;

Output with dummy data :



**Problem 2: How long until the first customer orders?**

Here data output will show how many days each customer took to place the first order. The used query can be described in two steps:

1. It will extract all ordering details for each customer along with days difference between customer creation and order placing.
2. Then Only the first transaction will be extracted

SELECT

cust\_data.first\_name,

cust\_data.last\_name,

cust\_data.loyalty\_number,

cust\_data.customer\_created\_date,

cust\_data.order\_created\_date,

cust\_data.Reqired\_Days

FROM

(

SELECT

cust.customer\_id,

cust.first\_name,

cust.last\_name,

cust.loyalty\_number,

cust.created\_date as customer\_created\_date,

ord.created\_date as order\_created\_date,

(DAYS(ord.created\_date) - DAYS(cust.created\_date)) AS Reqired\_Days,

ROW\_NUMBER() OVER (PARTITION BY first\_name,last\_name,loyalty\_number ORDER BY cust.created\_date) row\_num

FROM customer cust

INNER JOIN order ord

ON ord.customer\_id = cust.customer\_id

AND ord.created\_date >= cust.created\_date

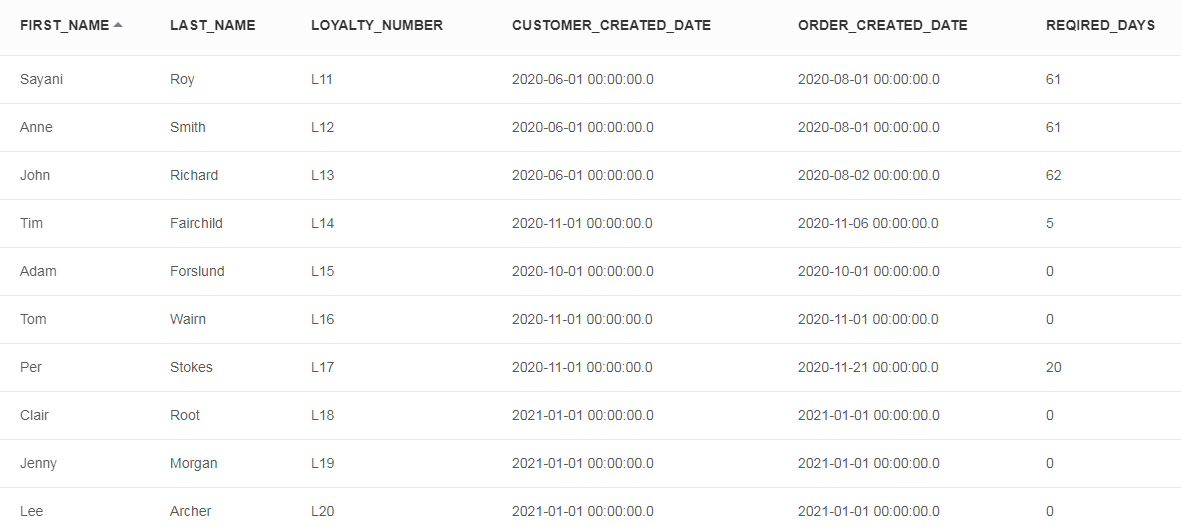
) cust\_data

WHERE cust\_data.row\_num = 1

ORDER BY cust\_data.loyalty\_number

;

Output with dummy data :



**Problem 3: What items are the most/least popular?**

All items from item table are joined with placed orders from order table so that the resultset can reflect how many quantity of the item has been sold. Here, order placing created date should be greater than or same as item creation date.

