Reports Done at Behtar(LMIH Technologies)

Reporting Manager:
Ashish Bajpai.

Done by:
Bona Swathi
Business Analyst Intern
swathibona@gmail.com
8762883698

Contents:

28) Behtar Inventory_amount report
   a) Warehouse_wise
   b) Store_wise
29) Landing_Price& Tax
30) Shrinkage Report
a)Store_wise
b)Warehouse_wise
31)Warehouseinventoryamount_quater_wise
32)Stock transfer with seller_id and Supplier_id
33) Sku_correction
   a) Sku_correction with barcode, store_id, cashier_login
   b) Sku_correction with sp, mrp, unit
34)Top 50 SP with barcodes
35) Sales demand prediction model using python.
36) Margin by article
37) Articles not being sold at the store
   a) where not_sold for != null
   b) where not_sold for = null

1. **Ageing Report:** Ageing and not being sold items and made to tell the importance of expiry_date data.

Ageing Report (redash.io)

a)Warehouse wise: This report contains the data of the items which are lying in the warehouse by not being sent to the store

Objective: To dispatch the items which are coming into the warehouse based on inwarding date or batch wise

Approach: Taken the Warehouse_warehouseinventory(WWI) table(where the warehouse inventory data exists) left joined with order_storewarehouseorderitem(OSWI -where the stock transfer data from warehouse to stock exists) and given a condition where sent_qty is NULL, means, if sent_qty is null then the items are not sent to any store and they are lying in the warehouse from the past days.

Results:

- ➢ In this report, we can separately check the ageing items in each warehouse and in each warehouse_id, with the help of filters.
- ➢ This report contains the item_name,article_name, warehouse_name&id, unit, grammage, article_code, Barcode, along with the nonmovingdays- which tells from how many days the item is not moving from the warehouse.
- ➢ This report is for bangalore city.

Benefits:

- ➢ This Kind of data is helpful in dispatching the items fastly, before expiry_date from the warehouse to store.
- ➢ This data helps in avoiding the goods damage because of expiry
- ➢ This helps in reducing the loss by dispatching the items fast before it gets expire

—– Items which are lying in the warehouse not being sent to store– bangalore

```
select
date(wwi.created_on)  as DATE,
w.name as "WAREHOUSE_NAME::filter",
–w.name as WAREHOUSE_NAME,
–wwi.sku_id,
WWi.warehouse_id as "warehouse_id::filter",
–wwi.warehouse_id,
wwi.non_moving,
–osw.invoice_number,
–osw.store_id,

sart.article_code as ARTICLE_CODE,
sku.barcode as BARCODE,
sart.title as ARTICLE_NAME,
sart.grammage as GRAMMAGE,
sart.unit as UNIT,
sku.title as SKU_NAME,
sku.city_id,
cs.title as SECTION,
–poc.name,
DATEDIFF(day, CAST(wwi.created_on as date), current_date)+1 AS NON_MOVINGDAYS,
oswi.sent_qty,
oswi.quantity
from "mercuryproduction_public_warehouse_warehouseinventory" wwi
left outer JOIN "mercuryproduction_public_order_storewarehouseorderitem" oswi on
(wwi.sku_id = oswi.sku_id)
left join "mercuryproduction_public_order_storewarehouseorder" osw on
osw.id=oswi.store_warehouse_order_id
left join "mercuryproduction_public_warehouse_warehouse" w on wwi.warehouse_id =
w.id
left join "mercuryproduction_public_sku_items_sku" sku on wwi.sku_id = sku.id
left join "mercuryproduction_public_catalog_section" cs on cs.id= sku.section_id
left join "mercuryproduction_public_sku_items_regionarticle"srart on srart.id=
sku.article_id
left join "mercuryproduction_public_sku_items_article" sart on sart.id= srart.article_id
left join "mercuryproduction_public_purchase_orders_company" poc on
poc.id=sku.company_id
left join "mercuryproduction_public_catalog_toplevel" ct on sart.top_level_id=ct.id
where oswi.sent_qty  is null
and sku.city_id='1'
–and ct.title<>'Fresh'
–and osw.order_type='A'
–) a
order by date DESC
```

b)Store Wise: This report contains the data of the items which are lying in the store by not being sold.

Objective: To sell the items which are coming into the store against the batch or which are inwarded first into the store.

Approach: Taken the store_storeinventory table, with the help of last_sold column found from how many days the items are not selling from the store.

Results:

➢ In this report, we can separately check the ageing items in each store and in each store_id with the help of filters.

➢ This report contains the item_name,article_name, store_name&id, unit, grammage, article_code, Barcode, along with the notsolddays- which tells from how many days the item is not selling from the store

➢ This report is for bangalore city.

—Items which are lying in the store by not being sold— Bangalore

```
select
s.name as "Store_Name::filter",
–s.name as store,
–s.id as store_id,
s.id as "store_id::filter",
art.article_code,
sku.barcode,
art.title as article_name,
CASE WHEN sku.grammage is not null THEN (sku.title || ' ' || sku.grammage ||' ' ||
sku.unit) ELSE sku.title END as sku_name,
s_inv.quantity,
sku.sp,
s_inv.quantity*sku.sp as item_total,
s.city_id as city_id,
CAST(last_sold as date),
DATEDIFF(day, CAST(last_sold as date), current_date)+1 AS Not_Sold_For
from mercuryproduction_public_store_storeinventory s_inv
left join mercuryproduction_public_store_store s on s.id=s_inv.store_id
left join mercuryproduction_public_sku_items_sku sku on sku.id = s_inv.sku_id
left join mercuryproduction_public_sku_items_regionarticle rart on rart.id=sku.article_id
left join mercuryproduction_public_sku_items_article art on art.id = rart.article_id

where s_inv.quantity > 0
and s.id != 5
and s.city_id ='1'

order by 2 asc, 3 asc nulls last
```

Benefits:

- ➢ This Kind of data is helpful in selling the items fastly, before expiry_date from the store.
- ➢ This data helps in avoiding the goods damage because of expiry
- ➢ This helps in reducing the loss by selling the items first which are expiring soon

2. Stock Transfer from warehouse to store: This report contains the data of the stock which are transferred from warehouse to store.
Stock transfer from warehouse to store (redash.io)

Objective: To find the items which are transferring from warehouse to store.

Approach: Taken the order_storewarehouseorder and order_storewarehouseorderitem table by giving the condition sent_qty>0, so that only sent items from warehouse to store will record.

Results: In this report, we can see from when the stock transferred from warehouse and when the stock reached to store with the warehouse and stock information and how much quantity is sent from warehouse to store.

Benefits : With this information, we will know how much stock is transferring from warehouse to store monthly, based on that, we can make the decisions how much stock we should purchase for the next month.

```
–transfered goods from warehouse to store

select CAST(Created_Date as date) as created,
CAST(Fulfilled_Date as date) as fulfilled,
CAST(Store_Received_Date as date) as store_received,
invoice_number,
warehouse as "warehouse_name::filter",
warehouse_id,
store,
store_id,
article_code,
barcode,
article_name,
sku_name,
grammage,
unit,
sent_quantity,
item_total,
section,
status


from
(
select
o.created_on as Created_Date,
o.fullfilled_on as Fulfilled_Date,
o.store_receive_timestamp as Store_Received_Date,
o.invoice_number,
s.name as store,
s.id as store_id,
art.article_code as article_code,
sku.barcode as barcode,
art.title as article_name,
sku.title as sku_name,
art.grammage as grammage,
art.unit as unit,
oi.sent_qty as sent_quantity,
–oi.quantity as requested_quantity,
oi.item_total as item_total,
w.name as warehouse,
w.id as warehouse_id,
cs.title as section,
o.status
from order_storewarehouseorder o
left join order_storewarehouseorderitem oi on o.id=oi.store_warehouse_order_id
left join sku_items_sku sku on sku.id=oi.sku_id
left join store_store s on s.id = o.store_id
left join warehouse_warehouse w on w.id = o.warehouse_id
left join sku_items_regionarticle rart on rart.id=oi.article_id
left join sku_items_article art on rart.article_id=art.id
left join catalog_section cs on cs.id=sku.section_id
where s.status = 'A'
and sent_qty > 0
–and w.id=9
and CAST(o.created_on as date) between '{{daterange.start}}' and '{{daterange.end}}'
) a

order by 1 desc nulls last
```

3. Sales Order Returns: This report shows the returns from the sales

Sales order returns for october (redash.io)

Objective: to find the items which are returning from the sales

Approach: Taken order_storeorder and order_orderreturn tables and join sku_items_sku with the barcode, so that only the returned items will record.

Results: This report contains the details of the items along with the bill_number, store_name, how many no. of items are returned, its value and the article_code.

Benefits: With this report, we can get the information of how many items are returning and we can check why they are returning, so that we can try to minimize them for the next month.

```
–sales order returns with article_code and barcode(from order_return json file)
select
ss.name,
date(o.created_on) as "order_date",
o.order_number as "Bill Number",
o.value as "order value",
date(r.created_on) as "returned_date",
r.items,
r.id,
r.value,
r.total_returned_items,
r.MRP_ITEM,
r.ORDERBARCODE,
sia.article_code
from (
select
created_on,
items,
value,
id,
order_id,
json_array_length(items::json) as total_returned_items,
jsonb_array_elements(items)->>'mrp' as MRP_ITEM ,
jsonb_array_elements(items)->>'barcode' as ORDERBARCODE
from order_orderreturn
) r
left outer join order_storeorder o on o.id=r.order_id
left outer join store_store ss on o.store_id=ss.id
left outer join sku_items_sku skuitems on skuitems.barcode = r.ORDERBARCODE
left outer join sku_items_regionarticle sir on skuitems.article_id=sir.id AND
sir.city_id=ss.city_id
left outer join sku_items_article sia on sir.article_id=sia.id
where sia.article_code is not null and
CAST(r.created_on as DATE) between '{{daterange.start}}' and '{{daterange.end}}'
and o.is_duplicate ='false'
```

4. Fraud Detection in offline sale: This report contains the details of the items of fraud where the items sold_quantity is > 10 in offline.
Fraud detection for offline sale (redash.io)

Objective: To find fraud items in offline sales.

Approach:

- ➢ To find the offline sale items, given a condition like where mobile number is NULL in analytics_storeorderitems(sales)
- ➢ Fraud usually happens in bulk, so given the condition quantity >10.

Results: This reports contains the sales items information in offline sale where the quantity>10, along with the store name, article_code and barcode

Benefits: With this report we can check where the frauds are happening in offline sales and we can take steps to minimise those offline frauds.

```
—fraud detection in offline sale where mobilenumber is null and quantity>10
select CAST(sales.timestamp as date) as _date,
cast((sales.timestamp + '5:30:00') as timestamp),
s.name as store,
bill.order_number as invoice_number,
us.mobile,
sales.article_code,
sales.barcode,
art.title as article_name,
sku.title as sku_name,
sku.grammage,
sku.unit,
sku.mrp,
sales.sp,
sales.quantity,
ct.title as LO_top_level,
cs.title as L1_section,
cc.title as L2_category,
csb.title as L3_subcategory,
poc.name as company_name,
bill.is_duplicate

from analytics_storeorderitems sales
left join order_storeorder bill on bill.id=sales.order_id
left join store_store s on s.id = sales.store_id
left join sku_items_sku sku on sku.id = sales.sku_id
left join sku_items_article art on sales.article_code = art.article_code

left join purchase_orders_company poc on poc.id = art.company_id
left join catalog_section cs on cs.id = art.section_id
left join catalog_category cc on cc.id = art.category_id
left join catalog_subcategory csb on csb.id = art.subcategory_id
left join catalog_toplevel ct on ct.id = art.top_level_id
left join users_behtaruser us on bill.user_id=us.id

where CAST(sales.timestamp as date) between '{{DateRange.start}}' and
'{{DateRange.end}}'
–and bill.is_duplicate ='false'
and s.status = 'A'
–and s.city_id = 1
–and (s.id=19 or s.id=32 or s.id=47 or s.id=59)
and us.mobile isnull  and quantity > 10

order by 1 desc nulls last
```

5.Multiple Additions and Deductions of Inventory: Report to track multiple inventory additions or deductions on a daily basis.
Duplicate deductions and additions of inventory (redash.io)

Approach: To Find the duplicate rows which are entered multiple times, snapshot_storeinventoryarticlecodechanges table was taken and used the COUNT function to find the duplicate rows.

