# SQL Advanced – Finance Analytics

Projects / AtliQ Data Analytics / AFA-1 / AFA-5

## Croma India product wise sales report for fiscal year 2021

Attach | Add a child issue | Link issue

Description

Normal text

**As a** product owner, **I want to** generate a report of individual product sales (aggregated on a monthly basis at the product level) for Amazon India customers for FY=2021 **so that** I can track individual product sales and run further product analytics on it in excel.

The report should have the following fields,

1. Month
2. Product Name & Variant
3. Sold Quantity
4. Gross Price Per Item
5. Gross Price Total
6. Variants

SQL QUERY

```sql
SELECT
    s.date,p.product,p.variant,
    s.sold_quantity,g.gross_price,
    ROUND(s.sold_quantity*g.gross_price,2) as gross_price_total
FROM fact_sales_monthly s
    JOIN
        dim_product p ON s.product_code=p.product_code
    JOIN
        fact_gross_price g
    ON
        p.product_code=g.product_code AND
        get_fiscal_year(s.date)=g.fiscal_year
WHERE
    customer_code='90002002' AND
    get_fiscal_year(s.date)=2021;
```

| date | product | variant | sold_quantity | gross_price | gross_price_total |
|---|---|---|---|---|---|
| 2020-09-01 | AQ Dracula HD... | Standard | 202 | 19.0573 | 3849.57 |
| 2020-10-01 | AQ Dracula HD... | Standard | 95 | 19.0573 | 1810.44 |
| 2020-12-01 | AQ Dracula HD... | Standard | 113 | 19.0573 | 2153.47 |
| 2021-01-01 | AQ Dracula HD... | Standard | 182 | 19.0573 | 3468.43 |
| 2021-02-01 | AQ Dracula HD... | Standard | 208 | 19.0573 | 3963.92 |
| 2021-04-01 | AQ Dracula HD... | Standard | 199 | 19.0573 | 3792.40 |
| 2021-05-01 | AQ Dracula HD... | Standard | 58 | 19.0573 | 1105.32 |
| 2021-06-01 | AQ Dracula HD... | Standard | 205 | 19.0573 | 3906.75 |
| 2021-08-01 | AQ Dracula HD... | Standard | 88 | 19.0573 | 1677.04 |
| 2020-09-01 | AQ Dracula HD... | Plus | 162 | 21.4565 | 3475.95 |

# Gross monthly total sales report for Croma

Attach | Add a child issue | Link issue | •••

Description

**As a** product owner, **I need** an aggregate monthly gross sales report for Croma India customer **so that** I can track how much sales this particular customer is generating for AtliQ and manage our relationships accordingly.

The report should have the following fields,

1. Month
2. Total gross sales amount to Croma India in this month

```sql
SELECT
    s.date,
    ROUND(SUM(g.gross_price*s.sold_quantity),1) AS Gross_price_total
FROM fact_sales_monthly s
JOIN fact_gross_price g
ON
    s.product_code = g.product_code AND
    g.fiscal_year=get_fiscal_year(s.date)
WHERE customer_code=90002002
GROUP By s.date
ORDER BY date ASC;
```

| date | Gross_price_total |
|---|---|
| 2017-09-01 | 122407.6 |
| 2017-10-01 | 162687.6 |
| 2017-12-01 | 245673.8 |
| 2018-01-01 | 127574.7 |
| 2018-02-01 | 144799.5 |
| 2018-04-01 | 130643.9 |
| 2018-05-01 | 139165.1 |
| 2018-06-01 | 125735.4 |
| 2018-08-01 | 125409.9 |
| 2018-09-01 | 343337.2 |
| 2018-10-01 | 440562.1 |
| 2018-12-01 | 653944.7 |

# Generate a yearly report for Croma India where there are two columns

1. Fiscal Year
2. Total Gross Sales amount In that year from Croma

```sql
SELECT
    get_fiscal_year(date) as fiscal_year,
    SUM(ROUND(g.gross_price*s.sold_quantity,1)) as yearly_sales
FROM fact_sales_monthly s
JOIN fact_gross_price g
ON
    s.product_code=g.product_code AND
    g.fiscal_year=get_fiscal_year(s.date)
WHERE
    customer_code=90002002
GROUP By get_fiscal_year(s.date)
ORDER BY yearly_sales DESC;
```

| fiscal_year | yearly_sales |
|---|---|
| 2022 | 44638198.5 |
| 2021 | 23216512.2 |
| 2020 | 6502184.0 |
| 2019 | 3555081.7 |
| 2018 | 1324098.1 |

Add epic / AFA-17

## Created stored proc for monthly gross sales report

Attach | Add a child issue | Link issue

Description

Normal text

As a data analyst, I want to create a stored proc for monthly gross sales report so that I don't have to manually modify the query every time. Stored proc can be run by other users to (who have limited access to database) and they can generate this report without having to involve data analytics team.

The report should have the following columns,

1. Month
2. Total gross sales in that month from a given customer

# STORED PROCEDURE

Stored procedure is a way to automate repeated tasks such as creating the same report for different customers.

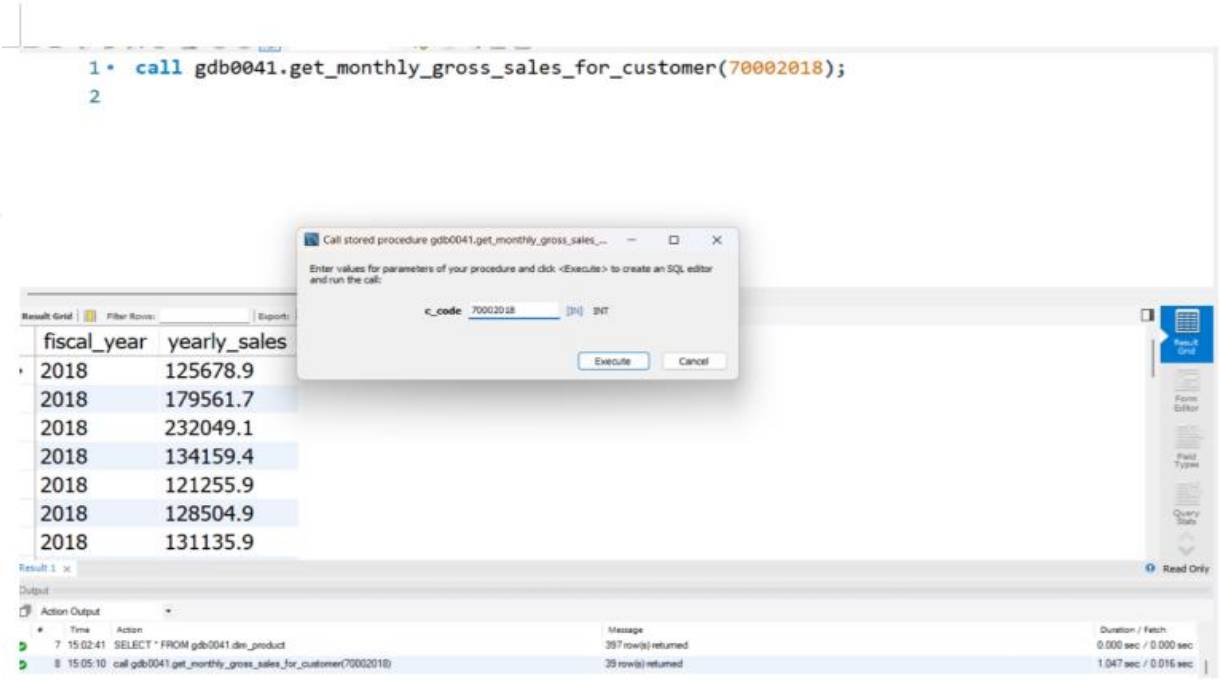
```sql
CREATE PROCEDURE `get_monthly_gross_sales_for_customer` (
            c_code INT
)
BEGIN
    SELECT
    get_fiscal_year(date) as fiscal_year,
    SUM(ROUND(g.gross_price*s.sold_quantity,1)) as yearly_sales
FROM fact_sales_monthly s
JOIN fact_gross_price g
ON
    s.product_code=g.product_code AND
    g.fiscal_year=get_fiscal_year(s.date)
WHERE
    customer_code=c_code
GROUP By date;
END
```

CONTINUED...

**Let's say If we want fiscal year too , then we will create another variable called fiscal_year.**

```
CREATE PROCEDURE `get_monthly_gross_sales_for_customer` (
            c_code INT,
            f_year YEAR
)BEGIN
    SELECT
    get_fiscal_year(date) as fiscal_year,
    SUM(ROUND(g.gross_price*s.sold_quantity,1)) as yearly_sales
FROM fact_sales_monthly s
JOIN fact_gross_price g
ON
    s.product_code=g.product_code AND
    g.fiscal_year=get_fiscal_year(s.date)
WHERE
    customer_code=c_code AND fiscal_year=f_year
GROUP By date;
```

Entered customer_code = 70002018



Add epic / AFA-18

# Stored proc for market badge

Attach | Add a child issue | Link issue | ...

**Description**

Create a stored proc that can determine the market badge based on the following logic.

If **total sold quantity > 5 million** that market is considered **Gold** else it is **Silver**

My input will be,

- market
- fiscal year

Output

- market badge

SIMPLY WROTE A QUERY FIRST FOR A SOLD QUANTITY ____

Start

```mysql
SELECT
    SUM(s.sold_quantity) as total_sold_quantity
FROM
    dim_customer c
JOIN fact_sales_monthly s
    ON
      c.customer_code=s.customer_code
    WHERE
    get_fiscal_year(s.date)=2021 and market="India"
GROUP By c.market;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| total_sold_quantity |
|---|
| 13751429 |

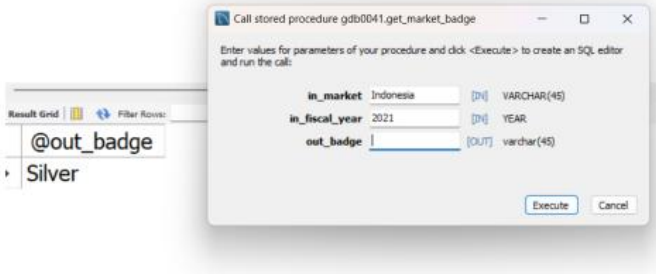
End

```mysql
CREATE PROCEDURE `get_market_badge` (
        IN in_market VARCHAR(45),
        IN in_fiscal_year YEAR,
        OUT out_badge varchar(45)
)
BEGIN
    DECLARE qty INT DEFAULT 0;

    #DETERMINE SOLD QUANTITY BY MARKET + FYEAR
    SELECT
    SUM(s.sold_quantity) INTO qty
FROM
    dim_customer c
JOIN fact_sales_monthly s
    ON
      c.customer_code=s.customer_code
    WHERE
    get_fiscal_year(s.date)=in_fiscal_year
    and c.market=in_market
GROUP By c.market;

    #DETERMINE MARKET BADGE
    IF qty>5000000 THEN
        SET out_badge = "Gold";
    ELSE
        SET out_badge = "Silver";
        END IF;
END
```

CONTINUED..

## Execution ____

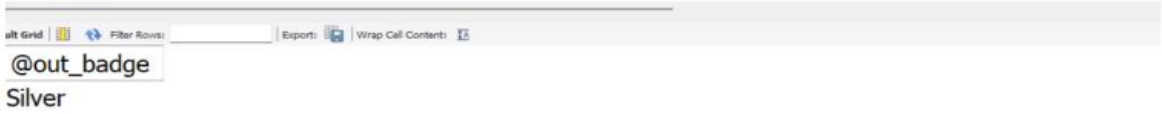
```
set @out_badge = '0';
call gdb0041.get_market_badge('Indonesia', 2021, @out_badge);
select @out_badge;
```




SET AS A DEFAULT MARKET _ INDIA

If we don't pass anything in market and just Fiscal year then we need to set as a default market .