SELECT

Quantity.item\_name ,

COALESCE(SUM(Quantity.item\_quantity),0) Total\_Sold\_Quantity

FROM

(

SELECT

ord\_lst.order\_id,

itm.item\_id,

itm.item\_name,

ord\_lst.sale\_price ordered\_sale\_price,

itm.sale\_price item\_sale\_price,

ord\_lst.sale\_price/itm.sale\_price AS item\_quantity

FROM item itm

LEFT JOIN order\_list ord\_lst

ON itm.item\_id = ord\_lst.item\_id

AND ord\_lst.created\_date >= itm.created\_date

WHERE itm.parent\_item\_id is null

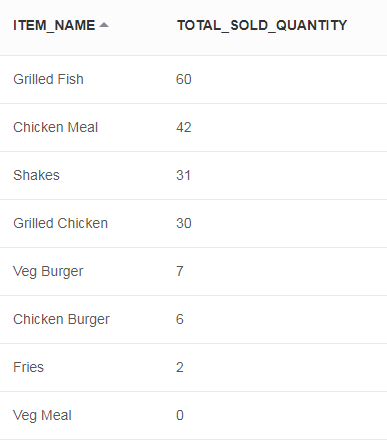
) Quantity

GROUP BY Quantity.item\_name

ORDER BY Total\_Sold\_Quantity DESC

;

Output with dummy data :

****

**Problem 4: What is our gross per sale?**

Here average gross revenue has been considered as gross earned money per sale or order. Gross revenue is without subtracting value of cost price for particular order. For that, column ‘total’ has been considered as it is showing the true transaction amount.

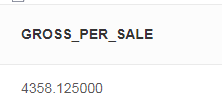
SELECT

AVG(total) Gross\_per\_sale

FROM order

;

Output with dummy data :



**Problem 5: What items should we remove due to customer preferences?**

All items from item table are joined with placed orders from order table so that the resultset can reflect how many quantity of the item has been sold. Here, order placing created date should be greater than or same as item creation date.

Those Items can be removed from the menu which are not at all popular or sold.

SELECT

Quantity.item\_name ,

COALESCE(SUM(Quantity.item\_quantity),0) Total\_Sold\_Quantity

FROM

(

SELECT

ord\_lst.order\_id,

itm.item\_id,

itm.item\_name,

ord\_lst.sale\_price ordered\_sale\_price,

itm.sale\_price item\_sale\_price,

ord\_lst.sale\_price/itm.sale\_price AS item\_quantity

FROM item itm

LEFT JOIN order\_list ord\_lst

ON itm.item\_id = ord\_lst.item\_id

AND ord\_lst.created\_date >= itm.created\_date

WHERE itm.parent\_item\_id is null

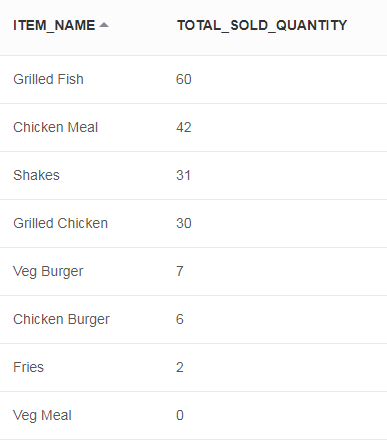
) Quantity

GROUP BY Quantity.item\_name

ORDER BY Total\_Sold\_Quantity DESC

;

Output with dummy data :



**Problem 6: How can we improve targeting of marketing campaigns?**

Campaign table and Order table have been joined based on Campaign id to track the total revenue and number of item sold for each Campaign. Order placing date should be greater than or same as Campaign creation date. Also the Campaign should not be expired.

Now, we can analyse the data to understand the success or failure rate for each campaign to improve our marketing campaign strategy.