Results:

➢ This report contains the details of the items which are entered multiple times, along with the information of invoice, article_code, mode, countof(how many no of duplicate rows).
➢ The invoices of storeorderdeletion and storeauditcleanup is null because in the data itself, they are null.

```
—Duplicate rows of additions and deductions of inventory
—some INVOICES are null for storeorderdeletion and storeauditcleanup
```

```
SELECT
  date(created_on) as created, invoice, article_code, COUNT(id) AS countOf, mode,
from_qty, to_qty, store_id
   FROM snapshot_storeinventoryarticlecodechanges
    where CAST(created_on as date) between '{{daterange.start}}' and '{{daterange.end}}'
   GROUP BY  date(created_on), invoice, article_code, mode, from_qty, to_qty, store_id
   HAVING COUNT(id) >1 order by created


  – CAST(oswi.created_on as date) between '{{daterange.start}}' and '{{daterange.end}}'
```

6. Count of Multiple Coupons: This report contains the details of the items where the coupons are applied multiple times with the help of coupon_applied column and count function.
count of multiple coupons (redash.io)

```
–count of multiple coupons for november for 17 days
—select * from order_storeorder where coupon_applied=true limit 10

select  created_on, store_id, order_number, count(order_number) from
(select id,
value,
store_id,
created_on ,
order_number,
payment_mode ,
jsonb_array_elements(items)->>'mrp' as Barcode_mrp,
json_array_length(items::json) as total_line_items_in_bill,
–json_array_length(items::json->'mrp') as total_line_items_in_bill,
extra_data->>'mrp' as total_bill_amount,
extra_data->>'tax' as Bill_Tax,
extra_data->>'savings' as behtar_savings,
extra_data->'breakup'->'tax'->'gst' as gst,
extra_data->'breakup'->'tax'->'cess' as cess,
coupon_applied
from order_storeorder os
where
is_duplicate ='false'
and created_on BETWEEN NOW() - INTERVAL '18 DAY' AND NOW()
–and jsonb_array_elements(items)->>'mrp'  <= 0
–order by created_on desc ;)
) table1
where CAST(Barcode_mrp as FLOAT) <= 0
and coupon_applied = true
group  by created_on, store_id, order_number having count(order_number)>1
order by count(order_number) desc ;
```

7. Returned items of multiple coupons applied:

multiple coupons returns (redash.io)

Objective: This report is mainly done to find the items or people who have applied multiple coupons while purchasing and returning the items with the intention of crediting the extra amount into their accounts.

Approach: Taken the order_storeorder and order_orderreturn table and used the coupon_applied column and count function to find the returned items where the coupons are applied multiple times.

Results:

➢ The report is showing the returned items with the count of coupons applied along with bill_number.

Benefits : we can check how many coupons are applying and how much of money we are losing with the multiple coupons and we can take actions to minimize the loss.

```
—people who have returned the items after applying multiple coupons with the intention of crediting of amount into their accounts
select  created_on, store_id, order_number, count(order_number) from
(select o.id,
o.value,
o.store_id,
o.created_on ,
o.order_number,
o.payment_mode ,
jsonb_array_elements(o.items)->>'mrp' as Barcode_mrp,
json_array_length(o.items::json) as total_line_items_in_bill,
–json_array_length(items::json->'mrp') as total_line_items_in_bill,
extra_data->>'mrp' as total_bill_amount,
extra_data->>'tax' as Bill_Tax,
extra_data->>'savings' as behtar_savings,
extra_data->'breakup'->'tax'->'gst' as gst,
extra_data->'breakup'->'tax'->'cess' as cess,
coupon_applied
from order_storeorder o
left join order_orderreturn r on r.order_id = o.id
where
o.created_on BETWEEN NOW() - INTERVAL '18 DAY' AND NOW()
and o.is_duplicate ='false'
–and jsonb_array_elements(items)->>'mrp'  <= 0
–order by created_on desc ;)
) table1
where
CAST(Barcode_mrp as FLOAT) <= 0
and coupon_applied = true
group  by created_on, store_id, order_number having count(order_number)>1
order by count(order_number) desc
```

8. Order Details of the items: This report contains the order details of the last one year
order details with barcode from oct 2019 to oct 2020 (redash.io)

```
-- order details of past 1 year
select id,
value,
store_id,
created_on ,
order_number,
payment_mode ,
–jsonb_array_elements(items)->>'mrp' as Barcode_mrp,
json_array_length(items::json) as total_line_items_in_bill,
–json_array_length(items::json->'mrp') as total_line_items_in_bill,
extra_data->>'mrp' as total_bill_amount,
extra_data->>'tax' as Bill_Tax,
extra_data->>'savings' as behtar_savings,
extra_data->'breakup'->'tax'->'gst' as gst,
extra_data->'breakup'->'tax'->'cess' as cess
from order_storeorder where
created_on >= '2019-10-01'
and created_on < '2020-10-31'
and is_duplicate ='false'
order by created_on desc
limit 10
–select count(distinct(created_on)) from order_storeorder where created_on >= '2019-10-01'
–and created_on < '2020-10-31'
```

9. Sale Return with High Price than Purchase Price: A daily report to identify the cases across all stores where an item was purchased at a price and was returned at a higher price(whether it might be bill buster, coupon applied, discount offered)

sale return with high price (redash.io)

Objective: To find the items which are purchased at one price and returned at higher price. This report is mainly done to find where the fraud is happening like buying the items with discount, or coupon applied,or billbuster and returning the same items with the intention of crediting the extra amount into their accounts.

Approach: Taken the order_returnorder, order_storeorder and analyticsstoreorderitems tables by giving the filter of returnunitprice > orderunitprice.

Results: This report contains the details of the items where the returnunitprice>orderunitprice along with sold_quantity, returnedquantity, sales_sp, coupon applied or not.

Benefits: It helps in finding the frauds in returns

```
—-a daily report where the items are purchased at one price and returned at higher price
select
date,
–id,
article_code,
orderid,
store_id,
barcode,
store_name,
invoice,
mrp_item,
sales_sp,
orderunitprice,
returnunitprice,
returned_value,
quantity,
sold_quantity,
coupon_appliedornot,
B
from(
select
date(r.created_on) as date,
–sales.timestamp as date,
sales.article_code,
sales.sp as sales_sp,
sales.store_id,
sales.barcode,
sales.quantity as sold_quantity,
–sales.id,
so.value as returned_value,
s.name as store_name,
so.id as orderid,
so.order_number as invoice,
so.mrp_item,
so.quantity,
so.orderunitprice as orderunitprice,
r.returnunitprice,
so.coupon_applied as coupon_appliedornot,
–sales.sp=((CASE WHEN (sales_sp < so.unitprice) 1 ) ELSE 0
–case when sales.sp < 'so.unitprice' then 1 else 0 end
CASE WHEN cast(r.returnunitprice as numeric(6,2)) > cast(so.orderunitprice as
numeric(6,2)) THEN 1 ELSE 0 END AS B
–so.return_sp
from
(
select
created_on,
items,
value,
id,
store_id,
order_number,
coupon_applied,
jsonb_array_elements(items)->>'mrp' as MRP_ITEM,
jsonb_array_elements(items)->>'quantity' as quantity,
jsonb_array_elements(items)->>'barcode' as ORDERBARCODE,
jsonb_array_elements(items)->>'unitPrice' as orderunitprice
from order_storeorder
) so
left join
(select
order_id,
created_on,
jsonb_array_elements(items)->>'unitPrice' as returnunitprice from
order_orderreturn) r on r.order_id = so.id
–left join order_storeorder so on so.id = sales.order_id
left join analytics_storeorderitems sales on sales.order_id=so.id and sales.barcode =
so.orderbarcode
left outer join store_store s on so.store_id=s.id
left join sku_items_sku sku on sku.id=sales.sku_id
–where CAST(sales.timestamp as date)
where CAST(r.created_on as date) between '{{DateRange.start}}' and '{{DateRange.end}}'
and s.city_id=1
and s.status='A'
and sku.city_id=1
)a
where B = '1'
–and sales_sp >0
–and B = '1'
group by 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16

order by date
—-select * from analytics_storeorderitems where id ='13460364' or id='13422273' or
id='13460363' or id ='13489681' or id = '13523808'
```

10. Inward & Outward: This report shows the inward & Outward inventory of the warehouse.

Redash

a) **Inward:** This report contains the data of the stock which are coming into the warehouse daily

Approach: Used the grn_goodsreceiptenote, grn_grnitems, sku_items_sku tables to find the stock which is coming into the warehouse daily.

Results: Report contains the details of the grn data along with the filter for date and warehouse_name.

Benefits: with this data, the operational and marketing team can make the comparison between how much the stock is coming into the warehouse and how much the stock is going out of the warehouse and they can take some decisions like how much stock needs to buy or sell for the next month.

```
select
date as "grndate",
–date as created_on,
grn_number,
status,
article_code,
article_title,
quantity,
item_total,
grammage,
unit,
landing_price,
sp,
mrp,
scheme_discount,
basic_price,
basic_total,
tax_value,
tax_total,
gst,
LO_top_level,
L1_section,
L2_category,
L3_subcategory,
company_name,
brand,
warehouse_name as "warehouse_name",
vendor_id,
supplier,
Po_Number,
PO_qty

from
(
select
  CAST(grn.created_on as date) as date
  , grn.grn_number
  , grn.status
–   , grn.total_value
  , grni.article_code
  , ar.title as article_title
  , grni.quantity
  , grni.item_total as item_total
  , grni.landing_price as landing_price
  , grni.unit as unit
  , grni.sp as sp
  , grni.mrp as mrp
  , grni.scheme_discount as scheme_discount
  , grni.basic_price as basic_price
  , grni.basic_total as basic_total
  , grni.tax_total as tax_total
  , grni.tax_value as tax_value
  , grni.gst as gst
  , grni.grammage as grammage
  , ct.title as LO_top_level
  , cs.title as L1_section
  , cc.title as L2_category
  , csb.title as L3_subcategory
  , poc.name as company_name
  ,cb.title as Brand

  , grn.supplier_invoice_number
  , grn.warehouse_id
  , w.name as warehouse_name
  , ps.supplier_code as vendor_id
  , Ps.name as supplier
  , grn.purchase_order_id as Po_Number
  ,pi.quantity as PO_qty
from
"GRN_goodsrecieptnote" grn
left join "GRN_grnitems" grni on
  grni.grn_id = grn.id
left join warehouse_warehouse w on
  w.id = grn.warehouse_id


  left join sku_items_sku sku on
    sku.id = grni.sku_id
  left join sku_items_regionarticle r on
    r.id = sku.article_id
  left join sku_items_article ar on
    ar.id = r.article_id

  left join purchase_orders_company poc on
    poc.id = ar.company_id

  left join catalog_section cs on
    cs.id = ar.section_id
  left join catalog_category cc on
    cc.id = ar.category_id
  left join catalog_subcategory csb on
    csb.id = ar.subcategory_id
  left join catalog_toplevel ct on
    ct.id = cs.top_level
    left join "purchase_orders_purchaseorders" p on
    p.number=grn.purchase_order_id
    left join purchase_orders_purchaseorderitems pi on pi.purchase_orders_id=p.number
    left join "purchase_orders_supplier" ps on
    ps.id= p.supplier_id
    left join "catalog_brand" cb on cb.id=ar.brand_id


  where Cast(grn.created_on as date) between '{{ daterange.start }}'and
'{{ daterange.end }}'
and pi.article_code=grni.article_code
  ) a
where quantity > 0
order by 22 desc
```

-

b) **Outward:** This report contains the data of the stock which are going out of the warehouse daily.

Approach: Used the orde_storewarehouseorder, order_storewarehouseorderitem, sku_items_sku tables to find the stock which is going out of the warehouse daily.

Results: Report contains the details of the stock transfer data from warehouse to store along with the filter for date and warehouse_name.

Benefits: with this data, the operational and marketing team can make the comparison between how much the stock is coming into the warehouse and how much the stock is going out of the warehouse and they can take some decisions like how much stock needs to buy or sell for the next month.