```
set @out_badge = '0';
call gdb0041.get_market_badge('', 2021, @out_badge);
select @out_badge;
```



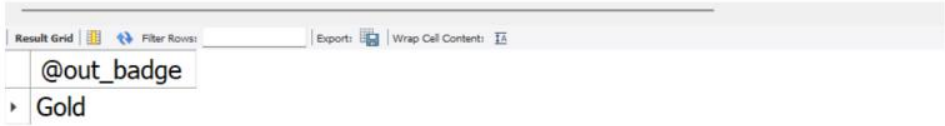

It is showing "Silver" market badge because it's going in Else body

```
BEGIN
    DECLARE qty INT DEFAULT 0;

    #SET DEFAULT MARKET TO BE INDIA
    IF in_market = "" THEN
        SET in_market="India";
    END IF;

    #DETERMINE SOLD QUANTITY BY MARKET + FYEAR
    SELECT
    SUM(s.sold_quantity) INTO qty
FROM
    dim customer c
```

OUTPUT

SEE NOW OUTPUT IS GOLD ____

```
set @out_badge = '0';
call gdb0041.get_market_badge('', 2021, @out_badge);
select @out_badge;
```



# SQL ADVANCED ____ TOP CUSTOMERS ,PRODUCTS, MARKETS

TOP CUSTOMERS ,MARKETS AND PRODUCTS

Projects / AtliQ Data Analytics / Add epic / AFA-19

## Top markets, products, customers for a given financial year

Attach | Add a child issue | Link issue | ...

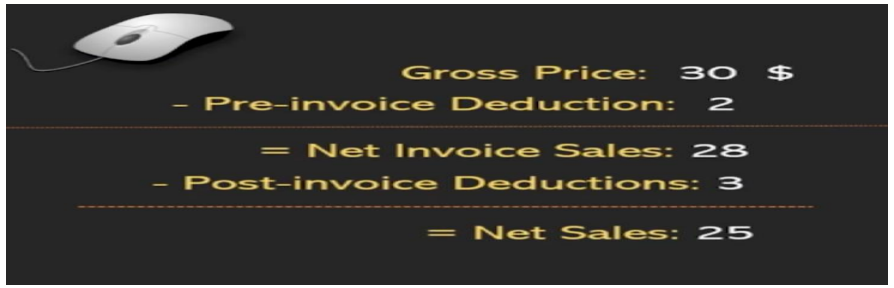
Description

**As a product owner**, I want a report for top markets, products, customers by net sales for a given financial year so that I can have a holistic view of our financial performance and can take appropriate actions to address any potential issues.

We will probably write stored proc for this as we will need this report going forward as well.

1. Report for **top markets**.

| Rank | Market | Net Sales (in millions) |
|---|---|---|
| 1 | India | 210.67 |
| 2 | USA | 132.05 |
| 3 | South Korea | 64.01 |




1. Report for **top markets**.

| Rank | Market | Net Sales (in millions) |
|---|---|---|
| 1 | India | 210.67 |
| 2 | USA | 132.05 |
| 3 | South Korea | 64.01 |

2. Report for **top products**.

| Rank | Product | Net Sales |
|---|---|---|
| 1 | AQ BZ Allin1 | 33.75 |
| 2 | AQ Qwerty | 27.84 |

CONTINUED....

```sql
-- Include pre-invoice deductions in Croma detailed report
SELECT
        s.date,
        s.product_code,
        p.product,
    p.variant,
        s.sold_quantity,
        g.gross_price as gross_price_per_item,
        ROUND(s.sold_quantity*g.gross_price,2) as gross_price_total,
        pre.pre_invoice_discount_pct
FROM fact_sales_monthly s
JOIN dim_product p
        ON s.product_code=p.product_code
JOIN fact_gross_price g
        ON g.fiscal_year=get_fiscal_year(s.date)

            AND g.product_code=s.product_code
    JOIN fact_pre_invoice_deductions as pre
            ON pre.customer_code = s.customer_code AND
            pre.fiscal_year=get_fiscal_year(s.date)
    WHERE
        s.customer_code=90002002 AND
            get_fiscal_year(s.date)=2021
    LIMIT 1000000;
```

OUTPUT

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

| date | product_code | product | variant | sold_quantity |
|---|---|---|---|---|
| 2020-09-01 | A0118150101 | AQ Dracula HD... | Stan... | 202 |
| 2020-10-01 | A0118150101 | AQ Dracula HD... | Stan... | 95 |
| 2020-12-01 | A0118150101 | AQ Dracula HD... | Stan... | 113 |
| 2021-01-01 | A0118150101 | AQ Dracula HD... | Stan... | 182 |
| 2021-02-01 | A0118150101 | AQ Dracula HD... | Stan... | 208 |
| 2021-04-01 | A0118150101 | AQ Dracula HD... | Stan... | 199 |
| 2021-05-01 | A0118150101 | AQ Dracula HD... | Stan... | 58 |
| 2021-06-01 | A0118150101 | AQ Dracula HD... | Stan... | 205 |
| 2021-08-01 | A0118150101 | AQ Dracula HD... | Stan... | 88 |
| 2020-09-01 | A0118150102 | AQ Dracula HD... | Plus | 162 |
| 2020-10-01 | A0118150102 | AQ Dracula HD... | Plus | 237 |
| 2020-12-01 | A0118150102 | AQ Dracula HD... | Plus | 172 |
| 2021-01-01 | A0118150102 | AQ Dracula HD... | Plus | 121 |
| 2021-02-01 | A0118150102 | AQ Dracula HD... | Plus | 77 |

Wrap Cell Content: | Fetch rows:

| gross_price_per_item | gross_price_total | pre_invoice_discount_pct |
|---|---|---|
| . 19.0573 | 3849.57 | 0.3025 |
| . 19.0573 | 1810.44 | 0.3025 |
| . 19.0573 | 2153.47 | 0.3025 |
| . 19.0573 | 3468.43 | 0.3025 |
| . 19.0573 | 3963.92 | 0.3025 |
| . 19.0573 | 3792.40 | 0.3025 |
| . 19.0573 | 1105.32 | 0.3025 |
| . 19.0573 | 3906.75 | 0.3025 |
| . 19.0573 | 1677.04 | 0.3025 |
| . 21.4565 | 3475.95 | 0.3025 |
| . 21.4565 | 5085.19 | 0.3025 |
| . 21.4565 | 3690.52 | 0.3025 |
| . 21.4565 | 2596.24 | 0.3025 |
| . 21.4565 | 1652.15 | 0.3025 |

# PERFORMANCE IMPROVEMENT #1

```sql
-- Include pre-invoice deductions in Croma detailed report
EXPLAIN ANALYZE
    SELECT
            s.date,
            s.product_code,
            p.product,
        p.variant,
            s.sold_quantity,
            g.gross_price as gross_price_per_item,
            ROUND(s.sold_quantity*g.gross_price,2) as gross_price_total,
            pre.pre_invoice_discount_pct
    FROM fact_sales_monthly s
    JOIN dim_product p
            ON s.product_code=p.product_code
     JOIN fact_gross_price g
            ON g.fiscal_year=get_fiscal_year(s.date)
            AND g.product_code=s.product_code
     JOIN fact_pre_invoice_deductions as pre
            ON pre.customer_code = s.customer_code AND
            pre.fiscal_year=get_fiscal_year(s.date)
     WHERE
         s.customer_code=90002002 AND
            get_fiscal_year(s.date)=2021
     LIMIT 1000000;
```

**QUERY OPTIMIZATION**

EXPLAIN ANALYZE

Binary | Text

```
-> Limit: 1000000 row(s)  (cost=43393 rows=43099) (actual time=3..916 rows=3006 loops=1)
  -> Nested loop inner join  (cost=43393 rows=43099) (actual time=3..915 rows=3006 loops=1)
    -> Nested loop inner join  (cost=161 rows=118) (actual time=0.1..3.25 rows=334 loops=1)
      -> Filter: (g.fiscal_year = 2021)  (cost=119 rows=118) (actual time=0.0784..0.945 rows=334 loops=1)
        -> Table scan on g  (cost=119 rows=1182) (actual time=0.074..0.766 rows=1182 loops=1)
      -> Single-row index lookup on p using PRIMARY (product_code=g.product_code)  (cost=0.251 rows=1)
(actual time=0.00638..0.00644 rows=1 loops=334)
    -> Filter: ((s.customer_code = 90002002) and (get_fiscal_year(s.`date`) = 2021))  (cost=1.44 rows=365)
(actual time=1.22..2.73 rows=9 loops=334)
      -> Index lookup on s using PRIMARY (product_code=g.product_code)  (cost=1.44 rows=3646) (actual
time=0.0469..2.15 rows=4082 loops=334)
```

_Limit : 1000000 rows___ actual time= 3 millisecond to fetch one row and 9sec to fetch all rows 3006

-Filter ((s.customer_code = 90002002) and ((get_fiscal_year(s.date)=2021)) (cost =1.44 rows = 365 )

(actual time = 1.22 ..2.73 rows =9 loops =334)

get_fiscal_year(s.date) is taking lot of time because it's scanning same date every time which is taking lot of time.

CONTINUED....

| date |
| --- |
| 2020-09-01 |
| 2020-09-01 |
| 2020-09-01 |
| 2020-09-01 |
| 2020-09-01 |
| 2020-09-01 |
| 2020-09-01 |
| 2020-09-01 |
| 2020-09-01 |
| 2020-09-01 |
| 2020-09-01 |
| 2020-09-01 |
| 2020-09-01 |

I am calling get_fiscal_year function which is calling date as an input and will do 4 months interval addition into it and then it will take YEAR out of it .