SELECT

camp.campaign\_id,

camp.campaign\_name,

COALESCE(COUNT(ord.order\_id),0) Number\_of\_Orders,

COALESCE(SUM(ord.total),0) revenue

FROM campaign camp

LEFT JOIN order ord

ON ord.campaign\_id = camp.campaign\_id

AND ord.created\_date >= camp.created\_date

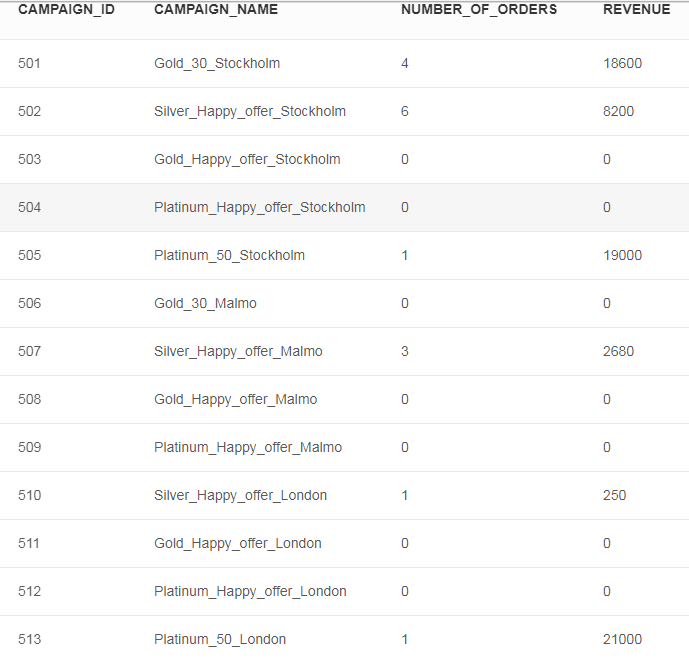
AND UPPER(camp.is\_expired) = 'NO'

GROUP BY camp.campaign\_id,camp.campaign\_name

ORDER BY camp.campaign\_id

;

Output with dummy data :



**Problem 7: How long does it take for an order to be processed/delivered?**

Order and order status tables have been joined to fetch all placed order details along with the status. As the updated\_date column will be updated with any changes in order status. So, difference between creation date and updated date field value will show the awaited time.

SELECT

ord.order\_id,

ord\_St.order\_status\_name,

ord\_St.updated\_date,

ord\_St.created\_date,

(MINUTES\_BETWEEN(ord\_St.updated\_date,ord\_St.created\_date)) AS Minutes\_Taken

FROM order ord

INNER JOIN order\_status ord\_St

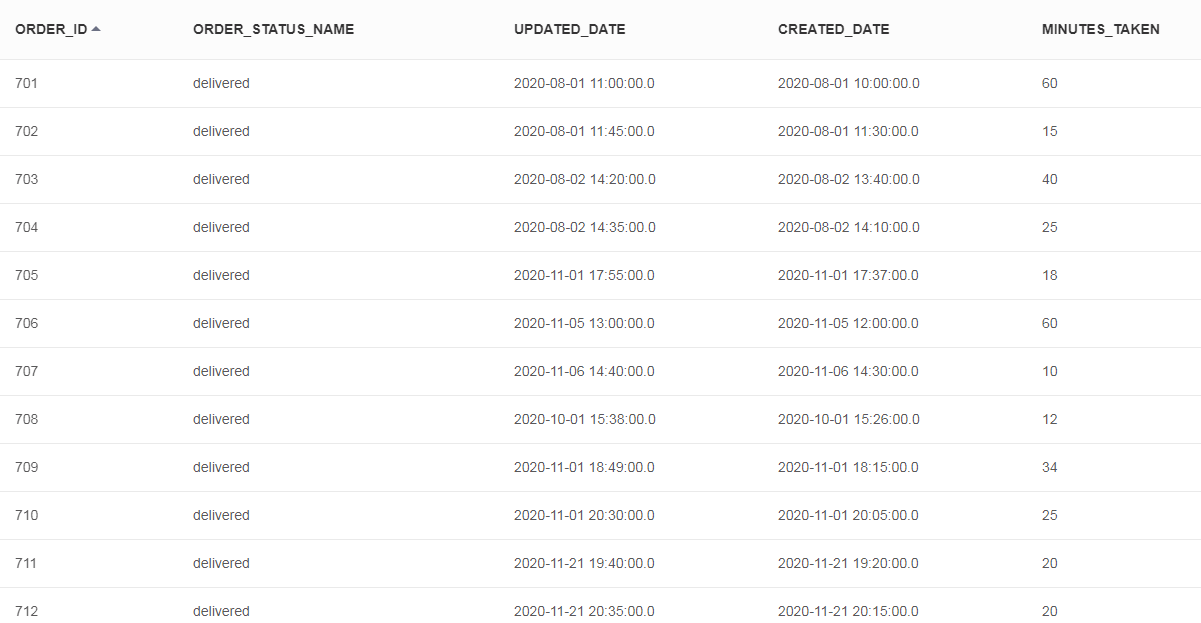
ON ord.order\_status\_id = ord\_St.order\_status\_id

AND LOWER(ord\_St.order\_status\_name) IN ('in progress', 'delivered')

ORDER BY ord.order\_id

;

Output with dummy data :