–transfered goods from warehouse to store

```
select CAST(Created_Date as date) as created,
CAST(Fulfilled_Date as date) as fulfilled,
CAST(Store_Received_Date as date) as store_received,
invoice_number,
warehouse as "warehouse_name::filter",
warehouse_id,
store,
store_id,
article_code,
barcode,
article_name,
sku_name,
grammage,
unit,
sent_quantity,
item_total,
section,
status


from
(
select
o.created_on as Created_Date,
o.fullfilled_on as Fulfilled_Date,
o.store_receive_timestamp as Store_Received_Date,
o.invoice_number,
s.name as store,
s.id as store_id,
art.article_code as article_code,
sku.barcode as barcode,
art.title as article_name,
sku.title as sku_name,
art.grammage as grammage,
art.unit as unit,
oi.sent_qty as sent_quantity,
–oi.quantity as requested_quantity,
oi.item_total as item_total,
w.name as warehouse,
w.id as warehouse_id,
cs.title as section,
o.status

from order_storewarehouseorder o
left join order_storewarehouseorderitem oi on o.id=oi.store_warehouse_order_id
left join sku_items_sku sku on sku.id=oi.sku_id
left join store_store s on s.id = o.store_id
left join warehouse_warehouse w on w.id = o.warehouse_id
left join sku_items_regionarticle rart on rart.id=oi.article_id
left join sku_items_article art on rart.article_id=art.id
left join catalog_section cs on cs.id=sku.section_id
where s.status = 'A'
and sent_qty > 0
–and w.id=9
and CAST(o.created_on as date) between '{{daterange.start}}' and '{{daterange.end}}'
) a

order by 1 desc nulls last
```

11. Clear cart

clear cart (redash.io)

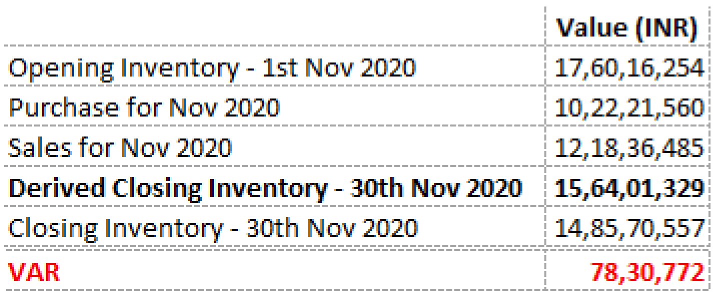
a) StoreID, Countof clearcarts with value zero, count of clearcarts with value > 0
b) Storeid, userlogin, count of clearcarts, in past 15 days
c) Storeid, cartvalue, clearcart, discount/coupon applied, in desc order for the last 24 hours.

```
–StoreID, Countof clearcarts with value zero, count of clearcarts with value > 0
select
storeid,
countofclearcart
from(
select
event_details ->'storeId' as storeid,
count(event_name)  as countofclearcart
from analytics_analyticsevents aa
where event_name  = 'clearCart'
and event_details ->'discount' is not null
group by 1)ae
where countofclearcart >='0'
order by countofclearcart asc


—2. Storeid, userlogin, count of clearcarts, in past 15 days
select
```

```
created_on,
storeid,
countofclearcart,
userlogin
from(
select
date(created_on) as created_on,
event_details ->'storeId' as storeid,
event_details ->'email' as userlogin,
count(event_name)  as countofclearcart
from analytics_analyticsevents aa
where event_name  = 'clearCart'
and event_details ->'discount' is not null
and date(created_on) BETWEEN NOW() - INTERVAL '15 DAY' AND NOW()
group by 1,2,3)ae
where countofclearcart >='0'
order by created_on desc
```

12. Inventory Gap: Op/stock inventory + purchase inventory-sales = derived c/stock -actual c/stock = var: This report is mainly done to know the difference between actual closing stock and the derived closing stock, which helps in finding the variance in stock at the warehouse level and store level.
[Inventory Gap(Warehouse_wise) (redash.io)]



	Value (INR)
Opening Inventory - 1st Nov 2020	17,60,16,254
Purchase for Nov 2020	10,22,21,560
Sales for Nov 2020	12,18,36,485
Derived Closing Inventory - 30th Nov 2020	**15,64,01,329**
Closing Inventory - 30th Nov 2020	14,85,70,557
VAR	**78,30,772**

a) **Warehouse_wise:** This report contains the stock difference at warehouse level

Objective: To know the difference between derived closing inventory and closing inventory at warehouse level

Approach: Taken the opening and closing inventory from warehouse_warehousesnapshotarticle article table, inward from purchase_GRN table and Outward(outward sku movement which means the stock transferred from warehouse to store) from orderstorewarehouseorder table and applied a formula to find the stock difference.

(openingvalue+inward_value-stocktransfer_value)-warehouseactual_closingvalue) as wareinventory_difference.

Results:

- ➢ This report contains the openining_stock, closing stock, inward, outward stock of the warehouse for every month or every day.

Benefits:

- ➢ With this data, we can see the difference between the actual and derived closing stock and can check where the difference is happening in the stock at warehouse level.

```
— opening_stock + purchases(GRN)-Stock transfer= derived closing_stock-actual
closing_stock= wareinventory_diff
select
created_date,
WAREHOUSE_ID as "WAREHOUSE_ID::filter",
opening_stock,
closing_stock,
PURCHASE_GRN,
stocktransfer_ST,
wareinventory_diff
from
(select
coalesce(w._date,inward.counted_on,st._date) as created_date,
coalesce(w.warehouse_id,inward.warehouse_id,st.id) as WAREHOUSE_ID,
coalesce(w.openingvalue,0.00) as opening_stock,
coalesce(w.closingvalue,0.00) as closing_stock,
coalesce(inward.value,0.00) as PURCHASE_GRN,
–coalesce(hold.value,0.00) as IN_TRANSIT_TO_STORE,
coalesce(st.value,0.00) as stocktransfer_ST,
((w.openingvalue+inward.value-st.value)-w.closingvalue) as wareinventory_diff
```

```
from
(select winv._date,
warehouse_id,
sum(winv.opening*db.sp) as openingvalue,
sum(winv.closing*db.sp) as closingvalue
from
(select CAST(wsa.date as date) as _date,
warehouse_id,
wsa.article_code,
sum(wsa.start_quantity) as opening,
sum(wsa.end_quantity) as closing

from warehouse_warehousesnapshotarticle wsa
left join warehouse_warehouse w on w.id=wsa.warehouse_id
where w.city_id=1
and (warehouse_id = '6' or warehouse_id = '9')

–order by "date" desc nulls last
group by 1,2,3
)winv
left join

(
select
–sku.barcode,
art.article_code,
–art.title as article_name,
–poc.name as company,
–brand.title as brand,
–ct.title as top_level,
–cs.title as section,
–cc.title as category,
–csb.title as sub_category,
–art.gst,
–art.hsn,
avg(sku.mrp) as mrp,
avg(sku.sp) as sp
```

```
from sku_items_sku sku
left join sku_items_regionarticle rart on rart.id = sku.article_id
left join sku_items_article art on art.id = rart.article_id

–left join purchase_orders_company poc on poc.id = art.company_id
–left join purchase_orders_companybrandmapping pocbm on pocbm.company_id =
poc.id
–left join catalog_brand brand on brand.id = pocbm.brand_id

–left join catalog_section cs on cs.id = art.section_id
–left join catalog_category cc on cc.id = art.category_id
–left join catalog_subcategory csb on csb.id = art.subcategory_id
–left join catalog_toplevel ct on ct.id = cs.top_level

where sku.city_id = 1

group by 1
–and sku.sp>0
)db on db.article_code=winv.article_code

group by 1,2

order by 1 desc nulls last
)w



–_________________-_________________–ST–_________________

left join



(—–outward  sku movement which means the stock transfered from warehouse to store
select a._date,
a.id,
a.value
from
(select CAST(os.created_on as date) as _date,
–os.invoice_number,
–os.store_id,
–s.name as store_name,
–w.name as warehouse,
–w.name,
w.id,
–sart.article_code,
–sku.barcode,
–sku.title as Description,
–osi.quantity as required,
–osi.sent_qty,
sum(osi.sent_qty*sku.sp) as value
–cs.title as section,
–osi.dms,
–osi.not_available,
–osi.store_inventory,
–osi.warehouse_inventory,
–poc.name

from
"order_storewarehouseorder" os
left join "order_storewarehouseorderitem" osi on os.id = osi.store_warehouse_order_id
left join "sku_items_sku" sku on osi.sku_id=sku.id
left join "store_store" s on os.store_id=s.id
left join "warehouse_warehouse" w on os.warehouse_id = w.id
left join "catalog_section" cs on cs.id= sku.section_id
left join "sku_items_regionarticle"srart on srart.id= sku.article_id
left join "sku_items_article" sart on sart.id= srart.article_id
left join "purchase_orders_company" poc on poc.id=sku.company_id
left join "catalog_toplevel" ct on sart.top_level_id=ct.id

–and s.city_id = '1'
–and sku.city_id='1'
–and ct.title<>'Fresh'
where os.order_type='A'
group by 1,2
order by 1 desc
)a
```

```
)st on st._date=w._date and st.id=w.warehouse_id – and st.id=hold.warehouse_id and
st._date=hold._date


—__________—________-–________–Inward–_________

left join

(select CAST(grn.counted_on as date) as counted_on,
–current_date,
–current_date - 10,
–w.name as warehouse,
grn.warehouse_id as warehouse_id,
–grnit.article_code  as article_code,
–art.title  as article,
sum(grnit.quantity*sku.sp) as value
–sum(grnit.quantity) as quantity,
–sum(grnit.item_total) as total_value

from "GRN_goodsrecieptnote" grn
left join "GRN_grnitems" grnit on grn.id = grnit.grn_id
left join warehouse_warehouse w on w.id=grn.warehouse_id

left join sku_items_sku sku on sku.id=grnit.sku_id
left join sku_items_regionarticle rart on rart.id=sku.article_id
left join sku_items_article art on rart.article_id=art.id

left join purchase_orders_company poc on poc.id = art.company_id

left join catalog_section cs on cs.id = art.section_id
left join catalog_category cc on cc.id = art.category_id
left join catalog_subcategory csb on csb.id = art.subcategory_id
left join catalog_toplevel ct on ct.id = art.top_level_id

where grnit.quantity > 0
–and w.city_id=1
–and w.status='A'
–and sku.city_id=1
–and (w.id='6' or w.id = '9')
```

```
group by 1,2

)inward on w._date=inward.counted_on and inward.warehouse_id=w.warehouse_id
where (coalesce(w.warehouse_id,inward.warehouse_id,st.id) = 6 or
coalesce(w.warehouse_id,inward.warehouse_id,st.id) = 9)
)total

where CAST(total.created_date as DATE) between '{{daterange.start}}' and
'{{daterange.end}}'
order by 1 desc
```

b) <u>Store_wise</u> : This report contains the stock difference at Store level

<u>Objective</u>: To know the difference between derived closing inventory and closing inventory at storelevel.

<u>Approach</u>: Taken the opening and closing inventory from snapshot_storeinventorysnapshotdaily, inward(stock transfer from warehouse to store) from order_storewarehouseorder table and Outward(sales at store_level) from analytics_storeorderitemstable and applied a formula to find the stock difference.

(openingvalue+inward_value-sales_value)-storeactual_closingvalue) as store_difference.

<u>Results:</u>

- ➢ This report contains the openining_stock, closing stock, inward, outward stock of the store for every month or every day.