```
CREATE DEFINER=`root`@`localhost` FUNCTION `get_fiscal_year`(calendar_date
    DETERMINISTIC
BEGIN
    DECLARE fiscal_year INT;
    SET fiscal_year = YEAR(DATE_ADD(calendar_date, INTERVAL 4 MONTH));
    RETURN fiscal_year;
END
```

So that operation is going to take some time. And I am doing for all 1.4 million rows, that's what is taking most of the time and these dates are repetitive. Calling the same function repetitively is taking more time.

How will we solve this ??

- Will create a lookup table __ dim_date ---
  -2017-09-01 -> 2018
  -2017-10-01 ->2018
  -2018-09-01->2019

We will get a **Fiscal year** from it as a **Generated column**

CONTINUED……

1. Calendar date DATA TYPE – date
2. Fiscal year DATA TYPE - YEAR

-YEAR(DATE_ADD(calendar_date,INTERVAL 4 MONTH))

Now let's insert date and fiscal year however we will just insert dates into it and fiscal_year as a generated column will be automatically updated .

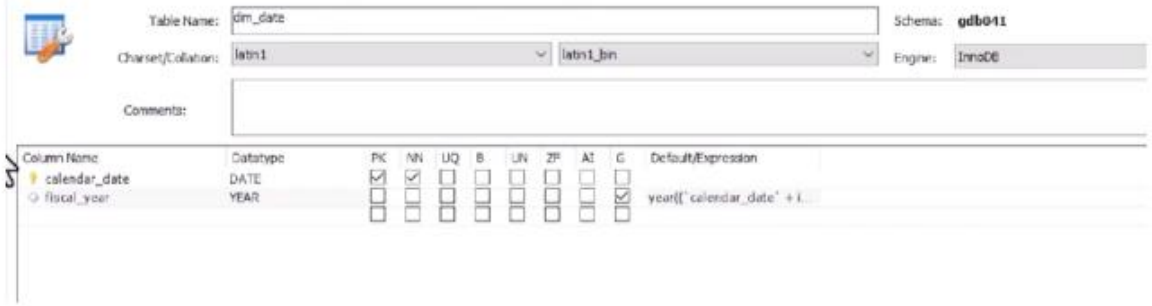
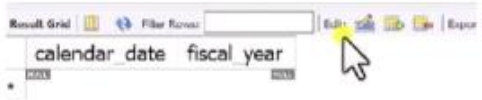
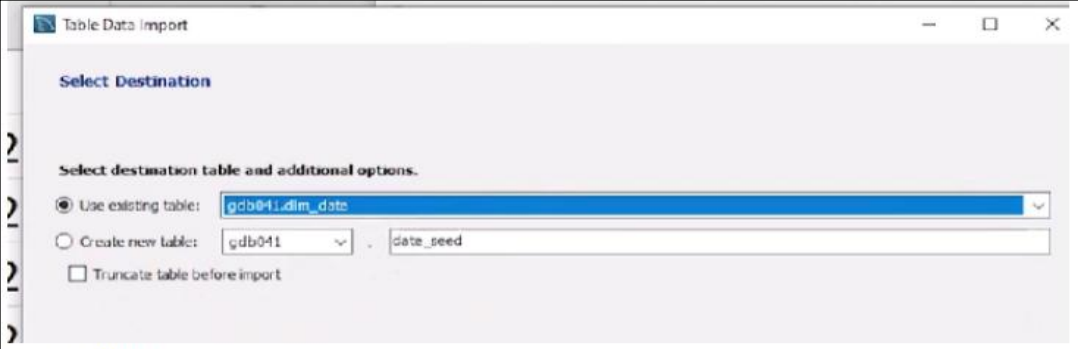



Table is now empty __




- Will insert possible dates into an excel file and will import those date into this sql file.
- First save into some place or download etc ..
- Seed file for dates as data_seed file I have saved and will import just date and not fiscal year because it will be generated from dates we have imported.
- Select import in output field.




-Click on Next


```
-- Include pre-invoice deductions in Croma detailed report
EXPLAIN ANALYZE
    SELECT
            s.date,
            s.product_code,
            p.product,
            p.variant,
            s.sold_quantity,
            g.gross_price as gross_price_per_item,
            ROUND(s.sold_quantity*g.gross_price,2) as gross_price_total,
            pre.pre_invoice_discount_pct
    FROM fact_sales_monthly s
    JOIN dim_date dt
     ON dt.calendar_date= s.date
     JOIN dim_product p
            ON s.product_code=p.product_code
     JOIN fact_gross_price g
            ON g.fiscal_year=dt.fiscal_year
            AND g.product_code=s.product_code
     JOIN fact_pre_invoice_deductions as pre
            ON pre.customer_code = s.customer_code AND
            pre.fiscal_year=dt.fiscal_year
     WHERE
         s.customer_code=90002002 AND
         dt.fiscal_year=2018 LIMIT 1000000;
```

Now time has been reduced ____

**DATE FORMAT should be YY-MM-DD otherwise date data won't be imported .**

```
-> Limit: 1000000 row(s)  (cost=276 rows=30.7) (actual time=1.71..13.4 rows=624 loops=1)
    -> Nested loop inner join  (cost=276 rows=30.7) (actual time=1.71..13.4 rows=624 loops=1)
        -> Inner hash join (no condition)  (cost=195 rows=30.7) (actual time=1.68..2.21 rows=1044 loops=1)
            -> Filter: (dt.fiscal_year = 2018)  (cost=0.0275 rows=2.6) (actual time=0.0284..0.0999 rows=12 loops=1)
                -> Table scan on dt  (cost=0.0275 rows=26) (actual time=0.0278..0.0846 rows=26 loops=1)
            -> Hash
                -> Nested loop inner join  (cost=161 rows=118) (actual time=0.1..1.58 rows=87 loops=1)
                    -> Filter: (g.fiscal_year = 2018)  (cost=119 rows=118) (actual time=0.0777..1.18 rows=87 loops=1)
                        -> Table scan on g  (cost=119 rows=1182) (actual time=0.0758..1.02 rows=1182 loops=1)
                    -> Single-row index lookup on p using PRIMARY (product_code=g.product_code)  (cost=0.251
rows=1) (actual time=0.00425..0.00431 rows=1 loops=87)
        -> Single-row index lookup on s using PRIMARY (product_code=g.product_code, date=dt.calendar_date,
customer_code=90002002)  (cost=0.257 rows=1) (actual time=0.0104..0.0104 rows=0.598 loops=1044)
```

The join we perform it acted kind of a **HashMap** and it saved us a lot of time.
We have lot of ways to improve performance —
Sometimes we can use **calculated column , index and a lookup table** etc.

# Performance Improvement # 2_

Why don't we just create a fiscal year column in fact_sales_monthly table ??

| date | product_code | customer_code | sold_quantity |
|---|---|---|---|
| 2017-09-01 | A0118150101 | 70002017 | 51 |
| 2017-09-01 | A0118150101 | 70002018 | 77 |
| 2017-09-01 | A0118150101 | 70003181 | 17 |
| 2017-09-01 | A0118150101 | 70003182 | 6 |
| 2017-09-01 | A0118150101 | 70006157 | 5 |
| 2017-09-01 | A0118150101 | 70006158 | 7 |
| 2017-09-01 | A0118150101 | 70007198 | 29 |
| 2017-09-01 | A0118150101 | 70007199 | 34 |
| 2017-09-01 | A0118150101 | 70008169 | 22 |
| 2017-09-01 | A0118150101 | 70008170 | 5 |
| 2017-09-01 | A0118150101 | 70011193 | 10 |
| 2017-09-01 | A0118150101 | 70011194 | 4 |
| 2017-09-01 | A0118150101 | 70012042 | 0 |

-But in that table millions of records are present so It will consume lot of storage but storage is cheap , so we can keep fiscal year column in here rather than having a separate table dim_date and we don't have to use an extra JOIN in that case.

Also we don't have to call any extra function get_fiscal_year etc.

| date | product_code | customer_code | sold | fiscal_year |
|---|---|---|---|---|
| 2017-09-01 | A0118150101 | 70002017 | 51 | 2018 |
| 2017-09-01 | A0118150101 | 70002018 | 77 | 2018 |
| 2017-09-01 | A0118150101 | 70003181 | 17 | 2018 |
| 2017-09-01 | A0118150101 | 70003182 | 6 | 2018 |
| 2017-09-01 | A0118150101 | 70006157 | 5 | 2018 |
| 2017-09-01 | A0118150101 | 70006158 | 7 | 2018 |
| 2017-09-01 | A0118150101 | 70007198 | 29 | 2018 |
| 2017-09-01 | A0118150101 | 70007199 | 34 | 2018 |
| 2017-09-01 | A0118150101 | 70008169 | 22 | 2018 |
| 2017-09-01 | A0118150101 | 70008170 | 5 | 2018 |
| 2017-09-01 | A0118150101 | 70011193 | 10 | 2018 |

Table Name: fact_sales_monthly

Charset/Collation: latin1 latin1_bin

Comments:

| Column Name | Datatype | PK | NN | UQ | B | UN | ZF | AI | G | Default/Expression |
|---|---|---|---|---|---|---|---|---|---|---|
| date | DATE | ☑ | ☑ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | |
| product_code | VARCHAR(45) | ☑ | ☑ | ☐ | ☐ | ☐ | ☐ | ☐ | ☑ | |
| customer_code | INT | ☑ | ☑ | ☐ | ☐ | ☑ | ☐ | ☐ | ☐ | |
| sold_quantity | INT | ☐ | ☑ | ☐ | ☐ | ☑ | ☐ | ☐ | ☐ | |
| fiscal_year | YEAR | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☑ | year((`date` + interval 4... |
| | | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | |

```sql
-- Include pre-invoice deductions in Croma detailed report
SELECT
        s.date,
        s.product_code,
        p.product,
        p.variant,
        s.sold_quantity,
        g.gross_price as gross_price_per_item,
        ROUND(s.sold_quantity*g.gross_price,2) as gross_price_total,
        pre.pre_invoice_discount_pct
FROM fact_sales_monthly s
JOIN dim_product p
        ON s.product_code=p.product_code
JOIN fact_gross_price g
        ON g.fiscal_year=s.fiscal_year
        AND g.product_code=s.product_code
JOIN fact_pre_invoice_deductions as pre
        ON pre.customer_code = s.customer_code AND
        pre.fiscal_year=s.fiscal_year
WHERE
    s.fiscal_year=2021
    LIMIT 1000000;
```

Essentially we are deriving fiscal_year from date column in fact_sales_monthly table.

Now we will change the query and replace date table dt from s.fiscal year__

| date | product_code | product | variant | sold_quantity | gross_price_per_item | gross_price_total | pre_invoice_discount_pct |
|---|---|---|---|---|---|---|---|
| 2020-09-01 | A0118150101 | AQ Dracula H... | Standard | 248 | 19.0573 | 4726.21 | 0.0703 |
| 2020-09-01 | A0118150101 | AQ Dracula H... | Standard | 240 | 19.0573 | 4573.75 | 0.2061 |
| 2020-09-01 | A0118150101 | AQ Dracula H... | Standard | 31 | 19.0573 | 590.78 | 0.0974 |
| 2020-09-01 | A0118150101 | AQ Dracula H... | Standard | 37 | 19.0573 | 705.12 | 0.2065 |
| 2020-09-01 | A0118150101 | AQ Dracula H... | Standard | 7 | 19.0573 | 133.40 | 0.1068 |
| 2020-09-01 | A0118150101 | AQ Dracula H... | Standard | 12 | 19.0573 | 228.69 | 0.2612 |
| 2020-09-01 | A0118150101 | AQ Dracula H... | Standard | 17 | 19.0573 | 323.97 | 0.2471 |
| 2020-09-01 | A0118150101 | AQ Dracula H... | Standard | 60 | 19.0573 | 1143.44 | 0.0858 |
| 2020-09-01 | A0118150101 | AQ Dracula H... | Standard | 34 | 19.0573 | 647.95 | 0.2450 |
| 2020-09-01 | A0118150101 | AQ Dracula H... | Standard | 24 | 19.0573 | 457.38 | 0.0736 |

TIME HAS REDUCED MORE ____

Binary Text

```
-> Limit: 1000000 row(s)  (cost=58475 rows=43099) (actual time=1.31..1051 rows=608108 loops=1)
   -> Nested loop inner join  (cost=58475 rows=43099) (actual time=1.31..1023 rows=608108 loops=1)
      -> Nested loop inner join  (cost=43390 rows=43099) (actual time=1.31..519 rows=608108 loops=1)
         -> Nested loop inner join  (cost=161 rows=118) (actual time=0.0689..1.16 rows=334 loops=1)
            -> Filter: (g.fiscal_year = 2021)  (cost=119 rows=118) (actual time=0.0544..0.426 rows=334 loops=1)
               -> Table scan on g  (cost=119 rows=1182) (actual time=0.0516..0.35 rows=1182 loops=1)
            -> Single-row index lookup on p using PRIMARY (product_code=g.product_code)  (cost=0.251 rows=1)
(actual time=0.00205..0.00207 rows=1 loops=334)
         -> Filter: (s.fiscal_year = 2021)  (cost=1.42 rows=365) (actual time=0.538..1.47 rows=1821 loops=334)
            -> Index lookup on s using PRIMARY (product_code=g.product_code)  (cost=1.42 rows=3646) (actual
time=0.0188..1.29 rows=4082 loops=334)
      -> Single-row index lookup on pre using PRIMARY (customer_code=s.customer_code, fiscal_year=2021)
(cost=0.25 rows=1) (actual time=689e-6..711e-6 rows=1 loops=608108)
```

- Now we need **Net Invoice sales ___**

# DATABASE VIEWS INTRODUCTION

```sql
-- Net Invoice sales calculated
    WITH CTE1 as
    (SELECT
            s.date,
            s.product_code,
            p.product,
            p.variant,
            s.sold_quantity,
            g.gross_price as gross_price_per_item,
            ROUND(s.sold_quantity*g.gross_price,2) as gross_price_total,
            pre.pre_invoice_discount_pct
    FROM fact_sales_monthly s
    JOIN dim_product p
            ON s.product_code=p.product_code
    JOIN fact_gross_price g
            ON g.fiscal_year=s.fiscal_year
            AND g.product_code=s.product_code
    JOIN fact_pre_invoice_deductions as pre
            ON pre.customer_code = s.customer_code AND
            pre.fiscal_year=s.fiscal_year
    WHERE
        s.fiscal_year=2021)
    SELECT * ,
    (gross_price_total-gross_price_total*pre_invoice_discount_pct) AS
    net_invoice_sales
    FROM CTE1
```

| date | product_code | product | variant | sold_quantity | gross_price_per_item | gross_price_total | pre_invoice_discount_pct | net_invoice_sales |
|---|---|---|---|---|---|---|---|---|
| 2020-09-01 | A0118150101 | AQ Dracula HDD - ... | Standard | 248 | 19.0573 | 4726.21 | 0.0703 | 4393.957437 |
| 2020-09-01 | A0118150101 | AQ Dracula HDD - ... | Standard | 240 | 19.0573 | 4573.75 | 0.2061 | 3631.100125 |
| 2020-09-01 | A0118150101 | AQ Dracula HDD - ... | Standard | 31 | 19.0573 | 590.78 | 0.0974 | 533.238028 |
| 2020-09-01 | A0118150101 | AQ Dracula HDD - ... | Standard | 37 | 19.0573 | 705.12 | 0.2065 | 559.512720 |
| 2020-09-01 | A0118150101 | AQ Dracula HDD - ... | Standard | 7 | 19.0573 | 133.40 | 0.1068 | 119.152880 |
| 2020-09-01 | A0118150101 | AQ Dracula HDD - ... | Standard | 12 | 19.0573 | 228.69 | 0.2612 | 168.956172 |
| 2020-09-01 | A0118150101 | AQ Dracula HDD - ... | Standard | 17 | 19.0573 | 323.97 | 0.2471 | 243.917013 |
| 2020-09-01 | A0118150101 | AQ Dracula HDD - ... | Standard | 60 | 19.0573 | 1143.44 | 0.0858 | 1045.332848 |
| 2020-09-01 | A0118150101 | AQ Dracula HDD - ... | Standard | 34 | 19.0573 | 647.95 | 0.2450 | 489.202250 |
| 2020-09-01 | A0118150101 | AQ Dracula HDD - ... | Standard | 24 | 19.0573 | 457.38 | 0.0736 | 423.716832 |

**Derived columns cannot be used in the same query for calculations ....that's why CTE is used.**

- Now we need Post_invoice_deductions to calculate Net sales in a table but it will be lengthy so we will create a **database view** of this table called virtual table which will not be an actual/physical table.
- Now we need market column as well so we will join dim_customer table as well in the same query.

**JOIN dim_customer c**

**ON c.customer_code=s.customer_code**

For market column as c.market **.**

Continued...

```sql
CREATE VIEW `sales_pre_inv_discount` AS
SELECT
    s.date,
    s.fiscal_year,
    s.product_code,
    p.product,
    c.market,
    p.variant,
    s.sold_quantity,
    g.gross_price as gross_price_per_item,
    ROUND(s.sold_quantity*g.gross_price,2) as gross_price_total,
    pre.pre_invoice_discount_pct
        FROM fact_sales_monthly s
    JOIN dim_customer c
            ON c.customer_code=s.customer_code
    JOIN dim_product p
            ON s.product_code=p.product_code
    JOIN fact_gross_price g
            ON g.fiscal_year=s.fiscal_year
    AND g.product_code=s.product_code
    JOIN fact_pre_invoice_deductions as pre
            ON pre.customer_code = s.customer_code AND
            pre.fiscal_year=s.fiscal_year
```

CREATED A VIEW

```sql
SELECT * FROM gdb0041.sales_pre_inv_discount;
```

| date | fiscal_year | product_code | product | market | variant | sold_quantity | gross_price_per_item | gross_price_total | pre_invoice_discount_p |
|---|---|---|---|---|---|---|---|---|---|
| 2017-09-01 | 2018 | A0118150101 | AQ Dracula HDD – ... | India | Standard | 51 | 15.3952 | 785.16 | 0.0824 |
| 2017-09-01 | 2018 | A0118150101 | AQ Dracula HDD – ... | India | Standard | 77 | 15.3952 | 1185.43 | 0.2956 |
| 2017-09-01 | 2018 | A0118150101 | AQ Dracula HDD – ... | Indonesia | Standard | 17 | 15.3952 | 261.72 | 0.0536 |
| 2017-09-01 | 2018 | A0118150101 | AQ Dracula HDD – ... | Indonesia | Standard | 6 | 15.3952 | 92.37 | 0.2378 |
| 2017-09-01 | 2018 | A0118150101 | AQ Dracula HDD – ... | Philiphines | Standard | 5 | 15.3952 | 76.98 | 0.1057 |
| 2017-09-01 | 2018 | A0118150101 | AQ Dracula HDD – ... | Philiphines | Standard | 7 | 15.3952 | 107.77 | 0.1875 |
| 2017-09-01 | 2018 | A0118150101 | AQ Dracula HDD – ... | South Korea | Standard | 29 | 15.3952 | 446.46 | 0.0700 |
| 2017-09-01 | 2018 | A0118150101 | AQ Dracula HDD – ... | South Korea | Standard | 34 | 15.3952 | 523.44 | 0.2551 |
| 2017-09-01 | 2018 | A0118150101 | AQ Dracula HDD – ... | Australia | Standard | 22 | 15.3952 | 338.69 | 0.0953 |

# POST_INVOICE_DISCOUNT VIEW

```sql
CREATE VIEW `sales_postinv_discount` AS
SELECT
    s.date, s.fiscal_year,
    s.customer_code,s.market,
    s.product_code,s.product,s.variant,
    s.sold_quantity,s.gross_price_total,
    s.pre_invoice_discount_pct,
    (1-pre_invoice_discount_pct)*gross_price_total AS
    net_invoice_sales,
    (po.discounts_pct+po.other_deductions_pct)
    as post_invoice_discount_pct
FROM sales_pre_inv_discount s
    JOIN fact_post_invoice_deductions po
ON po.date=s.date AND
    po.product code=s.product code AND
    po.customer_code=s.customer_code;
```

| date | fiscal_year | customer_code | market | product_code | product |
|---|---|---|---|---|---|
| 2017-09-01 | 2018 | 70002017 | India | A0118150101 | AQ Dracula HDD – . |
| 2017-09-01 | 2018 | 70002018 | India | A0118150101 | AQ Dracula HDD – . |
| 2017-09-01 | 2018 | 70003181 | Indonesia | A0118150101 | AQ Dracula HDD – . |
| 2017-09-01 | 2018 | 70003182 | Indonesia | A0118150101 | AQ Dracula HDD – . |
| 2017-09-01 | 2018 | 70006157 | Philiphines | A0118150101 | AQ Dracula HDD – . |
| 2017-09-01 | 2018 | 70006158 | Philiphines | A0118150101 | AQ Dracula HDD – . |
| 2017-09-01 | 2018 | 70007198 | South Korea | A0118150101 | AQ Dracula HDD – . |
| 2017-09-01 | 2018 | 70007199 | South Korea | A0118150101 | AQ Dracula HDD – . |
| 2017-09-01 | 2018 | 70008169 | Australia | A0118150101 | AQ Dracula HDD – . |
| 2017-09-01 | 2018 | 70008170 | Australia | A0118150101 | AQ Dracula HDD – . |
| 2017-09-01 | 2018 | 70011193 | France | A0118150101 | AQ Dracula HDD – . |
| 2017-09-01 | 2018 | 70011194 | France | A0118150101 | AQ Dracula HDD – |

- **Now need to create Net Sales __**