**Problem 8: How long does it take for customers to add their first preference?**

Here data output will show how many days each customer took to add their first preference. The used query can be described in two steps:

1. It will extract all preference details for each customer along with days difference between customer creation and preference addition.
2. Then Only the first transaction will be extracted

SELECT

cust\_data.first\_name,

cust\_data.last\_name,

cust\_data.loyalty\_number,

cust\_data.customer\_created\_date,

cust\_data.cust\_pref\_created\_date,

cust\_data.Reqired\_Days

FROM

(

SELECT

cust.customer\_id,

cust.first\_name,

cust.last\_name,

cust.loyalty\_number,

cust.created\_date as customer\_created\_date,

cust\_pref.created\_date as cust\_pref\_created\_date,

(DAYS(cust\_pref.created\_date) - DAYS(cust.created\_date)) AS Reqired\_Days,

ROW\_NUMBER() OVER (PARTITION BY cust.first\_name,cust.last\_name,cust.loyalty\_number ORDER BY cust.created\_date) row\_num

FROM customer cust

LEFT JOIN customer\_preference cust\_pref

ON cust\_pref.customer\_id = cust.customer\_id

AND cust\_pref.created\_date >= cust.created\_date

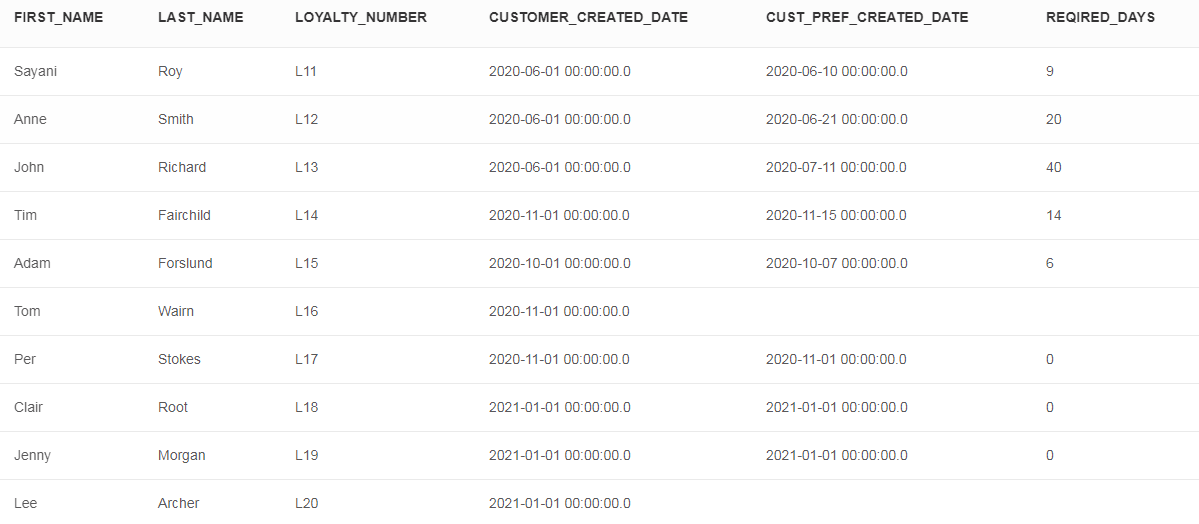
) cust\_data

WHERE cust\_data.row\_num = 1

ORDER BY cust\_data.loyalty\_number

;

Output with dummy data :



**Problem 9: How long does it take for customers to improve their membership?**

Here data output will show how many days each customer took to be upgraded. The used query can be described in two steps:

1. It will extract all details for each customer by self joining to extract those customer details who has several entries for membership improvement.
2. Membership type table has been joined to show membership type name to which the customer is upgraded.