<u>Benefits:</u>

- ➢ With this data, we can see the difference between the actual and derived closing stock and can check where the difference is happening in the stock at store level.

```
—opening_stock + inward(stock_transferfrom warehouse to store)-sales=
derived_closing_stock - actual cl/stock = store_inven_var
select
_date,
Store_ops,
store_stvalue,
store_salesvalue,
Store_cs as store_cs,
store_inven_var
from

(select coalesce(s._date,st._date,sales._date) as _date,
–coalesce(w.value,0.00) as Warehouse_SOH,
coalesce(s.opening_value,0.00) as Store_ops,
coalesce(s.closing_value,0.00) as Store_cs,
coalesce(st.stvalue,0.00) as store_stvalue,
coalesce(sales.salesvalue) as store_salesvalue,
–coalesce(hold.value,0.00) as IntermediateStoreHold
(s.opening_value +st.stvalue-sales.salesvalue-s.closing_value) as store_inven_var
from(
select inv._date,
–inv.store_id,
–inv.store_name,
–inv.article_code,
–inv.barcode,
sum(inv.opening*db.sp) as Opening_Value,
sum(inv.closing*db.sp) as Closing_Value

from
(
select coalesce(op._date, cl._date) as _date,
coalesce(op.store_id, cl.store_id) as store_id,
coalesce(op.store, cl.store) as store_name,
coalesce(op.barcode, cl.barcode) as barcode,
coalesce(op.article_code,cl.article_code) as article_code,
coalesce(op.end,0.00) as opening,
coalesce(cl.end,0.00) as closing

from
(select sis.date+1 as _date,
sis.barcode,
art.article_code,
sis.end,
sis.store_id,
s.name as store
from mercuryproduction_public_snapshot_storeinventorysnapshotdaily sis
left join mercuryproduction_public_store_store  s on s.id = sis.store_id
left join mercuryproduction_public_sku_items_sku  sku on sku.barcode = sis.barcode
left join mercuryproduction_public_sku_items_regionarticle  rart on rart.id=sku.article_id
left join mercuryproduction_public_sku_items_article  art on art.id = rart.article_id

left join mercuryproduction_public_purchase_orders_company  poc on poc.id =
art.company_id

left join mercuryproduction_public_catalog_section  cs on cs.id = art.section_id
left join mercuryproduction_public_catalog_category  cc on cc.id = art.category_id
left join mercuryproduction_public_catalog_subcategory  csb on csb.id =
art.subcategory_id
left join mercuryproduction_public_catalog_toplevel ct on ct.id = art.top_level_id

where
s.city_id =1
and s.status='A'
and sku.city_id=1
and s.id!='84'
and sis.end >-10000
and sis.end <10000

–sis.store_id  = 1

)op

full outer join

(select sis.date as _date,
sis.barcode,
art.article_code,
sis.end,
sis.store_id,
s.name as store
from mercuryproduction_public_snapshot_storeinventorysnapshotdaily sis
left join mercuryproduction_public_store_store  s on s.id = sis.store_id
left join mercuryproduction_public_sku_items_sku sku on sku.barcode = sis.barcode
left join mercuryproduction_public_sku_items_regionarticle rart on rart.id=sku.article_id
left join mercuryproduction_public_sku_items_article art on art.id = rart.article_id

left join mercuryproduction_public_purchase_orders_company poc on poc.id =
art.company_id

left join mercuryproduction_public_catalog_section cs on cs.id = art.section_id
left join mercuryproduction_public_catalog_category cc on cc.id = art.category_id
left join mercuryproduction_public_catalog_subcategory csb on csb.id =
art.subcategory_id
left join mercuryproduction_public_catalog_toplevel ct on ct.id = art.top_level_id

where
s.city_id =1
and s.status='A'
and sku.city_id=1
and s.id!='84'
and sis.end >-10000
and sis.end <10000
–where sis.store_id = 1
)cl on op.barcode=cl.barcode and op.store_id=cl.store_id and op._date=cl._date

order by 3 asc, 1 asc nulls last

)inv

left join

(
select
sku.barcode,
art.article_code,
(art.title || ' ' || art.grammage || ' ' || art.unit) as article_name,
poc.name as company,
brand.title as brand,
ct.title as top_level,
cs.title as section,
cc.title as category,
csb.title as sub_category,
art.gst,
art.hsn,
avg(sku.mrp) as mrp,
avg(sku.sp) as sp


from mercuryproduction_public_sku_items_sku sku
left join mercuryproduction_public_sku_items_regionarticle rart on rart.id =
sku.article_id
left join mercuryproduction_public_sku_items_article art on art.id = rart.article_id

left join mercuryproduction_public_purchase_orders_company poc on poc.id =
art.company_id
left join mercuryproduction_public_purchase_orders_companybrandmapping pocbm on
pocbm.company_id = poc.id
left join mercuryproduction_public_catalog_brand brand on brand.id = pocbm.brand_id

left join mercuryproduction_public_catalog_section cs on cs.id = art.section_id
left join mercuryproduction_public_catalog_category cc on cc.id = art.category_id
left join mercuryproduction_public_catalog_subcategory csb on csb.id =
art.subcategory_id
left join mercuryproduction_public_catalog_toplevel ct on ct.id = cs.top_level

where sku.city_id = 1

group by 1,2,3,4,5,6,7,8,9,10,11
–and sku.sp>0
)db on db.barcode=inv.barcode

group by 1

order by 1 desc nulls last
)s

—___________________________stock transferinward___________________-
left join

(select a._date,
–a.id,
a.value as stvalue
from
(select CAST(store_receive_timestamp as date) as _date,
–st.store_id,
sum(osi.sent_qty*sku.sp) as value

from
mercuryproduction_public_order_storewarehouseorder os
left join mercuryproduction_public_order_storewarehouseorderitem osi on os.id =
osi.store_warehouse_order_id
left join mercuryproduction_public_sku_items_sku sku on osi.sku_id=sku.id
left join mercuryproduction_public_store_store s on os.store_id=s.id
left join mercuryproduction_public_warehouse_warehouse w on os.warehouse_id = w.id
left join mercuryproduction_public_catalog_section cs on cs.id= sku.section_id
left join mercuryproduction_public_sku_items_regionarticle srart on srart.id=
sku.article_id
left join mercuryproduction_public_sku_items_article sart on sart.id= srart.article_id
left join mercuryproduction_public_purchase_orders_company poc on
poc.id=sku.company_id
left join mercuryproduction_public_catalog_toplevel ct on sart.top_level_id=ct.id

–and s.city_id = '1'
–and sku.city_id='1'
–and ct.title<>'Fresh'
where os.order_type='A'
group by 1
order by 1 desc
)a
)st on st._date=s._date

—__________________________sales___________________-
left join

(select CAST(sales.timestamp as date) as _date,
–cast((sales.timestamp + '5:30:00') as timestamp),
–s.name as store,
–s.id as store_id,
sum(sales.sp*sales.quantity) as salesvalue
```

```
from mercuryproduction_public_analytics_storeorderitems sales
left join mercuryproduction_public_order_storeorder bill on bill.id=sales.order_id
left join mercuryproduction_public_store_store s on s.id = sales.store_id
left join mercuryproduction_public_sku_items_sku sku on sku.id = sales.sku_id
left join mercuryproduction_public_sku_items_article art on sales.article_code =
art.article_code

left join mercuryproduction_public_purchase_orders_company poc on poc.id =
art.company_id

left join mercuryproduction_public_catalog_section cs on cs.id = art.section_id
left join mercuryproduction_public_catalog_category cc on cc.id = art.category_id
left join mercuryproduction_public_catalog_subcategory csb on csb.id =
art.subcategory_id
left join mercuryproduction_public_catalog_toplevel ct on ct.id = art.top_level_id
left join mercuryproduction_public_users_behtaruser us on bill.user_id=us.id

where  s.status = 'A'
–and bill.is_duplicate ='false'
group by 1
order by 1 desc nulls last
)sales on st._date = sales._date
–sales.store_id=st.store_id  and st.store_received=sales._date–and
sales.barcode=st.barcode
order by 1 desc
)total
where CAST(total._date as DATE) between '{{daterange.start}}' and '{{daterange.end}}'
order by 1 desc
```

13. Redash Clean Up: Archived the queries which has the name of New Query, practice query and long queries and unpublished queries and queries which are not in use after confirming with the creator.

14. Category_wise share in Overall sales along with ABV and Bill_Cuts: This report contains the data of all stores each category share and category sales in overall sales along with AVB and Bill_cuts.

[category share in overallsales with ABV, Bill_cuts (redash.io)]()

Approach: created subqueries for overall sales and sales by category_wise and bill_cuts, mainly using tables analytics_storeorderitems, order_storeorder and catlog_toplevel tables.

Results : category_wise sales, share % in overall sales in each store_id along with ABV and Bill_cuts.

Benefits : Through this, we can find which category is contributing more and which is less contributing to the sales and in which category ABV and Bill cuts are occurring more.

```
select
tb3.store_id as "store_id::filter",
tb3.category,
tb3.category_sales,
tb3.category_share,
tb6.avgabv,
tb6.bill_cuts


from
(select
tb2.store_id,
tb2.category,
sum(tb2.total) as category_sales,
sum(tb2.total)/sum(tb1.grandtotal)*100  as category_share

from
(select
sales2.store_id,
sum(sp*quantity) as  grandtotal
from analytics_storeorderitems as sales2  where sales2.quantity<888888
and sales2.timestamp BETWEEN NOW() - INTERVAL '90 DAY' AND NOW() group by
1)tb1

left join

(select
sales.store_id,
ct.title as category,
sum(sales.sp*sales.quantity) as total
from analytics_storeorderitems sales
left join order_storeorder bill on bill.id=sales.order_id
left join store_store s on s.id = sales.store_id
left join sku_items_sku sku on sku.id = sales.sku_id
left join sku_items_article art on sales.article_code = art.article_code
left join purchase_orders_company poc on poc.id = art.company_id
left join catalog_section cs on cs.id = art.section_id
left join catalog_category cc on cc.id = art.category_id
left join catalog_subcategory csb on csb.id = art.subcategory_id
left join catalog_toplevel ct on ct.id = art.top_level_id
where sales.quantity<888888888
and bill.is_duplicate ='false'
and sales.timestamp BETWEEN NOW() - INTERVAL '90 DAY' AND NOW()
group by 1,2)tb2 on tb1.store_id = tb2.store_id
group by 1,2)tb3

left join

(select
store_id,
title,
bill_cuts,
avg(abv) as avgabv

from(select
store_id,
title,
tb4.bill_cuts,
tb4.total_sale/tb4.bill_cuts::float as ABV

from
(select
sales3.store_id,
ct3.title,
sum(os.value) as total_sale,
count(os.value) as bill_cuts
from order_storeorder os
left join analytics_storeorderitems sales3 on sales3.order_id = os.id
left join store_store s3 on s3.id = sales3.store_id
left join sku_items_sku sku3 on sku3.id = sales3.sku_id
left join sku_items_article art3 on sales3.article_code = art3.article_code

left join purchase_orders_company poc3 on poc3.id = art3.company_id

left join catalog_section cs3 on cs3.id = art3.section_id
left join catalog_category cc3 on cc3.id = art3.category_id
left join catalog_subcategory csb3 on csb3.id = art3.subcategory_id
left join catalog_toplevel ct3 on ct3.id = art3.top_level_id
where sales3.timestamp BETWEEN NOW() - INTERVAL '90 DAY' AND NOW()
and os.is_duplicate ='false' group by 1,2 )as tb4

) tb5
group by 1,2,3)tb6 on tb6.store_id = tb3.store_id and tb6.title= tb3.category

group by 1,2,3,4,5,6
```

15. Current Selling Price details(after discount): This report made to know the current selling price, mrp and discount offers to the product.

Current Selling Price details(after discounts) (redash.io)

Approach: Used the tables sku_items_multiplemrpandsp and offers_discountofferarticle to know the selling price and mrp for the products

- There are discount offers i.e. percentage and value, so used two different formulas for each to know the actual sold price after subtracting the discount from the selling price.
- discount_type ='PERCENTAGE' THEN sp-(sp*value/100)
- discount_type = 'VALUE' THEN (sp-value) else 0

Results: The results were showing the city wise : article_code and barcode wise selling price , mrp discount_type and actualsellingprice(after removing discount value).