```sql
SELECT
    *,
    (1-post_invoice_discount_pct)*net_invoice_sales as net_sales
FROM gdb041.sales_postinv_discount;
```

| | variant | sold_quantity | gross_price_total | pre_invoice_discount_pct | net_invoice_sales | post_invoice_discount_pct | net_sales |
|---|---|---|---|---|---|---|---|
| D – 3.5 Inc... | Standard | 4 | 61.58 | 0.2803 | 44.319126 | 0.3905 | 27.0125072970 |
| D – 3.5 Inc... | Standard | 16 | 246.32 | 0.2803 | 177.276504 | 0.4139 | 103.9017589944 |
| D – 3.5 Inc... | Standard | 4 | 61.58 | 0.2803 | 44.319126 | 0.3295 | 29.7159739830 |
| D – 3.5 Inc... | Standard | 6 | 92.37 | 0.2803 | 66.478689 | 0.3244 | 44.9130022884 |
| D – 3.5 Inc... | Standard | 9 | 138.56 | 0.2803 | 99.721632 | 0.3766 | 62.1664653888 |
| D – 3.5 Inc... | Standard | 6 | 92.37 | 0.2803 | 66.478689 | 0.3615 | 42.4466420 |
| D – 3.5 Inc... | Standard | 7 | 107.77 | 0.2803 | 77.562069 | 0.3173 | 52.95162 |
| D – 3.5 Inc... | Standard | 10 | 153.95 | 0.2803 | 110.797815 | 0.3501 | 72.00749 |
| D – 3.5 Inc... | Standard | 6 | 92.37 | 0.2803 | 66.478689 | 0.3740 | 41.61565 |
| D – 3.5 Inc... | Standard | 4 | 61.58 | 0.2117 | 48.543514 | 0.2863 | 34.64550 |
| D – 3.5 Inc... | Standard | 2 | 30.79 | 0.2117 | 24.271757 | 0.2851 | 17.35187 |
| D – 3.5 Inc... | Standard | 3 | 46.19 | 0.2117 | 36.411577 | 0.2882 | 25.91776 |
| D – 3.5 Inc... | Standard | 5 | 76.98 | 0.2117 | 60.683334 | 0.3334 | 40.45151 |
| D – 3.5 Inc... | Standard | 1 | 15.40 | 0.2117 | 12.139820 | 0.3296 | 8.138535 |

**Will create one more view as net_sales**

**CREATE VIEW AS**

**SELECT *,**

**(1-post_invoice_discount_pct)*net_invoice_sales as net_sales**

**FROM gdb0041.sales_postinv_discount;**

Continue..

Create a view for gross sales. It should have the following columns,

date, fiscal_year, customer_code, customer, market, product_code, product, variant, sold_quanity, gross_price_per_item, gross_price_total

```sql
SELECT
    s.date,
    s.fiscal_year,
    s.customer_code,
    c.customer,
    c.market,
    s.product_code,
    p.product,
    p.variant,
    s.sold_quantity,
    g.gross_price as gross_price_per_item,
    ROUND(s.sold_quantity*g.gross_price,2) as gross_price_total
FROM

fact_sales_monthly s
JOIN fact_gross_price g
ON s.product_code=g.product_code AND
s.fiscal_year=g.fiscal_year
JOIN dim_customer c
ON c.customer_code=s.customer_code
JOIN dim_product p
ON p.product_code= s.product_code;
```

# TOP 5 MARKETS

```sql
SELECT
    market,
    ROUND(SUM(net_sales)/1000000,2)as net_sal_mln
FROM net_sales
WHERE fiscal_year=2021
GROUP BY market
ORDER BY net_sal_mln DESC
LIMIT 5;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

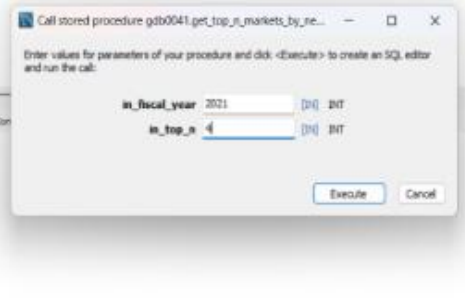
| market | net_sal_mln |
|---|---|
| India | 210.67 |
| USA | 132.05 |
| South Korea | 64.01 |
| Canada | 45.89 |
| United Kingdom | 44.73 |

**We need to create STORED PROCEDURE because this report needs every time.**

# TOP N MARKETS BY NET SALES IN MILLIONS IN A GIVEN FISCAL YEAR

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `get_top_n_markets_by_net_sa
        in_fiscal_year INT,
        in_top_n INT
    )
BEGIN
    SELECT
    market,
    ROUND(SUM(net_sales)/1000000,2)as net_sal_mln
FROM net_sales
WHERE fiscal_year=in_fiscal_year
GROUP BY market
ORDER BY net_sal_mln DESC
LIMIT in_top_n;
END
```

```
call gdb0041.get_top_n_markets_by_net_sales(2021, 4);
```

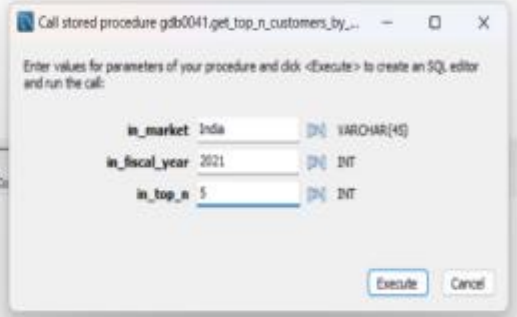


| market | net_sal_mln |
|---|---|
| India | 210.67 |
| USA | 132.05 |
| South Korea | 64.01 |
| Canada | 45.89 |

# TOP N CUSTOMERS BY NETSALES MARKETWISE

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `get_top_n_customers_by_net
        in_market VARCHAR(45),
        in_fiscal_year INT,
        in_top_n INT
)
BEGIN
    SELECT
    customer,
    ROUND(SUM(net_sales)/1000000,2)as net_sal_mln
FROM net_sales s
JOIN dim_customer c
ON c.customer_code=s.customer_code
WHERE s.fiscal_year= in_fiscal_year
AND s.market=in_market
GROUP BY customer
ORDER BY net_sal_mln DESC
LIMIT in_top_n;
END
```

```
call gdb0041.get_top_n_customers_by_net_sales('india', 2021, 5);
```



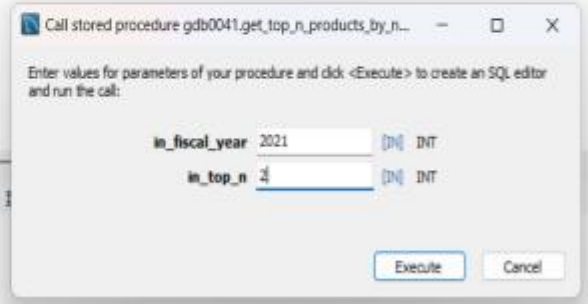

Result Grid | Filter Rows | Export | Wrap Cell Co

| customer | net_sal_mln |
|---|---|
| Amazon | 30.00 |
| Atliq Exclusive | 23.98 |
| Flipkart | 12.96 |
| Electricalsocity | 12.31 |
| Propel | 11.86 |

# Top N products by net sales for a given year.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `get_top_n_products_by_net_s
            in_fiscal_year INT,
            in_top_n INT
)
BEGIN
    SELECT
    product,
    ROUND(SUM(net_sales)/1000000 ,2) as net_sal_mln
    FROM net_sales s
    WHERE s.fiscal_year=in_fiscal_year
    GROUP BY product
    ORDER BY net_sal_mln DESC
    LIMIT in_top_n;
END
```

```
call gdb0041.get_top_n_products_by_net_sales(2021, 2);
```

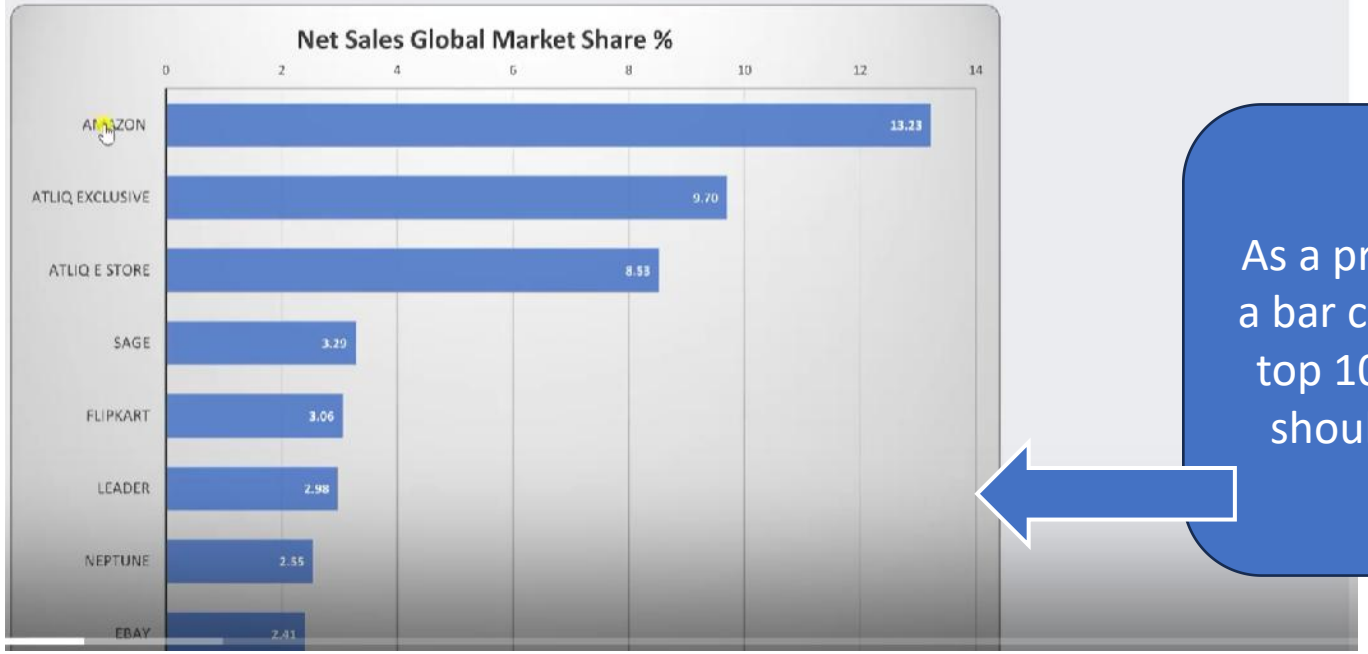



Filter Rows: Export: Wrap Cell Content:

| product | net_sal_mln |
|---|---|
| AQ BZ Allin1 | 33.75 |
| AQ Qwerty | 27.84 |

Description

As a product owner, I want to see a bar chart report for FY=2021 for top 10 markets by % net sales. It should look something like this.




As a product owner, I want to see a bar chart report for FY =2021 for top 10 markets by % net sales. It should look something like this