SELECT

cust\_data.cust1\_first\_name AS first\_name,

cust\_data.cust1\_last\_name AS last\_name,

cust\_data.cust1\_loyalty\_number AS loyalty\_number,

cust\_data.cust2\_created\_date AS upgrade\_date,

mem.membership\_name AS upgraded\_membership,

cust\_data.days\_taken

FROM

(

SELECT

cust1.first\_name as cust1\_first\_name,

cust1.last\_name as cust1\_last\_name,

cust1.loyalty\_number as cust1\_loyalty\_number,

cust1.membership\_type\_id as cust1\_membership\_type\_id,

cust1.created\_date as cust1\_created\_date,

cust2.first\_name as cust2\_first\_name,

cust2.last\_name as cust2\_last\_name,

cust2.loyalty\_number as cust2\_loyalty\_number,

cust2.membership\_type\_id as cust2\_membership\_type\_id,

cust2.created\_date as cust2\_created\_date,

DAYS\_BETWEEN(cust2.created\_date,cust1.created\_date) days\_taken,

ROW\_NUMBER() OVER (PARTITION BY cust1.first\_name,cust1.last\_name,cust1.loyalty\_number,cust1.membership\_type\_id ORDER BY cust2.membership\_type\_id) row\_num

FROM customer cust1

INNER JOIN customer cust2

ON cust1.first\_name = cust2.first\_name

AND cust1.last\_name = cust2.last\_name

AND cust1.loyalty\_number = cust2.loyalty\_number

AND cust2.membership\_type\_id > cust1.membership\_type\_id

AND cust2.created\_date > cust1.created\_date

) cust\_data

INNER JOIN membership\_type mem

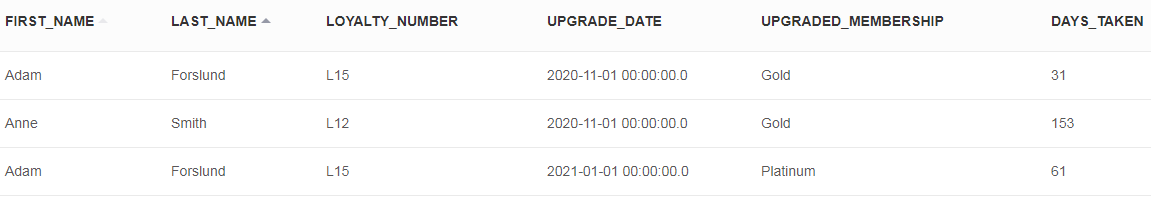
ON mem.membership\_type\_id = cust\_data.cust2\_membership\_type\_id

WHERE cust\_data.row\_num = 1

ORDER BY cust\_data.cust2\_created\_date

;

Output with dummy data :



**Problem 10: How many campaigns does it take to improve a customer's membership?**

Here data output will show how many campaigned each customer utilized to be upgraded. The used query can be described in four steps:

1. It will extract all details for each customer by self joining to extract those customer details who has several entries for membership improvement.
2. Membership type table has been joined to show membership type name to which the customer is upgraded.
3. Order table has been joined to count utilized campaign for each customer
4. **One count** has been increase to consider the transaction on the day when customer was upgraded (Assumed, the customer has used a campaign on the day he/she got upgraded).

SELECT

cust\_data.cust1\_first\_name AS first\_name,

cust\_data.cust1\_last\_name AS last\_name,

cust\_data.cust1\_loyalty\_number AS loyalty\_number,

cust\_data.cust2\_created\_date AS upgrade\_date,

mem.membership\_name AS upgraded\_membership,

(COUNT(ord.order\_id)+1) AS Num\_of\_Campaign\_for\_Upgradation

FROM

(

SELECT

cust1.customer\_id as cust1\_customer\_id,

cust1.first\_name as cust1\_first\_name,

cust1.last\_name as cust1\_last\_name,

cust1.loyalty\_number as cust1\_loyalty\_number,

cust1.membership\_type\_id as cust1\_membership\_type\_id,

cust1.created\_date as cust1\_created\_date,

cust2.customer\_id as cust2\_customer\_id,

cust2.first\_name as cust2\_first\_name,

cust2.last\_name as cust2\_last\_name,

cust2.loyalty\_number as cust2\_loyalty\_number,

cust2.membership\_type\_id as cust2\_membership\_type\_id,

cust2.created\_date as cust2\_created\_date,

ROW\_NUMBER() OVER (PARTITION BY cust1.first\_name,cust1.last\_name,cust1.loyalty\_number,cust1.membership\_type\_id ORDER BY cust2.membership\_type\_id) row\_num