Benefits: The prices will change almost every day and this must be properly communicated to the teams and customers to ensure transparency in operations. This report will be helpful in these kinds of situations.

```
select
sku.city_id as "city_id::filter",
date(msp.updated_on) as updated_on,
art.article_code,
art.title as article_name,
cb.title as "BRAND",
ct.title as "L0 CATEGORY",
cs.title as "L1 CATEGORY",
cc.title as "L2 CATEGORY",
csb.title as "L3 CATEGORY",
sku.barcode,
msp.mrp,
msp.sp,
odoa.discount_type,
odoa.value,
CASE WHEN odoa.discount_type ='PERCENTAGE' THEN msp.sp-
(msp.sp*odoa.value/100)
WHEN odoa.discount_type = 'VALUE' THEN (msp.sp-odoa.value)  else 0 END AS
actualsoldprice
from sku_items_multiplemrpandsp msp
left outer join sku_items_sku sku  on sku.id = msp.sku_id
left outer join sku_items_regionarticle rart on rart.id=sku.article_id
left outer join sku_items_article art on art.id= rart.article_id
left outer join catalog_toplevel ct on ct.id = art.top_level_id
left outer join catalog_section cs on cs.id = art.section_id
left outer join catalog_category cc on cc.id = art.category_id
left outer join catalog_subcategory csb on csb.id = art.subcategory_id
left outer join catalog_brand cb on cb.id = art.brand_id
left outer join purchase_orders_company pc on pc.id= art.company_id
left outer join offers_discountofferarticle odoa on odoa.article_id = art.id
–left outer join offers_discountoffer ofdo on ofdo.id = odos.id
where CAST(msp.updated_on as DATE) between '{{daterange.start}}' and
'{{daterange.end}}'
group by 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
order by 2 desc
```

16. Reports related to multiple Grn issue:

a) Multiple Grn- Warehouse_wise
https://app.redash.io/behtar1/queries/602090

```
select
date(swic.created_on),
grn_number,
count,
grn.barcode,
swic.warehouse_id as "warehouse_id::filter",
w.name as warehouse_name,
swic.mode as _mode,
swic.from_qty,
swic.to_qty
from

(select
grn_number,
barcode,
warehouse_id,
count(grn_number) as count
from "mercuryproduction_public_GRN_goodsrecieptnote" grn1
left join "mercuryproduction_public_GRN_grnitems"  grnit on grnit.grn_id=grn1.id
group by 1,2,3 having count(grn_number)>1)grn

left join mercuryproduction_public_snapshot_warehouseinventorychanges swic
on swic.barcode = grn.barcode
left join mercuryproduction_public_warehouse_warehouse w on w.id =
grn.warehouse_id
where CAST(swic.created_on as date) between '{{daterange.start}}' and
'{{daterange.end}}' group by 1,2,3,4,5,6,7,8,9
order by 1,2
```

b) Multiple store_inventory changes
https://app.redash.io/behtar1/queries/602834

```
—some INVOICES are null for storeorderdeletion and storeauditcleanup
select
```

```
date(created_on) as _date,
invoice,
count(invoice) as count,
barcode,
ssic.store_id as "store_id::filter",
s.name as store_name,
mode,
from_qty,
to_qty

from mercuryproduction_public_snapshot_storeinventorychanges ssic
left join mercuryproduction_public_store_store  s on s.id = ssic.store_id
where CAST(created_on as date) between '{{daterange.start}}' and
'{{daterange.end}}'
group by 1,2,4,5,6,7,8,9 having count(invoice)>1
```

c) Multiple store_inventory changes — only for stock transfermode
https://app.redash.io/behtar1/queries/610062

```
–some INVOICES are null for storeorderdeletion and storeauditcleanup
select
date(created_on) as _date,
invoice,
count(invoice) as count,
barcode,
ssic.store_id,
–as "store_id::filter",
s.name as store_name,
mode,
from_qty,
to_qty

from mercuryproduction_public_snapshot_storeinventorychanges ssic
left join mercuryproduction_public_store_store  s on s.id = ssic.store_id
where mode ='StockTransfer' and
CAST(created_on as date) between '{{daterange.start}}' and '{{daterange.end}}'
group by 1,2,4,5,6,7,8,9 having count(invoice)>1

–select distinct(mode) from
mercuryproduction_public_snapshot_storeinventorychanges
```

d) Multiple store_inventory changes —–for GRN

[Multiple storeinventory changes – for GRN (redash.io)]

```
select
date(swic.created_on),
swic.invoice,
count(invoice),
swic.barcode,
swic.warehouse_id as "warehouse_id::filter",
–w.name as warehouse_name,
swic.mode as _mode,
swic.from_qty,
swic.to_qty
from mercuryproduction_public_snapshot_warehouseinventorychanges swic
where
mode != 'StockTransfer'
and mode != 'StoreToWarehouseReverse'
and mode != 'WarehouseAuditCleanUp'
and mode != 'Damage'
and mode != 'Sinkage'
and mode != 'StoreWarehouseReverseOrder'
and mode != 'RTVStoreToWarehouse'
and mode != 'Freebie by Supplier'
and mode != 'Handling Damage'
and mode != 'Internal Consumption'
and mode != 'RTVWarehouse'
and mode != 'StoreToWarehouseTransferr'
and mode != 'StTrDiscarded'
and mode != 'WhToWhTransfer'
and mode != 'WarehouseAudit'
and mode != 'Internal Audit'
and mode != 'RTV'
and mode != 'Received Damage'
and mode != 'Expiry'
and mode != 'StoreToWarehouseTransfer'
and  CAST(swic.created_on as date) between '{{daterange.start}}' and
'{{daterange.end}}'
group by 1,2,4,5,6,7,8
having count(invoice)>1
```

17. Margin Store_wise(Summary): This report is mainly done to transfer one of the report from tableau to redash:

[Margin-store_wise(summary) (redash.io)]

Objective: To find the store_wise Margin_value, Margin Percent, sales and sales share of each store in overall sales(In the requirement format of tableau margin by store report)

Approach: Written the two sub-queries using the analytics_storeorderitems and few other tables.

- ➢ 1st sub-query is for calculating the grandtotal of sales
- ➢ 2nd subquery is for calculating store_wise sales, COGS, Margin and Margin percentage.
- ➢ Then written an outer query, by calculating the salesshare of each store in overall sales along with sales, margin and margin percentage of each store.

Results: Store_wise Sales, Margin, Margin percentage and sales share of each store in overall sales (as per the tableau report)

```
select
tb2.store_name,
tb2.store_id as store_id,
SUM(tb2.totalsales) as sales,
sum(COGS) as COGS,
sum(tb2.totalsales - COGS) as Margin_Value,
sum(tb2.totalsales - COGS)/sum(tb2.totalsales) *100 as Marginpercent,
sum(tb2.totalsales)/sum(tb1.grandtotal1) *100 as salesshare
from

(select
s1.city_id,
–(sum((sales1.sp)*(sales1.quantity))/sum(sales1.quantity))*sum(sales1.quantity) as
grandtotal1
sum(sales1.sp*sales1.quantity) as  grandtotal1
from analytics_storeorderitems as sales1
left join store_store s1 on s1.id = sales1.store_id
left join sku_items_sku sku1 on sku1.id=sales1.sku_id
left join catalog_category cat1 on cat1.id = sales1.category
left join sku_items_article art1 on art1.article_code=sales1.article_code
left join sku_items_regionarticle buy1 on buy1.article_id = art1.id
left join warehouse_warehousearticlecodeinventory wi1 on wi1.article_code=
art1.article_code
where
s1.city_id= 1
and buy1.city_id=1
and sales1.quantity > 0
and s1.warehouse_id= wi1.warehouse_id
and wi1.moving_avg_price >= 0 and wi1.moving_avg_price < 20000
–and sales1.quantity<888888
AND CAST(sales1.timestamp as date) between '{{daterange.start}}' and
'{{daterange.end}}'
group by 1)tb1

left join



(select
s2.name as store_name,
sales2.store_id,
s2.city_id,
sum(sales2.sp*sales2.quantity) as totalsales,
–(sum((sales2.sp)*(sales2.quantity))/sum(sales2.quantity))*sum(sales2.quantity) as
totalsales,
sum(sales2.quantity*wi.moving_avg_price) as COGS
from analytics_storeorderitems sales2
left join store_store s2 on s2.id = sales2.store_id
left join sku_items_sku sku on sku.id=sales2.sku_id
left join catalog_category cat on cat.id = sales2.category
left join sku_items_article art on art.article_code=sales2.article_code
left join sku_items_regionarticle buy on buy.article_id = art.id
left join warehouse_warehousearticlecodeinventory wi on wi.article_code=
art.article_code
where
wi.moving_avg_price >= 0 and wi.moving_avg_price < 20000
–sales2.quantity<888888888
```

```
and buy.city_id=1
and s2.city_id = 1
and sales2.quantity > 0
and s2.warehouse_id= wi.warehouse_id
AND CAST(sales2.timestamp as date) between '{{daterange.start}}' and
'{{daterange.end}}'
group by 1,2,3)tb2   on tb1.city_id =tb2.city_id


group by 1,2
order by 5 desc nulls last
```

18. SplitBill actual Testing query:

split bill testing query (redash.io)

Objective: This report is mainly done to test the split bill part in the Behtar App.

```
select order_number, value, value_cash, value_online, value_upi,
value_cash+value_online+value_upi as actual_value
from order_storeorder
where order_number ='BEHM-477336'
```

19. SplitBill actual and total value query:

splitbill actual and total value query (redash.io)

Objective**:** To find the products where the value of the product and the actual_value(cash+upi+online) are not equal.

Approach**:** Extracted the order_number, value, cash, upi, online, sum(cash+upi+online) From orderstoreorder table and given the condition of Value!=Actualvalue

```
select
_date,
store_id,
order_number,
value,
value_upi,
value_cash,
value_online,
actual_value
from
(select
store_id,
date(created_on) as _date,
order_number, value, value_cash, value_online, value_upi,
value_cash+value_online+value_upi as actual_value
from order_storeorder
where  date(created_on) BETWEEN NOW() - INTERVAL '3 DAY' AND NOW())tb1
where value != actual_value
–and value = actual_value
–and _date BETWEEN NOW() - INTERVAL '3 DAY' AND NOW()
limit 500
```

20.Store_wise contribution in sales, margin, bill_cuts:

Store_wise contribution in sales, margin, bill cuts (redash.io)

Objective: To find the store_wise sales, Margin, no.of bill cuts and each store contribution in sales and margin

Approach: Written the two sub-queries using the analytics_storeorderitems and few other tables.

- ➢ 1st subquery is for calculating the grand total of sales and COGS
- ➢ 2nd subquery is for calculating store_wise sales, COGS, Margin and Margin percentage.
- ➢ Then written an outer query, by calculating the sales share and margin of each store in overall sales and margin along with sales, Margin and No.of billcuts of each store.

Results: Store_wise Sales, Margin, No of BillCuts, sales and Margin share of each store in overall sales.

```
select
coalesce(tb2.month2, tb3.month3) as month,
coalesce(tb2.store_name, tb3.store_name) as store_name,
coalesce(tb2.store_id, tb3.store_id) as store_id,
–L0_CATEGORY,
–L1_CATEGORY,
–L2_CATEGORY,
–L3_CATEGORY,
SUM(tb2.totalsales) as sales,
–sum(COGS2) as COGS,
sum(tb2.totalsales - COGS2) as Margin,
tb3.bill_cuts as No_of_Billcuts,
–sum(tb2.totalsales - COGS2)/sum(tb2.totalsales) *100 as Marginpercent,
sum(tb2.totalsales)/sum(tb1.grandtotalsales1) *100 as salesshare,
sum(tb2.totalsales-COGS2)/sum(tb1.grandtotalCOGS1) *100 AS marginshare
from

(select
s1.city_id,
–(sum((sales1.sp)*(sales1.quantity))/sum(sales1.quantity))*sum(sales1.quantity) as
grandtotal1
sum(sales1.sp*sales1.quantity) as  grandtotalsales1,
sum(sales1.quantity*wi1.moving_avg_price) as grandtotalCOGS1
from analytics_storeorderitems as sales1
left join store_store s1 on s1.id = sales1.store_id
left join sku_items_sku sku1 on sku1.id=sales1.sku_id
left join catalog_category cat1 on cat1.id = sales1.category
left join sku_items_article art1 on art1.article_code=sales1.article_code
left join sku_items_regionarticle buy1 on buy1.article_id = art1.id
left join warehouse_warehousearticlecodeinventory wi1 on wi1.article_code=
art1.article_code
where
s1.city_id= 1
and buy1.city_id=1
and sales1.quantity > 0
and s1.warehouse_id= wi1.warehouse_id
and wi1.moving_avg_price >= 0 and wi1.moving_avg_price < 20000
–and sales1.quantity<888888
AND CAST(sales1.timestamp as date) between '{{daterange.start}}' and
'{{daterange.end}}'
group by 1)tb1
```

```
left join



(select
extract(month from CAST(sales2.timestamp as date)) as month2,
s2.name as store_name,
sales2.store_id,
s2.city_id,
–ct.title as L0_CATEGORY,
–cs.title as L1_CATEGORY,
–cc.title as L2_CATEGORY,
–csb.title as L3_CATEGORY,
sum(sales2.sp*sales2.quantity) as totalsales,
sum(sales2.quantity*wi.moving_avg_price) as COGS2
from analytics_storeorderitems sales2
left join store_store s2 on s2.id = sales2.store_id
left join sku_items_sku sku on sku.id=sales2.sku_id
left join catalog_category cat on cat.id = sales2.category
left join sku_items_article art on art.article_code=sales2.article_code
left join sku_items_regionarticle buy on buy.article_id = art.id
left join warehouse_warehousearticlecodeinventory wi on wi.article_code=
art.article_code
left join purchase_orders_company poc on poc.id = art.company_id
left join catalog_section cs on cs.id = art.section_id
left join catalog_category cc on cc.id = art.category_id
left join catalog_subcategory csb on csb.id = art.subcategory_id
left join catalog_toplevel ct on ct.id = art.top_level_id
where
wi.moving_avg_price >= 0 and wi.moving_avg_price < 20000
–sales2.quantity<888888888
and buy.city_id=1
and s2.city_id = 1
and sales2.quantity > 0
and s2.warehouse_id= wi.warehouse_id
AND CAST(sales2.timestamp as date) between '{{daterange.start}}' and
'{{daterange.end}}'
group by 1,2,3,4)tb2   on tb1.city_id =tb2.city_id
```

```
left join

(select
extract(month from CAST(sales3.timestamp as date)) as month3,
s3.city_id,
s3.name as store_name,
sales3.store_id,
–sum(os.value) as total_sale,
count(os.value) as bill_cuts
from order_storeorder os
left join analytics_storeorderitems sales3 on sales3.order_id = os.id
left join store_store s3 on s3.id = sales3.store_id
where CAST(os.created_on as date) between '{{daterange.start}}' and '{{daterange.end}}'
and os.is_duplicate ='false'
group by 1,2,3,4) tb3 on tb2.city_id = tb3.city_id and tb2.store_id = tb3.store_id and
tb2.month2 = tb3.month3
group by 1,2,3,6

order by 1,5 desc
```