```
WITH CTE as (
SELECT
    customer,
    ROUND(SUM(net_sales)/1000000,2)as net_sal_mln
FROM net_sales s
JOIN dim_customer c
ON c.customer_code=s.customer_code
WHERE s.fiscal_year= 2021
GROUP BY customer)
SELECT *,net_sal_mln*100/sum(net_sal_mln) over() as pct_share_overall FROM CTE
ORDER BY net_sal_mln DESC;
```

OUTPUT

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| customer | net_sal_mln | pct_share_overall |
|---|---|---|
| Amazon | 109.03 | 13.233402 |
| Atliq Exclusive | 79.92 | 9.700206 |
| Atliq e Store | 70.31 | 8.533803 |
| Sage | 27.07 | 3.285593 |
| Flipkart | 25.25 | 3.064692 |
| Leader | 24.52 | 2.976089 |
| Neptune | 21.01 | 2.550067 |
| Ebay | 19.88 | 2.412914 |
| Electricalsocity | 16.25 | 1.972327 |
| Synthetic | 16.10 | 1.954121 |
| Electricalslyti... | 15.64 | 1.898289 |
| Acclaimed St... | 14.32 | 1.738075 |

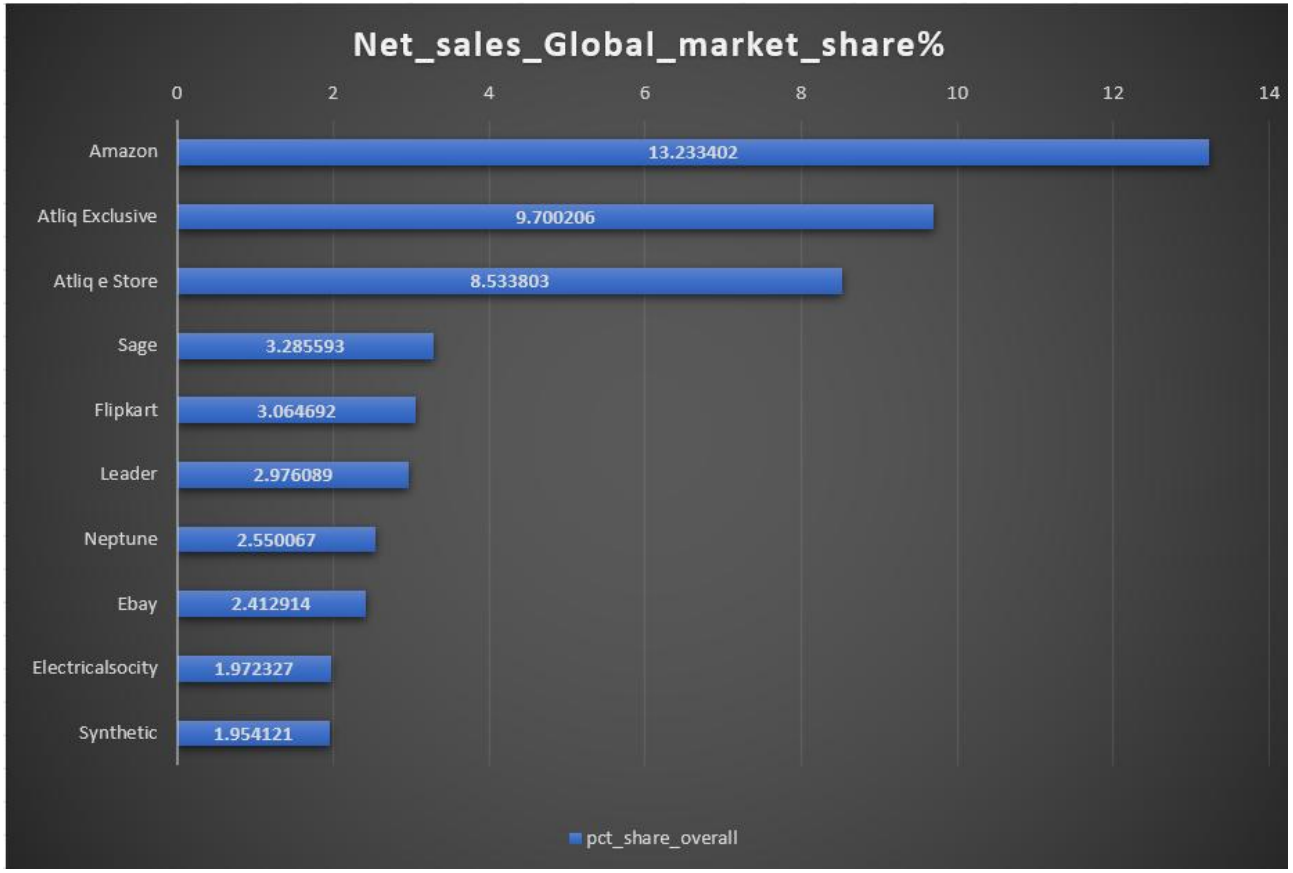
Continued..

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| customer | net_sal_mln | pct_share_overall |
|---|---|---|
| Amazon | 109.03 | 13.233402 |
| Atliq Exclusive | 79.92 | 9.700206 |
| Atliq e Store | 70.31 | 8.533803 |
| Sage | 27.07 | 3.285593 |
| Flipkart | 25.25 | 3.064692 |
| Leader | 24.52 | 2.976089 |
| Neptune | 21.01 | 2.550067 |
| Ebay | 19.88 | 2.412914 |
| Electricalsocity | 16.25 | 1.972327 |
| Synthetic | 16.10 | 1.954121 |
| Electricalslyti... | 15.64 | 1.898289 |
| Acclaimed St... | 14.32 | 1.738075 |

**OUTPUT**

**BAR CHART**



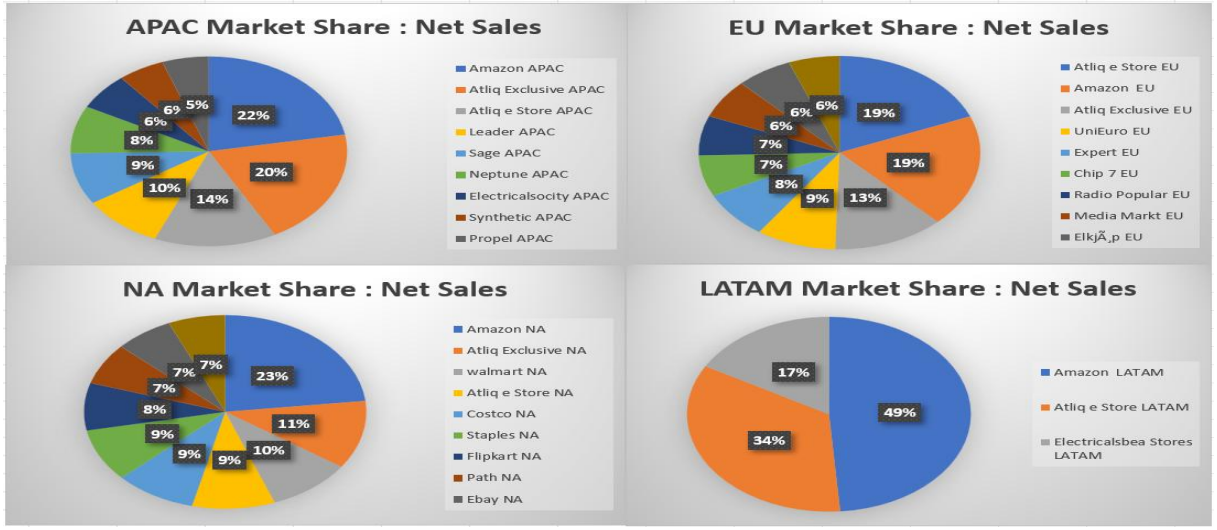
# AD HOC REQUEST

As a product owner I want to see region wise (APAC, EU, LATAM etc ) % net sales breakdown by customers in a respective region so that I can perform my regional analysis on financial performance of the company.

The end results should be bar charts in the following format for FY= 2021 , Build a reusable asset that we can use to conduct this analysis for any financial year.