FROM customer cust1

INNER JOIN customer cust2

ON cust1.first\_name = cust2.first\_name

AND cust1.last\_name = cust2.last\_name

AND cust1.loyalty\_number = cust2.loyalty\_number

AND cust2.membership\_type\_id > cust1.membership\_type\_id

AND cust2.created\_date > cust1.created\_date

) cust\_data

INNER JOIN membership\_type mem

ON mem.membership\_type\_id = cust\_data.cust2\_membership\_type\_id

INNER JOIN order ord

ON cust\_data.cust1\_customer\_id = ord.customer\_id

AND ord.campaign\_id IS NOT NULL

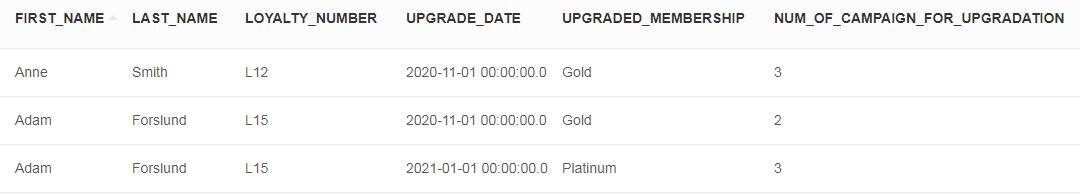
WHERE cust\_data.row\_num = 1

GROUP BY cust\_data.cust1\_first\_name, cust\_data.cust1\_last\_name, cust\_data.cust1\_loyalty\_number, cust\_data.cust2\_created\_date, mem.membership\_name

ORDER BY cust\_data.cust1\_loyalty\_number

;

Output with dummy data :



**Problem 11: Which store sees the most campaign action?**

This data analysis can be done with two different ways as describe below.

1. Store table and Order table have been joined based on store id to track the number transactions and the total revenue for each store. Order placing date should be greater than or same as store creation date.

SELECT

str.store\_id,

str.store\_name,

COALESCE(COUNT(ord.order\_id),0) Used\_Campaign\_Number,

COALESCE(SUM(ord.total),0) Earned\_Revenue

FROM store str

LEFT JOIN order ord

ON ord.store\_id = str.store\_id

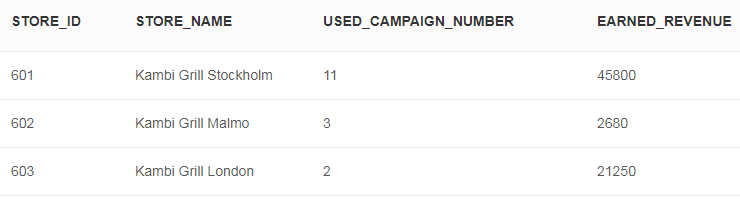
AND ord.campaign\_id IS NOT NULL

GROUP BY str.store\_id,str.store\_name

ORDER BY str.store\_id

;

Output with dummy data :



1. Store table, Order table, customer table and membership type have been joined to extract customer membership details so that number to different memberships can be counted for each store.

This result will show the impact of the campaigns by counting the presence of high range customer memberships for each store.

SELECT

cust\_data.store\_id,

cust\_data.store\_name,

cust\_data.membership\_name,

COUNT(1) Membership\_count

FROM

(

SELECT

str.store\_id,

str.store\_name,

cust.first\_name,

cust.last\_name,

cust.loyalty\_number,

mem.membership\_name,

ROW\_NUMBER() OVER (PARTITION BY str.store\_id, str.store\_name, cust.first\_name, cust.last\_name, cust.loyalty\_number ORDER BY cust.membership\_type\_id desc) row\_num

FROM store str

INNER JOIN order ord

ON ord.store\_id = str.store\_id

INNER JOIN customer cust

ON ord.customer\_id = cust.customer\_id

INNER JOIN membership\_type mem

ON cust.membership\_type\_id = mem.membership\_type\_id

) cust\_data

WHERE cust\_data.row\_num = 1

GROUP BY cust\_data.store\_id, cust\_data.store\_name, cust\_data.membership\_name

ORDER BY cust\_data.store\_id,cust\_data.membership\_name

;

Output with dummy data :



**Problem 12: Can we create new "menu meals" to improve sales?**

This data analysis can be done with two different ways as describe below.