22.Splitbill actual and total value query: This report was prepared to check for any differences between actual_value and value_cash+value_online+value_upi(where the payment happened through split bill) in the split Bills.
splitbill actual and total value query (redash.io)

```
select
_date,
store_id,
order_number,
value,
value_upi,
value_cash,
value_online,
actual_value
from
(select
store_id,
date(created_on) as _date,
order_number, value, value_cash, value_online, value_upi,
value_cash+value_online+value_upi as actual_value
from order_storeorder
where  date(created_on) BETWEEN NOW() - INTERVAL '3 DAY' AND NOW())tb1
where value != actual_value
–and value = actual_value
–and _date BETWEEN NOW() - INTERVAL '3 DAY' AND NOW()
limit 500
```

23.Duplicate Generated Bills:

Duplicate Generated Bills (redash.io)

Objective: To find the duplicate generated bills having count>1

Approach: Used the order_storeorder and analytics_storeorderitems Table with is_duplicate='false' condition

```
select
tb1.order_number,
coalesce(tb1._date,tb2._date) as date,
coalesce(tb1.id, tb2.order_id) as order_id,
coalesce(tb1.mrp,tb2.mrp) as mrp,
coalesce(tb1.barcode,tb2.barcode) as barcode,
coalesce(tb1.quantity,tb2.quantity) as quantity,
value,
value_cash,
value_online,
value_upi,
count(value)
–coalesce(tb1.brand,tb2.brand)
from

(select
_date,
id,
order_number,
value,
value_cash,
value_online,
value_upi,
cast(mrp as numeric),
```

```sql
barcode,
brand,
–quantity
cast(tb3.quantity as numeric)
from

(SELECT
date(created_on) as _date,
id,
order_number,
value,
value_cash,
value_online,
value_upi,
jsonb_array_elements(items)->>'mrp' as mrp,
jsonb_array_elements(items)->>'barcode' as barcode,
jsonb_array_elements(items)->>'brand' as brand,
jsonb_array_elements(items)->>'quantity' as quantity
from order_storeorder  o

where date(o.created_on) BETWEEN NOW() - INTERVAL  '1 DAY' AND NOW()
and o.is_duplicate ='false')tb3 )tb1
left join

(select
date(timestamp) as _date,
order_id,
barcode,
brand,
cast(mrp as numeric),
cast(quantity as numeric)
from analytics_storeorderitems
where date(timestamp) BETWEEN NOW() - INTERVAL  '3 DAY' AND NOW()
)tb2 on tb1.barcode = tb2.barcode  and tb1.id =tb2.order_id  and tb1._date = tb2._date
and tb1.mrp = tb2.mrp  and tb1.quantity =tb2.quantity


group by 1,2,3,4,5,6,7,8,9,10
having count(value) >0
```

–where tb1.id= tb2.order

24. Inventory Audit Report:

a) **Warehouse_Audit_inventory Report:**

Warehouse_inventory Audit Report (redash.io)

Used Tables: Snapshot_warehouseinventorychanges

Columns: Mode, Date, Warehouse_Id, Warehouse_name, City, barcode, SKU_Title, from Qty, To_quantity, Invoice_id, User_id, user_Name

Filters: City, Mode, User_name, Warehouse_name
–Total_Inventory, Warehouse_id, city_id
—Mode, Date, StoreId, Store_name, City, barcode, SKU_Title, from Qty, To_quantity, Invoice_id, User_id, user_Name
–Filters: City, Mode, User_name, Store, Operation_type

```
select
date(swic.created_on) as created_on,
swic.mode as "mode::filter",
swic.warehouse_id,
w.name as "warehouse_name::filter",
swic.barcode,
sku.title as sku_name,
swic.from_qty,
swic.to_qty,
swic.invoice,
w.city_id as "city_id::filter",
swic.User as user_id,
–ubu.id,
ubu.email as "user_name::filter"

from snapshot_warehouseinventorychanges swic
left join warehouse_warehouse w on w.id = swic.warehouse_id
left join sku_items_sku sku on sku.barcode = swic.barcode
left join users_behtaruser ubu on ubu.id = swic.user
where
swic.from_qty != swic.to_qty
and CAST(swic.created_on as date) between '{{daterange.start}}' and '{{daterange.end}}'
```

```
group by 1, 2,3,4,5,6,7,8,9,10,11,12
```

b) Store_Audit_Inventory Report:

Store Inventory Audit Report (redash.io)

Used Tables: Snapshot_warehouseinventorychanges

Columns: Mode, Date, Store_Id, Store_name, City, barcode, SKU_Title, from Qty, To_quantity, Invoice_id, User_id, user_Name

Filters: City, Mode, User_name, Store_name

```
—Mode, Date, StoreId, Store_name, City, barcode, SKU_Title, from Qty, To_quantity,
Invoice_id, User_id, user_Name
–Filters: City, Mode, User_name, Store, Operation_type
select
date(ssic.created_on) as created_on,
ssic.store_id,
ssic.mode as "mode::filter",
s.name as "Store_Name::filter",
ssic.barcode,
ssic.from_qty,
ssic.to_qty,
ssic.invoice,
s.city_id as "city_id::filter",
sku.title as sku_name,
ssic.User as user_id,
–ubu.id,
ubu.email as "user_name::filter"
from snapshot_storeinventorychanges ssic
left join store_store s on s.id = ssic.store_id
left join sku_items_sku sku on sku.barcode = ssic.barcode and s.city_id = sku.city_id
left join users_behtaruser ubu on ubu.id = ssic.user
where
ssic.from_qty != ssic.to_qty
and s.city_id =1
and s.status='A'
and sku.city_id=1
and CAST(ssic.created_on as date) between '{{daterange.start}}' and '{{daterange.end}}'
```

```
group by 1, 2,3,4,5,6,7,8,9,10,11,12
```

25) Stock Transfer Return: Returns of the stock transfer filtered by Warehouse_nameand invoice_number

Objective: to find the items which are returned from store to warehouse

Approach: used the storewarehousereverseorder and order_storewarehouseorder tables to find the items which are returned from store with barcodes, invoice_numbers and stock_transfer_ids

a)Stock transfer_return with warehouse_name filter: Applied Warehouse_name filter

Columns: Date, Invoice_number, Stock_transfer_id, total_value, sku_name, barcode, article_code, warehouse_name, warehouse_id, store_name, store_id, city_id

Stock Transfer Return–filtered by Warehouse (redash.io)

```
select
date(oswr.created_on),
oswr.invoice_number,
–as "invoice_number::filter",
oswr.stock_transfer_id,
oswr.total_value,
–oswri.quantity,
sku.title as sku_name,
sku.barcode,
art.article_code,
w.name as "warehouse_name::filter",
w.id as warehouse_id,
s.id as store_id,
s.name as store_name,
sku.city_id
–as "city_id::filter"
```

```
from order_storewarehousereverseorder oswr
left join order_storewarehousereverseorderitem oswri on oswr.id =
oswri.store_warehouse_reverse_order_id
left join sku_items_sku sku on sku.id =oswri.sku_id
left join sku_items_regionarticle rart on rart.id=sku.article_id
left join sku_items_article art on art.id = rart.article_id
left join order_storewarehouseorder oswo on oswr.stock_transfer_id  = oswo.id
left join warehouse_warehouse w on w.id = oswo.warehouse_id
left join store_store s on s.id = oswo.store_id
where CAST(oswr.created_on as date) between '{{daterange.start}}' and
'{{daterange.end}}'
```

b)Stock transfer_return with invoice_number filter: Applied Invoice_number filter

Columns: Date, Invoice_number, total_value, quantity, sku_id, sku_name, barcode, article_code, city_id

StockTransferReturn–filterby Invoice_number (redash.io)

```
select
date(oswr.created_on),
oswr.invoice_number as "invoice_number::filter",
oswr.total_value,
oswri.quantity,
oswri.sku_id,
sku.title,
sku.barcode,
art.article_code,
sku.city_id
—as "city_id::filter"

from order_storewarehousereverseorder oswr
left join order_storewarehousereverseorderitem oswri on oswr.id =
store_warehouse_reverse_order_id
left join sku_items_sku sku on sku.id =oswri.sku_id
left join sku_items_regionarticle rart on rart.id=sku.article_id
left join sku_items_article art on art.id = rart.article_id
```

```
where CAST(oswr.created_on as date) between '{{daterange.start}}' and
'{{daterange.end}}'
```

26)Fraud detection Reports: These are mainly done to find the frauds happening at the organization in different situations.

a) **Multiple Times of Returning:** Returning the same item Multiple times
multiple times of returning (redash.io)

Approach: Using Order_returnorder and order_storeorder tables and applied the condition of count(ORDERBARCODE)>1 so that only multiple returned items can be extracted.

Columns: Returned_on, order_id, total_returned_items, order_number, Order_barcode, count(orderbarcode), returned_barcode, returned_value, ordered_value, store_name, cashiers(mobile_numbers).
– returned same items multiple times

```
select
returned_on,
order_id,
os.order_number,
total_returned_items,
ORDERBARCODE,
count(ORDERBARCODE),
os.value as "ordered_value",
returned_value,
ss.name as store_name,
ss.cashiers
–coupon_applied
from

(select
date(orr.created_on) as returned_on,
orr.value as returned_value,
orr.order_id,
json_array_length(items::json) as total_returned_items,
jsonb_array_elements(items)->>'mrp' as MRP_ITEM ,
jsonb_array_elements(items)->>'barcode' as ORDERBARCODE
from order_orderreturn orr
) tb1
left outer join order_storeorder os on os.id=tb1.order_id
left outer join store_store ss on os.store_id=ss.id
where CAST(returned_on as date) between '{{DateRange.start}}' and
'{{DateRange.end}}'
group by 1,2,3,4,5,7,8,9,10
having count(ORDERBARCODE)>1
order by 8 desc
```

b) **Returning item which are not purchased:** Returning the items which are not purchased by the customer
Returning item which are not purchased (redash.io)

Approach: Using Order_returnorder and order_storeorder tables and applied the condition of Returnedbarcode != ORDERBARCODE, so that only returned items which are not ordered or purchased
Columns: Returned_on, order_id, total_returned_items, order_number, Order_barcode, returned_barcode, returned_value, ordered_value, store_name, cashiers(mobile_numbers).
—returned the items that was not purchased by the customer

```
select
returned_on,
order_id,
id,
total_returned_items,
order_number as "order_number::filter",
returned_value,
ordered_value,
ORDERBARCODE,
returnedbarcode,
store_name,
cashiers
from

(select
returned_on,
returned_value,
order_id,
os.id,
total_returned_items,
returnedbarcode,
jsonb_array_elements(os.items)->>'barcode' as ORDERBARCODE,
os.value as ordered_value,
os.order_number,
ss.name as store_name,
ss.cashiers
from

(select
date(orr.created_on) as returned_on,
orr.value as returned_value,
orr.order_id,
json_array_length(items::json) as total_returned_items,
jsonb_array_elements(items)->>'barcode' as returnedbarcode
from order_orderreturn orr
) tb1
left join order_storeorder os on os.id=tb1.order_id
left outer join store_store ss on os.store_id=ss.id)tb2

where CAST(returned_on as date) between '{{DateRange.start}}' and
'{{DateRange.end}}'
and returnedbarcode != ORDERBARCODE
and id = order_id
group by 1,2,3,4,5,6,7,8,9,10,11
order by 1, 6 desc
```

c) **Returned_value > ordered_value:** Returned items where the returned_value more than the ordered_value
Returned_value> ordered_value (redash.io)
Approach: Using Order_returnorder and order_storeorder tables
Columns: Returned_on, order_id, total_returned_items, Order_barcode, returned_value, ordered_value, extrareturned_amount, order_number, store_name, cashiers(mobile_numbers).
–returned the items with more than the ordered_value

```
select
returned_on,
order_id,
total_returned_items,
ORDERBARCODE,
returned_value,
os.value as "ordered_value",
(returned_value -os.value) as "extrareturnedamount",
os.order_number,
ss.name as store_name,
ss.cashiers

from

(select
date(orr.created_on) as returned_on,
orr.value as returned_value,
orr.order_id,
json_array_length(items::json) as total_returned_items,
jsonb_array_elements(items)->>'mrp' as MRP_ITEM ,
jsonb_array_elements(items)->>'barcode' as ORDERBARCODE
from order_orderreturn  orr
) tb1
left outer join order_storeorder  os on os.id=tb1.order_id
left outer join store_store  ss on os.store_id=ss.id
where CAST(returned_on as date) between '{{DateRange.start}}' and
'{{DateRange.end}}'
and returned_value > os.value
group by 1,2,3,4,5,6,7,8,9,10
order by 7 desc
```

27) Warehouse_inventory_opening& closing:

Inventory-warehouse-opening&Closing (redash.io)

Objective: To find the opening & closing Inventory value of each warehouse on each day.

