SQL Views

A virtual table but not store the data physically

3 level architecture –

Physical level – actual data is stored in the database, access to DBA.

Logical level – deal with how to organise the data, access to data engineer.

View level – multiple views for different positions like BA, DA, end users.

Syntax –

CREATE VIEW VIEW\_NAME AS

(

SELECT….

FROM….

WHERE….

)

Use cases Example –

WITH CTE\_Monthly AS

(

SELECT DATETRUNC(MONTH, created\_date) OrderMonth,

SUM(product\_id) TotalSales,

COUNT(userid) TotalOrders

FROM sales

GROUP BY DATETRUNC(MONTH, created\_date)

)

SELECT OrderMonth, SUM(TotalSales) OVER(ORDER BY OrderMonth) AS RunningTotal

FROM CTE\_Monthly

CREATE VIEW VMonthlySumm AS

(

SELECT DATETRUNC(MONTH, created\_date) OrderMonth,

SUM(product\_id) TotalSales,

COUNT(userid) TotalOrders

FROM sales

GROUP BY DATETRUNC(MONTH, created\_date)

)

SELECT \* FROM VMonthlySumm

SELECT OrderMonth, SUM(TotalSales) OVER(ORDER BY OrderMonth) AS RunningTotal

FROM VMonthlySumm

DROP VIEW VMonthlySumm

IF OBJECT\_ID('VMonthlySumm', 'V') IS NOT NULL

DROP VIEW VMonthlySumm;

GO

CREATE VIEW VMonthlySumm AS

(

SELECT DATETRUNC(MONTH, created\_date) OrderMonth,

SUM(product\_id) TotalSales,

COUNT(userid) TotalOrders

FROM sales

GROUP BY DATETRUNC(MONTH, created\_date)

)

CREATE VIEW V\_OrderDetails AS

(

SELECT s.userid, s.created\_date, p.product\_id, p.product\_name, p.price, u.userid, g.gold\_signup\_date

FROM sales s

LEFT JOIN product p

ON s.product\_id = p.product\_id

LEFT JOIN users u

ON s.userid = u.userid

LEFT JOIN goldusers\_signup g

ON u.userid = g.userid

)

Data Security as restricting data with the country zone by filtering it.

Flexibility if created Views

Use cases –

Store central complex business logic to be reused

Hide complexity by offering friendly views to users

Data Security by hiding sensitive rows & columns

Flexibity & dynamic

Offer your object in multiple languages

Virtual layers(Data Mart) & warehouses