1. Preference can be extracted from customer preference table. If coustomer has changed his/her preference then that will be counted.

Based on that, highly liked preferences can be considered for new menu. Suppose, if many customers prefer to add cheese then we can thing about any cheese overloaded item.

SELECT

pref.preference\_name,

pref.preference\_type,

COUNT(cust\_pref.preference\_id) Preference\_Count

FROM preference pref

LEFT JOIN

(

SELECT

cust.first\_name,

cust.last\_name,

cust.loyalty\_number,

cust\_pref.preference\_id

FROM customer\_preference cust\_pref

INNER JOIN customer cust

ON cust.customer\_id = cust\_pref.customer\_id

GROUP BY cust.first\_name, cust.last\_name, cust.loyalty\_number, cust\_pref.preference\_id

) cust\_pref

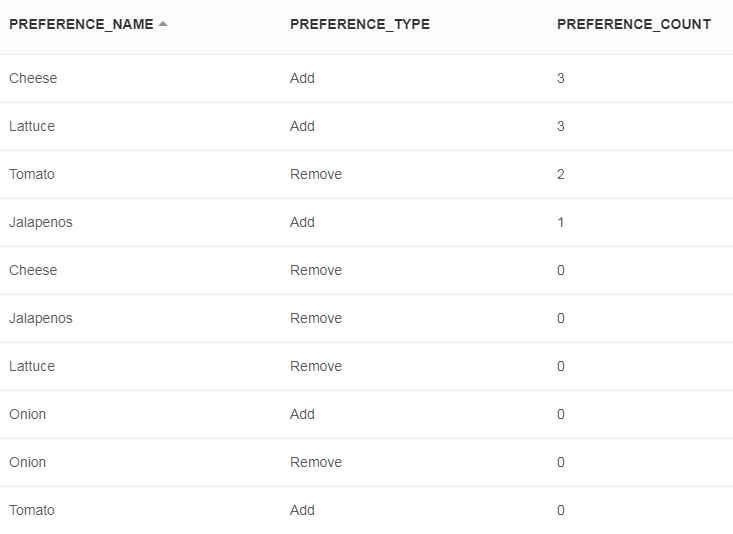
ON pref.preference\_id = cust\_pref.preference\_id

GROUP BY pref.preference\_name, pref.preference\_type

ORDER BY Preference\_Count DESC

;

Output with dummy data :



1. All items from item table are joined with placed orders from order table so that the resultset can reflect how many quantity of the item has been sold. Then new menu can be designed as per the popularity of item. Suppose for below resultset, grilled fish and shakes are enough popular comparing chicken meal. If these two items are combined to serve new meal item, that may help to increase sales.

SELECT

Quantity.item\_name ,

COALESCE(SUM(Quantity.item\_quantity),0) Total\_Sold\_Quantity

FROM

(

SELECT

ord\_lst.order\_id,

itm.item\_id,

itm.item\_name,

ord\_lst.sale\_price ordered\_sale\_price,

itm.sale\_price item\_sale\_price,

ord\_lst.sale\_price/itm.sale\_price AS item\_quantity

FROM item itm

LEFT JOIN order\_list ord\_lst

ON itm.item\_id = ord\_lst.item\_id

AND ord\_lst.created\_date >= itm.created\_date

WHERE itm.parent\_item\_id is null

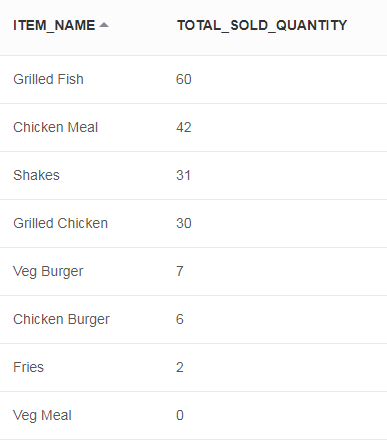
) Quantity

GROUP BY Quantity.item\_name

ORDER BY Total\_Sold\_Quantity DESC

;

Output with dummy data :



**Problem 13: We want to measure the value of membership types as members return to ensure we have the correct levels**

This data analysis can be done with two different ways as describe below.