```sql
WITH CTE as (
SELECT
    c.customer,
    c.region,
    ROUND(SUM(net_sales)/1000000,2)as net_sal_mln
FROM net_sales s JOIN dim_customer c
ON c.customer_code=s.customer_code
WHERE s.fiscal_year= 2021
GROUP BY c.customer,c.region )

SELECT * ,net_sal_mln*100/SUM(net_sal_mln) over(partition by region)  as
pct_share_region
FROM CTE ORDER BY region , net_sal_mln DESC;
```

| customer | region | net_sal_mln | pct_share_region |
|---|---|---|---|
| Amazon | APAC | 57.41 | 12.988688 |
| Atliq Exclusive | APAC | 51.58 | 11.669683 |
| Atliq e Store | APAC | 36.97 | 8.364253 |
| Leader | APAC | 24.52 | 5.547511 |
| Sage | APAC | 22.85 | 5.169683 |
| Neptune | APAC | 21.01 | 4.753394 |
| Electricalsocity | APAC | 16.25 | 3.676471 |
| Synthetic | APAC | 14.14 | 3.199095 |
| Propel | APAC | 14.14 | 3.199095 |
| Flipkart | APAC | 12.96 | 2.932127 |
| Novus | APAC | 12.91 | 2.920814 |
| Expression | APAC | 12.90 | 2.918552 |



## Get top n products in each division by their quantity sold

Attach | Add a child issue | Link issue | ...

Description

Write a stored proc for getting TOP n products in each division by their quantity sold in a given financial year. For example below would be the result for FY=2021.

| Division | Product | Total Quantity |
|---|---|---|
| N & S | AQ Pen Drive DRC | 2034569 |
| N & S | AQ Digit SSD | 1240149 |
| N & S | AQ Clx1 | 1238683 |
| P & A | AQ Gamers Ms | 2477098 |
| P & A | AQ Maxima Ms | 2461991 |
| P & A | AQ Master wireless x1 Ms | 2448784 |
| PC | AQ Digit | 135092 |
| PC | AQ Gen Y | 135031 |
| PC | AQ Elite | 134431 |

USING CTE(Common Table expressions)

```
WITH CTE AS (
SELECT
    p.division,
    p.product,
    SUM(sold_quantity) as total_qty
    FROM dim_product p
JOIN fact_sales_monthly s
ON p.product_code=s.product_code
WHERE fiscal_year=2021
GROUP BY product ORDER BY total_qty DESC),
CTE1 as
(SELECT *,
dense_rank() over(partition by division order by total_qty DESC) as drnk
FROM CTE)
SELECT * FROM CTE1 WHERE drnk <=3;
```

Output

| division | product | total_qty | drnk |
|---|---|---|---|
| N & S | AQ Pen Drive DRC | 2034569 | 1 |
| N & S | AQ Digit SSD | 1240149 | 2 |
| N & S | AQ Clx1 | 1238683 | 3 |
| P & A | AQ Gamers Ms | 2477098 | 1 |
| P & A | AQ Maxima Ms | 2461991 | 2 |
| P & A | AQ Master wireless x1 Ms | 2448784 | 3 |
| PC | AQ Digit | 135092 | 1 |
| PC | AQ Gen Y | 135031 | 2 |
| PC | AQ Elite | 134431 | 3 |

# STORED PROCEDURE

```
CREATE DEFINER=`root`@`localhost` PROCEDURE
 `get_top_n_products_per_div_by_qty_sold`(
            in_top_n INT,
            in_fiscal_year INT
)
BEGIN
    WITH CTE AS (
SELECT
    p.division,
    p.product,
    SUM(sold_quantity) as total_qty
    FROM dim_product p
JOIN fact_sales_monthly s
ON p.product_code=s.product_code
WHERE fiscal_year=in_fiscal_year
GROUP BY product ORDER BY total_qty DESC),
CTE1 as
(SELECT *,
dense_rank() over(partition by division order by total_qty DESC)
as drnk
FROM CTE)
SELECT * FROM CTE1 WHERE drnk <=in_top_n;
END
```

```
call gdb0041.get_top_n_products_per_div_by_qty_sold(4, 2020);
```

| division | product | total_qty | drnk |
|---|---|---|---|
| N & S | AQ Clx1 | 935128 | 1 |
| N & S | AQ Neuer SSD | 924264 | 2 |
| N & S | AQ Digit SSD | 920105 | 3 |
| N & S | AQ Wi Power Dx2 | 846576 | 4 |
| P & A | AQ Master wired x1 Ms | 1578253 | 1 |
| P & A | AQ Gamers Ms | 1566445 | 2 |
| P & A | AQ Lite Ms | 1564099 | 3 |
| P & A | AQ Master wireless x1 Ms | 1563844 | 4 |
| PC | AQ Digit | 68862 | 1 |
| PC | AQ Elite | 67841 | 2 |
| PC | AQ Aspiron | 59516 | 3 |
| PC | AQ BZ Compact | 52380 | 4 |

Using window and dense rank functions where we can find any top n products along with fiscal year in each division by their qty sold

Retrieve the top 2 markets in every region by their gross sales amount in FY=2021. i.e. result should look something like this,

| market | region | gross_sales_mln | rnk |
|---|---|---|---|
| India | APAC | 455.05 | 1 |
| South Korea | APAC | 131.86 | 2 |
| United Kingdom | EU | 78.11 | 1 |
| France | EU | 67.62 | 2 |
| Mexico | LATAM | 2.30 | 1 |
| Brazil | LATAM | 2.14 | 2 |
| USA | NA | 264.46 | 1 |
| Canada | NA | 89.78 | 2 |

WITH VIEW

```sql
WITH CTE AS (
SELECT
c.market,
c.region,
ROUND(SUM(s.gross_price_total)/1000000,2) as gross_sales_mln
FROM gross_sales s JOIN dim_customer c
ON s.customer_code=c.customer_code
WHERE s.fiscal_year=2021
GROUP BY s.market ORDER BY gross_sales_mln DESC),
CTE1 AS (SELECT* , dense_rank()
over(partition by region order by gross_sales_mln DESC) as rnk
FROM CTE)
SELECT * FROM CTE1 WHERE rnk<=2;
```

| market | region | gross_sales_mln | rnk |
|---|---|---|---|
| India | APAC | 455.05 | 1 |
| South Korea | APAC | 131.86 | 2 |
| United Kingd... | EU | 78.11 | 1 |
| France | EU | 67.62 | 2 |
| Mexico | LATAM | 2.30 | 1 |
| Brazil | LATAM | 2.14 | 2 |
| USA | NA | 264.46 | 1 |
| Canada | NA | 89.78 | 2 |

Result Grid

Form Editor

Field Types

Query Stats

Execution Plan

esult 5

Read Only

utput

Action Output

| # | Time | Action | Message | Duration / Fetch |
|---|---|---|---|---|
| 1 | 14:27:22 | WITH CTE AS ( SELECT c.market, c.region, ROUND(SUM(s.gross_price_total)/1000000,2) as gross_... | 8 row(s) returned | 4.562 sec / 0.000 sec |

# WITHOUT VIEW

```sql
WITH CTE AS (
SELECT
c.market,
c.region,
ROUND(SUM(g.gross_price*s.sold_quantity)/1000000,2) as gross_sales_mln
FROM fact_sales_monthly s JOIN dim_customer c
ON s.customer_code=c.customer_code
JOIN fact_gross_price g
ON s.product_code=g.product_code AND g.fiscal_year= s.fiscal_year
WHERE s.fiscal_year=2021
GROUP BY market ORDER BY gross_sales_mln DESC),
CTE1 AS (SELECT* , dense_rank()
over(partition by region order by gross_sales_mln DESC) as rnk
FROM CTE)
SELECT * FROM CTE1 WHERE rnk<=2;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| market | region | gross_sales_mln | rnk |
|---|---|---|---|
| India | APAC | 455.05 | 1 |
| South Korea | APAC | 131.86 | 2 |
| United Kingd... | EU | 78.11 | 1 |
| France | EU | 67.62 | 2 |
| Mexico | LATAM | 2.30 | 1 |
| Brazil | LATAM | 2.14 | 2 |
| USA | NA | 264.46 | 1 |
| Canada | NA | 89.78 | 2 |

Result Grid | Form Editor | Field Types | Query Stats | Execution Plan

Result 28 | Read Only

Output

Action Output

| # | Time | Action | Message | Duration / Fetch |
|---|---|---|---|---|
| 1 | 14:36:02 | WITH CTE AS ( SELECT c.market, c.region, ROUND(SUM(g.gross_price*s.sold_quantity)/1000000,2)... | 8 row(s) returned | 3.359 sec / 0.000 sec |

# SUPPLY CHAIN ANALYTICS

## Forecast Accuracy for all customers for a given fiscal year

Attach | Add a child issue | Link issue

**Description**

**As a** product owner, **I need** an aggregate forecast accuracy report for all the customers for a given fiscal year **so that** I can track the accuracy of the forecast we make for these customers.

The report should have the following fields.

1. Customer Code, Name, Market
2. Total Sold Quantity
3. Total Forecast Quantity
4. Net Error
5. Absolute Error
6. Forecast Accuracy %

- Will create a helper table by joining these two tables to get sold_quantity and forecast_quantity in a single table.
  - fact_forecast_monthly
  - fact_sales_monthly

fact_forecast_monthly - total rows - 1885943
inner join - 1390838
difference = 4,95,105

fact_sales_monthly - total rows - 1425708
inner join - 1390838
additional rows - 34,870

**Will join both tables on these 3 things basically – date, product_code and customer_code**

```
1• SELECT * FROM gdb041.fact_forecast_monthly
2
```

| date | fiscal_year | product_code | customer_code | forecast_quantity |
|---|---|---|---|---|
| 2017-09-01 | 2018 | A0118150101 | 70002017 | 18 |
| 2017-09-01 | 2018 | A0118150101 | 70002018 | 11 |
| 2017-09-01 | 2018 | A0118150101 | 70003181 | 9 |

```
1• SELECT * FROM gdb0041.fact_sales_monthly;
```

| date | product_code | customer_code | sold_quantity | fiscal_year |
|---|---|---|---|---|
| 2017-09-01 | A0118150101 | 90027207 | 4 | 2018 |
| 2017-11-01 | A0118150101 | 90027207 | 16 | 2018 |
| 2017-12-01 | A0118150101 | 90027207 | 4 | 2018 |
| 2018-01-01 | A0118150101 | 90027207 | 6 | 2018 |
| 2018-03-01 | A0118150101 | 90027207 | 9 | 2018 |

But here rows are different in both tables

Continued..