<u>Approach</u>: Used the warehouse_warehousesnapshotarticle for opening and closing quantity and sku for selling price(SP).

```
select tb1._date,
warehouse_id,
name as "warehouse_name::filter",
sum(tb1.opening*tb2.sp) as openingvalue,
sum(tb1.closing*tb2.sp) as closingvalue
from

(select CAST(wsa.date as date) as _date,
warehouse_id,
w.name,
wsa.article_code,
sum(wsa.start_quantity) as opening,
sum(wsa.end_quantity) as closing
from warehouse_warehousesnapshotarticle wsa
left join warehouse_warehouse w on w.id=wsa.warehouse_id
group by 1,2,3,4)tb1
left join

(select
art.article_code,
art.title as article_name,
–ct.title as top_level,
cs.title as section,
cc.title as category,
–csb.title as sub_category,
avg(sku.mrp) as mrp,
avg(sku.sp) as sp
from sku_items_sku sku
left join sku_items_regionarticle rart on rart.id = sku.article_id
left join sku_items_article art on art.id = rart.article_id
left join catalog_section cs on cs.id = art.section_id
left join catalog_category cc on cc.id = art.category_id
–left join catalog_subcategory csb on csb.id = art.subcategory_id
–left join catalog_toplevel ct on ct.id = cs.top_level
group by 1,2,3,4)tb2 on tb2.article_code=tb1.article_code
group by 1,2,3
```

```
order by 1 desc
```

28) Behtar Inventory_amount report: To find the overall Inventory value present at warehouse and store level of each article

a) Inventory_amount_Warehouse_wise:

Inventory_amount- warehouse_wise (redash.io)

Used_tables: Warehouse_warehouseinventory and sku by warehouse_name filter

Columns: created_on, city_id, arcticle_name, L1, L2, Warehouse_id, Warehouse_name,value, wh_qty.

```
–Total_Inventory, Warehouse_id, city_id
select
date(wwi.created_on) as created_on,
w.city_id,
art.title as article_name,
cs.title as L1_section,
cc.title as L2_category,
wwi.warehouse_id,
w.name as "wh_name::filter",
sum(wwi.quantity*sku.sp) as value,
sum(wwi.quantity) as wh_qty

from warehouse_warehouseinventory wwi
left join sku_items_sku sku on wwi.sku_id = sku.id
left join warehouse_warehouse w on w.id = wwi.warehouse_id
left join sku_items_regionarticle rart on rart.id=sku.article_id
left join sku_items_article art on art.id = rart.article_id
left join catalog_section cs on cs.id = art.section_id
left join catalog_category cc on cc.id = art.category_id
left join catalog_subcategory csb on csb.id = art.subcategory_id
left join catalog_toplevel ct on ct.id = art.top_level_id

where CAST(wwi.created_on as date) between '{{DateRange.start}}' and
'{{DateRange.end}}'
group by 1,2,3,4,5,6,7
```

b) Inventory_amount_store_wise:

Inventory_amount- Store_wise (redash.io)
Used_tables: store_storeinventory and sku by store_name filter

Columns: date, city_id, arctile_name, L1, L2, store_id, store_name,value, store_qty.
–Total_Inventory, Warehouse_id, city_id

```
select
date(ssi.last_received_date),
s.city_id,
art.title,
cs.title as L1_section,
cc.title as L2_category,
ssi.store_id,
s.name as "store_name::filter",
sum(ssi.quantity*sku.sp) as value,
sum(ssi.quantity) as store_qty

from store_storeinventory ssi
left join sku_items_sku sku on ssi.sku_id = sku.id
left join store_store s on s.id = ssi.store_id
left join sku_items_regionarticle rart on rart.id=sku.article_id
left join sku_items_article art on art.id = rart.article_id
left join catalog_section cs on cs.id = art.section_id
left join catalog_category cc on cc.id = art.category_id
left join catalog_subcategory csb on csb.id = art.subcategory_id
left join catalog_toplevel ct on ct.id = art.top_level_id
where CAST(ssi.last_received_date as date) between '{{DateRange.start}}' and
'{{DateRange.end}}'
group by 1,2,3,4,5,6,7
```

29)Landing_Price& Tax: To find the clear difference between basic_price, tax_value, price_with_gst.
LP & Tax (redash.io)

Used_tables: Warehouse_warehouse, grn tables.

Columns: sku_name, article_name, lo_top_level, l1_section, l2_category, l3_subcategory, mrp, sp, basic_price, tax_value, quantity, basic_total, tax_total, price_with_gst, avg_landing_price, brand, company_name.

```
--	SKU Name	Article Name Quantity	MRP	Current SP	Moving Average LP
	Tax Rate
--L0 Category L1 Category	L2 Category	L3 Category	Brand Company
--(latest MRP)		(lastest sp)	(store-wise)
select
grn.grn_number,
grnit.article_code,
grnit.barcode,
sku.title as sku_name,
art.title as art_name,
ct.title as L0_top_level,
cs.title as L1_section,
cc.title as L2_category,
csb.title as L3_sub_category,
sku.mrp,
sku.sp,
grnit.basic_price,
grnit.tax_value,
grnit.quantity,
grnit.basic_total,
grnit.tax_total,
grnit.landing_price as price_with_gst,
rart.avg_landing_price,
brand.title as brand,
poc.name as company_name

from "GRN_grnitems" grnit
left join "GRN_goodsrecieptnote" grn on grn.id=grnit.grn_id
left join sku_items_sku sku on grnit.sku_id=sku.id
left join warehouse_warehouse w on w.id = grn.warehouse_id
left join sku_items_regionarticle rart on rart.id=sku.article_id
left join sku_items_article art on art.id = rart.article_id
left join purchase_orders_company poc on poc.id = art.company_id
left join catalog_section cs on cs.id = art.section_id
left join catalog_category cc on cc.id = art.category_id
left join catalog_subcategory csb on csb.id = art.subcategory_id
left join catalog_toplevel ct on ct.id = art.top_level_id
left join catalog_brand brand on brand.id=art.brand_id
```

```
where CAST(grn.counted_on as date) between '{{DateRange.start}}' and
'{{DateRange.end}}'
–group by 1,2,3,4,5,6,7,8,9,10,11,12,13,14
```

30) Shrinkage Report: This report is mainly done to find the shrinkage_value (Like damage) at store and warehouse_level.

a)Store_wise: To find the items which are shrinkage wasting and expiring at store_level.
Shrinkage Report– Store_wise (redash.io)

Approach: Using snapshot_storeinventorychangesand sku_items_sku tables and applied the condition of mode ='Expiry and Damage’.

Columns: date, store_id, store, shrinkage_quantity, sp, total_value, mode, section, category, sku_name

```
–store month category       section         nonrecoverable        recoverable   sales

select
tb1._date,
–tb1._date,
tb1.store_id,
tb1.store,
–tb1.barcode,
tb1.section,
tb1.category,
tb1.shrink_qty,
sp,
tb1.shrink_qty*tb1.sp as shrink_value,
tb1.mode
–tb1.sku_name,
–top_level,
from

(select
–extract(month from
CAST(sic.created_on as date) as _date,
–CAST(sic.created_on as date) as _date,
sic.store_id,
s.name as store,
sic.mode,
–ct.title as top_level,
cs.title as section,
cc.title as category,
–sic.barcode,
CASE WHEN sku.grammage is not null THEN (sku.title || ' ' || sku.grammage || ' ' ||
sku.unit) ELSE sku.title END as sku_name,
sum(coalesce((sic.to_qty - sic.from_qty),0.00)) as shrink_qty,
avg(sku.sp) as sp

from snapshot_storeinventorychanges sic
left join store_store s on s.id = sic.store_id
left join sku_items_sku sku on sku.barcode = sic.barcode
left join sku_items_regionarticle rart on rart.id=sku.article_id
left join sku_items_article art on art.id = rart.article_id
left join catalog_section cs on cs.id = art.section_id
left join catalog_category cc on cc.id = art.category_id
–left join catalog_subcategory csb on csb.id = art.subcategory_id
left join catalog_toplevel ct on ct.id = art.top_level_id
where
s.city_id =1
and s.status='A'
and sku.city_id=1
and s.id!='84'
and s.id!='204'
and s.id!='206'
–and (coalesce((sic.to_qty - sic.from_qty),0.00)) < 10000
–and (coalesce((sic.to_qty - sic.from_qty),0.00)) > -10000
and CAST(sic.created_on as date) between '{{Date Range.start}}' and '{{Date
Range.end}}'
and (sic.mode = 'Damage' or sic.mode = 'Expiry')
group by 1,2,3,4,5,6,7)tb1

order by 1 desc, 2 asc, 4 asc nulls last
```

b)Warehouse_wise:

Objective: To find the items which are (shrinkage) wasting and expiring at store_level.

shrinkage-warehouse_wise (redash.io)

Approach: Using warehouse_warehouseskucorrectionitems and sku_items_sku tables and applied the condition of option ='Sinkage'

Columns: date, item_total, quantity, option, section, category, sku_name

```
–store month category       section         nonrecoverable       recoverable   sales
select
date(wsi.created_on),
wsi.item_total,
wsi.quantity,
wsi.option,
cs.title as section,
cc.title as category,
sku.title as sku_name
from warehouse_warehouseskucorrectionitems wsi
left join sku_items_sku sku on sku.id = wsi.sku_id
left join sku_items_regionarticle rart on rart.id=sku.article_id
left join sku_items_article art on art.id = rart.article_id
left join catalog_section cs on cs.id = art.section_id
left join catalog_category cc on cc.id = art.category_id
–left join catalog_subcategory csb on csb.id = art.subcategory_id
left join catalog_toplevel ct on ct.id = art.top_level_id
where CAST(wsi.created_on as date) between '{{Date Range.start}}' and '{{Date Range.end}}'
and option ='Sinkage'
```

31)Warehouseinventoryamount_quater_wise: To find the total_value and total_quantity of each warehouse for a quarter.

warehouseinventoryamount_quater_wise (redash.io)

Approach: Using warehouse_warehouseinventory and sku_items_sku tables

Columns: warehouse_id, warehouse_name, value, warehouse_quantity of each warehouse for a quarter.

```
select
warehouse_id,
warehouse as "warehouse_name",
value,
wh_qty
from

(select
wwi.warehouse_id,
w.name as "warehouse",
sum(wwi.quantity*sku.sp) as value,
sum(wwi.quantity) as wh_qty
from warehouse_warehouseinventory wwi
left join sku_items_sku sku on wwi.sku_id = sku.id
left join warehouse_warehouse w on w.id = wwi.warehouse_id
where CAST(wwi.created_on as date) between '{{DateRange.start}}' and
'{{DateRange.end}}'
group by 1,2
)tb1

group by 1,2,3,4
```

32). StockTransfer with seller_id and supplier_id: This is mainly prepared to know the stock transfer with supplier_id

Stock Transfer with seller_id and supplier_id (redash.io)

Approach: From purchase_orders_purchaseorderitems table taken the supplier_id column and left joined with stock transfer query.

Benefits: In this report we can see the stock transfer from warehouse to store along with the supplier_id and seller_id.