1. Membership type table, Customer table and Order table have been joined based on membership type id and customer id respectively to show the count of order transactions and total revenue for each membership type. Order placing date should be greater than or same as Customer creation date.

SELECT

mem.membership\_type\_id,

mem.membership\_name,

COALESCE(COUNT(ord.order\_id),0) Number\_of\_Orders,

COALESCE(SUM(ord.total),0) revenue

FROM membership\_type mem

INNER JOIN customer cust

ON mem.membership\_type\_id = cust.membership\_type\_id

LEFT JOIN order ord

ON ord.customer\_id = cust.customer\_id

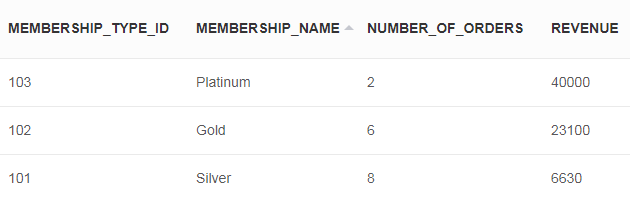
AND ord.created\_date >= cust.created\_date

GROUP BY mem.membership\_type\_id,mem.membership\_name

ORDER BY revenue DESC

;

Output with dummy data :



1. Membership type table, Customer table and Order table have been joined based on membership type id and customer id respectively to show the count of order transactions and total revenue for each customer in each membership level. It will help to analyse membership type success rates more prominently.

SELECT

cust.first\_name,

cust.last\_name,

cust.loyalty\_number,

mem.membership\_name,

COALESCE(COUNT(ord.order\_id),0) Number\_of\_Orders,

COALESCE(SUM(ord.total),0) revenue

FROM customer cust

INNER JOIN membership\_type mem

ON cust.membership\_type\_id = mem.membership\_type\_id

LEFT JOIN order ord

ON ord.customer\_id = cust.customer\_id

AND ord.created\_date >= cust.created\_date

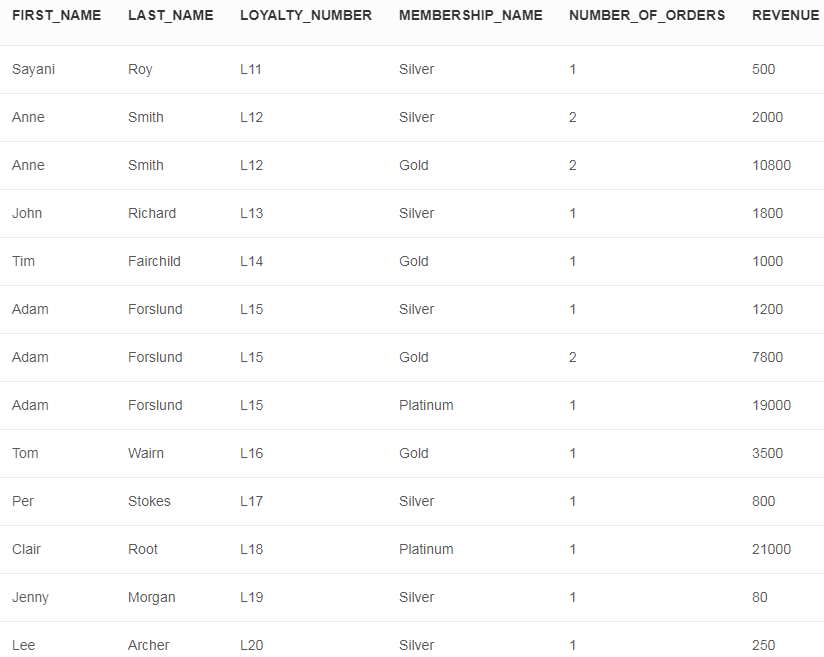
AND ord.created\_date BETWEEN cust.membership\_valid\_from\_date and cust.membership\_valid\_to\_date

GROUP BY cust.customer\_id, cust.first\_name, cust.last\_name, cust.loyalty\_number, mem.membership\_type\_id, mem.membership\_name

ORDER BY cust.loyalty\_number

;

Output with dummy data :



1. Conclusion:

In the above sections, I have extracted data from different data databases and integrated in a single data warehouse. Different business scenarios are analysed with this integrated data model which helped to find out the scope of further improvements.