**MERGING TWO TABLES and created a single table__fact_act_est**

Will define their data types for each column – **fiscal_year –**

**calculated column- year((‘date’ +interval 4 month))**

```
CREATE TABLE fact_act_est
(
    SELECT
        s.date ,
        s.fiscal_year,
        s.customer_code,
        s.product_code,
        s.sold_quantity,
        f.forecast_quantity
    FROM
        fact_sales_monthly s
    LEFT JOIN fact_forecast_monthly f
    USING(date,product_code,customer_code)
```

| Column Name | Datatype | PK | NN | UQ | B | UN | ZF | AI | G | Default/Expression |
|---|---|---|---|---|---|---|---|---|---|---|
| date | DATE | ☑ | ☑ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | |
| fiscal_year | YEAR | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☑ | year((`date` + interval 4 month)) |
| product_code | VARCHAR(45) | ☑ | ☑ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | '' |
| customer_code | INT | ☑ | ☑ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | '0' |
| sold_quantity | INT | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | NULL |
| forecast_quantity | INT | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | NULL |
| | | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | |

```
    UNION
    SELECT
        f.date ,
        f.fiscal_year,
        f.customer_code,
        f.product_code,
        s.sold_quantity,
        f.forecast_quantity
    FROM
        fact_sales_monthly s
    RIGHT JOIN fact_forecast_monthly f
    USING(date,product_code,customer_code)
);
```

**NOW UPDATING THOSE RECORDS WHERE SOLD_QUANTITY IS NULL AND**

```
SELECT * FROM gdb041.fact_act_est;

update fact_act_est
set sold_quantity = 0
where sold_quantity is null;
```

**FORECAST QUANTITY IS NULL**

```
UPDATE fact_act_est
set forecast_quantity = 0
WHERE forecast_quantity IS NULL;
```

**NOW we have our table created. We will create our report of forecast accuracy .**

# Database Triggers

- We have created a table – fact_act_est by joining two tables fact_forecast_monthly and fact_sales_monthly .
- What happens if any new record gets inserted in our fact_sales_monthly or fact_forecast_monthly. Our fact_act_est is kind of like a derived table from fact_sales_monthly and fact_forecast _monthly.
- If there is a new record that gets inserted into fact_sales_monthly we want to automatically insert corresponding record in actual estimated table as well (fact_act_est) and triggers will allow you to do this.
- Trigger is an SQL code that gets executed at a certain event. Event could be insertion of a record in fact_sales_monthly .

Will create a TRIGGER in fact_sales_monthly __ How to steps given below _

1. Click on second icon of fact_sales_monthly -> Click on Triggers below


- And creating a trigger which will get executed after any record insertion, click on + beside AFTER INSERT here

1. We will write an SQL code and will use insert into fact_act_est table and supply the values using VALUES () function , now what records would be inserted in this table . NEW RECORDS would be inserted ____ we will use a NEW operator in TRIGGERS . New will contain that record which will be inserted in fact_sales_monthly or fact_forecast_monthly

[redacted]

```
NEW.date,
NEW.prouct_code,
NEW.customer_code,
```

2. Now this is like a **KEY** __ date , product_code and customer_code.

3. Now what if for this key we already have a record in fact_act_est so that insert doesn't work because it will take new records only .that's why we will specify ( on duplicate key update ) – SYNTAX , where if there is a record which is represented by this KEY present in fact_act_est already. Then just update sold_quantity

_ What is **values(sold_quantity)** ?

- **Whatever is the value corresponding to sold_quantity column mentioned in insert statement which is equal to NEW.sold_quantity..**
- **Can also say sold_quantity= NEW.sold_quantity but that is better SYNTAX (sold_quantity= values(sold_quantity))**

```
CREATE DEFINER=`root`@`localhost` TRIGGER `fact_sales_monthly_
    insert into fact_act_est
        (date,  product_code, customer_code, sold_quantity)
    values (
        NEW.date,
        NEW.product_code,
        NEW.customer_code,
        NEW.sold_quantity
    )
    on duplicate key update
        sold_quantity = values(sold_quantity);
END
```

DATABASE TRIGGER CONTINUES…

_ Once it will updated we will check triggers

```
SELECT * FROM gdb041.fact_act_est;

show triggers
```

| Trigger | Event | Table | Statement | Timing | Created | sql |
|---|---|---|---|---|---|---|
| fact_sales_monthly_AFTE... | INSERT | fact_sales_monthly | BEGIN insert into fact_act... | AFTER | 2022-10-23 18:26:19.42 | STF |

**Same for fact_forecast_monthly table __**

```
CREATE DEFINER = CURRENT_USER TRIGGER `gdb041`.`fact_forecast_monthl
BEGIN
    insert into fact_act_est
        (date,  product_code, customer_code, forecast_quantity)
    values (
        NEW.date,
        NEW.product_code,
        NEW.customer_code,
        NEW.forecast_quantity
    )
    on duplicate key update
        forecast_quantity = values(forecast_quantity);
END
```

09:03 / 12:10

Triggers

# Temporary Tables & Forecast Accuracy Report

```
WITH forecast_err_table as
(
SELECT
    s.customer_code,
    sum(s.sold_quantity) as total_sold_qty,
    sum(s.forecast_quantity) as total_forecast_qty,
    sum((forecast_quantity-sold_quantity)) as net_err,
    sum((forecast_quantity-sold_quantity))*100/sum(forecast_quantity)
    as net_err_pct,
    sum(abs(forecast_quantity-sold_quantity)) as abs_net_error,
    sum(abs(forecast_quantity-sold_quantity))*100/sum(forecast_quantity)
    as abs_net_err_pct
FROM fact_act_est s
WHERE s.fiscal_year=2021
group by customer_code)
```

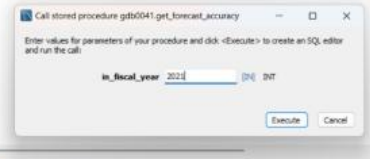
```
SELECT
    e.*,
    c.customer,
    c.market,
if(abs_net_error_pct>100,0,100-abs_net_error_pct) as forecast_accuracy
    FROM forecast_err_table e JOIN dim_customer c USING(customer_code)
ORDER BY forecast_accuracy DESC;
```

**STORED PROCEDURE _____ OF FORECAST ACCURACY REPORT FOR ANY FISCAL YEAR.**

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `get_forecast_accuracy`(
                in_fiscal_year INT
)
BEGIN
WITH forecast_err_table as (
    SELECT
    s.customer_code,
    sum(s.sold_quantity) as total_sold_qty,
    sum(s.forecast_quantity) as total_forecast_qty,
    sum((forecast_quantity-sold_quantity)) as net_err,
    sum((forecast_quantity-sold_quantity))*100/sum(forecast_quantity) a
    sum(abs(forecast_quantity-sold_quantity)) as abs_net_error,
    sum(abs(forecast_quantity-sold_quantity))*100/sum(forecast_quantity
FROM fact_act_est s
    WHERE s.fiscal_year=in_fiscal_year
    group by customer_code)
```

```
        SELECT
        e.*,
        c.customer,
        c.market,
        if(abs_net_error_pct>100,0,100-abs_net_error_pct) as forecast_a
            FROM forecast_err_table e JOIN dim_customer c USING(custome
        ORDER BY forecast_accuracy DESC;
    END
```

```
call gdb0041.get_forecast_accuracy(2021);
```




| customer_code | total_sold_qty | total_forecast_qty | net_err | net_err_pct | abs_net_error |
|---|---|---|---|---|---|
| 90013120 | 109547 | 133532 | 23985 | 17.9620 | 70467 |
| 70010048 | 119439 | 142010 | 22571 | 15.8940 | 75711 |
| 90023027 | 236189 | 279962 | 43773 | 15.6353 | 149303 |
| 90023026 | 228988 | 273492 | 44504 | 16.2725 | 146948 |
| 90017051 | 86823 | 118067 | 31244 | 26.4629 | 63568 |
| 90017058 | 86860 | 110195 | 23335 | 21.1761 | 59473 |
| 90023028 | 239081 | 283323 | 44242 | 15.6154 | 153058 |
| 90023024 | 246397 | 287233 | 40836 | 14.2170 | 155610 |
| 90013124 | 110898 | 136116 | 25218 | 18.5268 | 73826 |

| abs_net_error_pct | customer | market | forecast_accuracy |
|---|---|---|---|
| 52.7716 | Coolblue | Italy | 47.2284 |
| 53.3139 | Atliq e Store | Bangla... | 46.6861 |
| 53.3297 | Costco | Canada | 46.6703 |
| 53.7303 | Relief | Canada | 46.2697 |
| 53.8406 | Forward Stores | Portugal | 46.1594 |
| 53.9707 | Mbit | Portugal | 46.0293 |
| 54.0224 | walmart | Canada | 45.9776 |
| 54.1755 | Sage | Canada | 45.8245 |
| 54.2376 | Amazon | Italy | 45.7624 |

**WHAT IF WE DON'T WANT TO USE CTE SO WE WILL USE TEMPORARY TABLE , It will create a memory somewhere of this table THAT WILL BE VALID TILL THAT PARTICULAR CURRENT SESSION _____We have created a temporary table- forecast_err_table**

```sql
CREATE TEMPORARY TABLE forecast_err_table
SELECT
    s.customer_code,
    sum(s.sold_quantity) as total_sold_qty,
    sum(s.forecast_quantity) as total_forecast_qty,
    sum((forecast_quantity-sold_quantity)) as net_err,
    sum((forecast_quantity-sold_quantity))*100/sum(forecast_quantity)
    as net_err_pct,
    sum(abs(forecast_quantity-sold_quantity)) as abs_net_error,
    sum(abs(forecast_quantity-sold_quantity))*100/sum(forecast_quantity)
    as abs_net_err_pct
FROM fact_act_est s
WHERE s.fiscal_year=2021
group by customer_code;
```

```sql
SELECT
    e.*,
    c.customer,
    c.market,
if(abs_net_err_pct>100,0,100-abs_net_err_pct) as forecast_accuracy
    FROM forecast_err_table e JOIN dim_customer c USING(customer_code)
ORDER BY forecast_accuracy DESC;
```

| customer_code | total_sold_qty | total_forecast_qty | net_err | net_err_pct | abs_net_error | abs_net_ |
|---|---|---|---|---|---|---|
| 90013120 | 109547 | 133532 | 23985 | 17.9620 | 70467 | 52.7716 |
| 70010048 | 119439 | 142010 | 22571 | 15.8940 | 75711 | 53.3139 |
| 90023027 | 236189 | 279962 | 43773 | 15.6353 | 149303 | 53.3297 |
| 90023026 | 228988 | 273492 | 44504 | 16.2725 | 146948 | 53.7303 |
| 90017051 | 86823 | 118067 | 31244 | 26.4629 | 63568 | 53.8406 |
| 90017058 | 86860 | 110195 | 23335 | 21.1761 | 59473 | 53.9707 |
| 90023028 | 239081 | 283323 | 44242 | 15.6154 | 153058 | 54.0224 |
| 90023024 | 246397 | 287233 | 40836 | 14.2170 | 155610 | 54.1755 |
| 90013124 | 110898 | 136116 | 25218 | 18.5268 | 73826 | 54.2376 |

DIFFERENCE BETWEEN CTE AND TEMPORARY TABLE__ CTE IS JUST VALID TILL THE SCOPE OF THAT STATEMENT , BUT TEMPORARY TABLE WILL BE VALID TILL THAT PARTICULAR ENTIRE SESSION. WE CAN OPEN A NEW TAB AND RUN A QUERY ON TEMPORARY TABLE BUT NOT IN CASE OF CTE.

# THANK YOU.