Columns: Invoice_number, article_code, seller_id, supplier_id, supplier_name, sku_id, sent_qty, sent_value, warehouse_name, store_name.

```
–transfered goods from warehouse to store
—seller_id     sku_id category_id    section_id      st_value        st_qty

select
city_id as"city_id::filter",
CAST(tb1.created_on as date) as created_on,
CAST(Store_Received_Date as date) as store_received,
invoice_number,
tb1.article_code,
seller_id,
tb2.supplier_id,
tb2.name as supplier_name,
tb1.sku_id,
category_id,
section_id,
sent_quantity,
st_value,
warehouse_name,
tb1.warehouse_id,
store_name,
store_id,
status


from
(select
s.city_id,
art.article_code,
date(o.created_on) as created_on,
o.store_receive_timestamp as Store_Received_Date,
o.invoice_number,
s.seller_id,
sku.id as sku_id,
cc.id as category_id,
cs.id as section_id,
oi.sent_qty as sent_quantity,
oi.item_total as st_value,
s.name as store_name,
s.id as store_id,
w.name as warehouse_name,
w.id as warehouse_id,
o.status

from order_storewarehouseorder o
left join order_storewarehouseorderitem oi on o.id=oi.store_warehouse_order_id
left join sku_items_sku sku on sku.id=oi.sku_id
left join store_store s on s.id = o.store_id
left join warehouse_warehouse w on w.id = o.warehouse_id
left join sku_items_regionarticle rart on rart.id=oi.article_id
left join sku_items_article art on rart.article_id=art.id
left join catalog_section cs on cs.id=sku.section_id
left join catalog_category cc on cc.id = art.category_id
left join catalog_subcategory csb on csb.id = art.subcategory_id
left join catalog_toplevel ct on ct.id = art.top_level_id
left join purchase_orders_company poc on art.company_id=poc.id
where s.status = 'A'
and sent_qty > 0
group by 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16)tb1
–and w.id=9

left join

(select
pi.article_code,
pi.sku_id,
p.supplier_id,
ps.name
from "purchase_orders_purchaseorderitems" pi
left join "purchase_orders_purchaseorders" p on pi.purchase_orders_id= p.number
left join "purchase_orders_supplier" ps on p.supplier_id= ps.id
left join "sku_items_sku" sku on sku.id=pi.sku_id

group by 1,2,3,4
)tb2 on  tb1.article_code = tb2.article_code  and tb1.sku_id=tb2.sku_id
where CAST(tb1.created_on as date) between '{{daterange.start}}' and
'{{daterange.end}}'

order by 1 asc nulls last
```

33). Sku Correction: This report is prepared to find the new skus which are added newly after march 31st 2021.

a) **Sku_correction with barcode, store_id, cashier_login** sku_correction with store_id, barcode, cashierlogin (redash.io)

Approach: Used analytics_analyticsevents from productionsnapshot table, excluded the dummy store(store_id =5) and considered the new skus only after march 31st 2021, by giving the condition of event_name ='add_sku'

Columns: created_on, event_name, store_id, barcode, cashier_login.

```
—not included dummy stores(store_id =5)
select
created_on,
event_name,
storeid,
Barcode,
cashier_login
from

(select
date(created_on) as created_on,
event_name,
event_details ->'storeId' as storeid,
event_details -> 'barcode' as Barcode,
event_details ->'cashier_login' as cashier_login
from analytics_analyticsevents aa
where event_name ='add_sku')tb1

where (storeid !='5') –and storeid !='766')
and date(created_on) >= '2021-03-31'
group by 1,2,3,4,5
order by 1 desc
```

b) Sku_correction with sp, mrp , unit

sku_correction with sp, mrp, unit (redash.io)

Approach: Used analytics_analyticsevents from productionsnapshot table, excluded the dummy store(store_id =5) and considered the new skus only after march 31st 2021, by giving the condition of event_name ='add_sku'

Columns: created_on, barcode, count(barcode), sp, mrp, unit, event_name, store_id, sku_title, weightage, total_payable_value and cashier_login.

```
—not included dummy stores(store_id =5) and mrp, sp details exists only from 31 march
2021
select
created_on,
Barcode,
count(Barcode),
sp,
mrp,
unit as "unit::filter",
event_name,
event_type,
storeid,
sku_title,
weightage,
totalPayable,
cashier_login
from

(select
date(created_on) as created_on,
event_name,
event_type,
event_details ->'storeId' as storeid,
event_details -> 'barcode' as Barcode,
event_details ->'sp' as sp,
event_details -> 'mrp' as mrp,
event_details ->'unit' as unit,
event_details ->'sku_title' as sku_title,
event_details ->'weightage' as weightage,
event_details -> 'totalPayable' as totalPayable,
event_details ->'cashier_login' as cashier_login
from analytics_analyticsevents aa
where event_name ='add_sku')tb1

where (storeid !='5') –and storeid !='766')
and date(created_on) >= '2021-03-31'
group by 1,2,4,5,6,7,8,9,10,11,12,13
```

order by count(Barcode) desc

34). Top 50 SP with barcodes : This report is prepared to find the top high SP with the barcodes.
Top 50 SP with barcodes (redash.io)
.
Approach: Used sku_items_sku table and extracted the values where sp is not null.

Columns: city_id, sp, mrp, barcode.

```
— need some barcodes both in delhi and bangalore
select
city_id,
sp,
mrp,
barcode

from sku_items_sku
where sp is not null
group by 1,2,3,4
order by sp desc limit 50
```

35) Sales demand prediction model using python: Build this model, mainly to predict the next 3 month sales

salesprediction - Colaboratory (google.com)

Approach:

a)Data: The data was taken from feb 1st to 19th april which contains `(1695988 rows and 9 columns)` by writing a query in redash.
sales data for demand production model 2 (redash.io)
Columns: store_id, store_name, date, day of week, day_name, barcode, quantity, sp

b)processing: some findings from the data

- ➢ Sales trend over days - on which day sales occur more ?

- ➢ Which store_id has more sales ?
- ➢ Which barcode has more sales ?

c) Pre-Processing of the data: Removed the not required variables of store_name, day_name, date, day of week, sp

- ➢ To build the Multiple linear regression model, all the variables should be in float, so converted the barcode to float
- ➢ Removed the null variables
- ➢ After removing the null variables the data contains `(1515457 rows, 4 columns)`

d) correlation between dependent variable(sales) and independent variable

```
sales      1.000000
quantity   0.151901
barcode   -0.005886
store_id  -0.007971
```

e) Model Building:

- ➢ Splitted the dataset in 80:20 ratio
- ➢ Build the multiple linear regression model

f) Results: R2 is `0.03829,`

Model got very less R2 and model is not able to predict the matching or exact sales.
A low R-squared value indicates that our independent variable is not explaining much in the variation of our dependent.

36) Margin by article: This report is done to find the margin by article wise

margin by article (redash.io)

Columns: article_code, article_name, Brand, total_quantity_sold, weighted_avg_sp, total_sales_value, m_avg_lp, cogs, margin, md_margin

```
—article_code        article_name Brand  total_quantity_sold   weighted_avg_sp
        total_sales_value     m_avg_lp     cogs    margin          md_margin
select
```

```
article_code,
article_name,
brand_name,
Total_Quantity_Sold,
Weighted_Avg_SP,
Total_Sales_Value,
m_avg_lp,
COGS,
(Total_Sales_Value - COGS) as Margin,
(Total_Sales_Value - COGS)/Total_Sales_Value *100 as MD_Margin

from
(select
art.article_code as article_code,
art.title as article_name,
cb.title as brand_name,
wi.moving_avg_price as m_avg_lp,
sum(sales.quantity) as Total_Quantity_Sold,
sum((sales.sp)*(sales.quantity))/sum(sales.quantity) as Weighted_Avg_SP,
(sum((sales.sp)*(sales.quantity))/sum(sales.quantity))*sum(sales.quantity) as
Total_Sales_Value,
wi.moving_avg_price*sum(sales.quantity) as COGS,
(sum((sales.sp)*(sales.quantity))/sum(sales.quantity) - wi.moving_avg_price) as Margin


from analytics_storeorderitems sales
left join store_store s on s.id = sales.store_id
left join sku_items_sku as sku on sku.id=sales.sku_id
left join sku_items_regionarticle as reg on reg.id=sku.article_id
left join sku_items_article art on  art.id = reg.article_id
left join purchase_orders_company poc on poc.id = art.company_id
left join catalog_section cs on cs.id = art.section_id
left join catalog_category cc on cc.id = art.category_id
left join catalog_subcategory csb on csb.id = art.subcategory_id
left join catalog_toplevel ct on ct.id = art.top_level_id
left join catalog_brand cb on cb.id = art.brand_id
left join warehouse_warehousearticlecodeinventory wi on wi.article_code=
art.article_code
where
CAST(sales.timestamp as date) between '{{daterange.start}}' and '{{daterange.end}}'
and reg.city_id=1
and s.city_id = 1
and sales.quantity > 0
and s.warehouse_id= wi.warehouse_id
group by 1,2,3,4
)tb1
where  Total_Sales_Value >  0
order by 4 desc
```

37) Articles not being sold at the store: This report is prepared to find the articles which are not being sold from some days at the store

Columns: store_name, store_id, article_code, barcode, quantity, sp, overvalue, article_name, sku_name, last_sold, not_sold_for,

Approach: Taken the store_storeinventory table, with the help of last_sold column found from how many days the items are not selling from the store.
Not_sold_for has null values and not null values, so splitted this into two reports.

a) **where not_sold for != null:** To find the articles where not _sold_for column is not null by giving not_sold_for != null condition
   [articles not being sold at the store where not_soldfor != null (redash.io)]

```
— articles not being sold at the store
select
Store_Name as "store_name::filter",
store_id,
article_code,
barcode,
quantity,
sp,
over_value,
article_name,
sku_name,
city_id,
last_sold,
Not_Sold_For
from

(select
s.name as Store_Name,
s.id as store_id,
art.article_code as article_code,
sku.barcode as barcode,
s_inv.quantity as quantity,
sku.sp as sp,
s_inv.quantity*sku.sp as over_value,
art.title as article_name,
CASE WHEN sku.grammage is not null THEN (sku.title || ' ' || sku.grammage ||' ' ||
sku.unit) ELSE sku.title END as sku_name,

s.city_id as city_id,
CAST(last_sold as date) as last_sold,
DATEDIFF(day, CAST(last_sold as date), current_date)+1 AS Not_Sold_For

from mercuryproduction_public_store_storeinventory s_inv
left join mercuryproduction_public_store_store s on s.id=s_inv.store_id
left join mercuryproduction_public_sku_items_sku sku on sku.id = s_inv.sku_id
left join mercuryproduction_public_sku_items_regionarticle rart on
rart.id=sku.article_id
left join mercuryproduction_public_sku_items_article art on art.id = rart.article_id

where s_inv.quantity > 0
and s.id != 5
and s.city_id ='2')tb1
where last_sold is not null
order by last_sold desc
```

b) **where not_sold for = null:** To find the articles where not _sold_for column is null by giving not_sold_for= null condition

articles not being sold at the store – where not_soldfor = null (redash.io)

```
— articles not being sold at the store
select
Store_Name as "Store_Name::filter",
store_id,
article_code,
barcode,
quantity,
sp,
over_value,
article_name,
sku_name,
city_id,
last_sold,
Not_Sold_For
from

(select
s.name as Store_Name,
s.id as store_id,
art.article_code as article_code,
sku.barcode as barcode,
s_inv.quantity as quantity,
sku.sp as sp,
s_inv.quantity*sku.sp as over_value,
art.title as article_name,
CASE WHEN sku.grammage is not null THEN (sku.title || ' ' || sku.grammage ||' ' ||
sku.unit) ELSE sku.title END as sku_name,

s.city_id as city_id,
CAST(last_sold as date) as last_sold,
DATEDIFF(day, CAST(last_sold as date), current_date)+1 AS Not_Sold_For

from mercuryproduction_public_store_storeinventory s_inv
left join mercuryproduction_public_store_store s on s.id=s_inv.store_id
left join mercuryproduction_public_sku_items_sku sku on sku.id = s_inv.sku_id
left join mercuryproduction_public_sku_items_regionarticle rart on rart.id=sku.article_id
left join mercuryproduction_public_sku_items_article art on art.id = rart.article_id

where s_inv.quantity > 0
and s.id != 5
and s.city_id ='2')tb1
where last_sold  isnull

order by 12 desc
```