Prepared by Asif Bhat

SQL Tutorials - Part 1

In [1]:
```
import sqlalchemy
import numpy
import pandas as pd
import warnings
warnings.filterwarnings('ignore')
```

In [97]:
```
engine_mysql = sqlalchemy.create_engine('mysql+pymysql://root:root@localhos
```

In [98]:
```
print("connecting with engine " + str(engine_mysql))
```

```
connecting with engine Engine(mysql+pymysql://root:***@localhost:3306/em
p)
```

In [99]:
```
print(engine_mysql.table_names())
```

```
['employees']
```

In [101]:
```
con_mysql = engine_mysql.connect()
```

In [102]:
```
# Simple SELECT Query
query = '''
SELECT * FROM Employees
'''
df_mysql = pd.read_sql_query(query, con_mysql)
df_mysql
```

Out[102]:

	Id	Name	Gender	Salary
0	1	Mark	Male	5000
1	2	John	Male	4500
2	3	Pam	Female	5500
3	4	Sara	Female	4000
4	5	Todd	Male	3500
5	6	Mary	Female	5000
6	7	Ben	Male	6500
7	8	Jodi	Female	7000
8	9	Tom	Male	5500
9	10	Ron	Male	5000

In [46]:
```
# Group By
query = '''
SELECT Gender,
      COUNT(*) AS GenderTotal,
      AVG(Salary) AS AvgSal,
      MIN(Salary) AS MinSal,
      MAX(Salary) AS MaxSal
FROM Employees
GROUP BY Gender'''
df_mysql = pd.read_sql_query(query, con_mysql)
df_mysql
```

Out[46]:

	Gender	GenderTotal	AvgSal	MinSal	MaxSal
0	Male	6	5000.0	3500	6500
1	Female	4	5375.0	4000	7000

In [47]:
```
# Display Non-aggregated values (like employee Name and Salary) in result s
query = '''
SELECT Name,
      Salary,
      Employees.Gender,
      Genders.GenderTotals,
      Genders.AvgSal,
      Genders.MinSal,
      Genders.MaxSal
FROM Employees
INNER JOIN
(SELECT Gender,
        COUNT(*) AS GenderTotals,
        AVG(Salary) AS AvgSal,
        MIN(Salary) AS MinSal,
        MAX(Salary) AS MaxSal
FROM Employees
GROUP BY Gender) AS Genders
ON Genders.Gender = Employees.Gender;
'''
pd.read_sql_query(query, con_mysql)
```

Out[47]:

	Name	Salary	Gender	GenderTotals	AvgSal	MinSal	MaxSal
0	Mark	5000	Male	6	5000.0	3500	6500
1	John	4500	Male	6	5000.0	3500	6500
2	Pam	5500	Female	4	5375.0	4000	7000
3	Sara	4000	Female	4	5375.0	4000	7000
4	Todd	3500	Male	6	5000.0	3500	6500
5	Mary	5000	Female	4	5375.0	4000	7000
6	Ben	6500	Male	6	5000.0	3500	6500
7	Jodi	7000	Female	4	5375.0	4000	7000
8	Tom	5500	Male	6	5000.0	3500	6500
9	Ron	5000	Male	6	5000.0	3500	6500

In [48]:
```
'''Display Non-aggregated values (like employee Name and Salary)
in result set along with aggregated values using PARTITION'''

query = '''
select Name,
       Salary,
       Gender,
       count(Gender) over (partition by gender) as gendertotal,
       avg(Salary) over (partition by gender) as avg_salary,
       sum(salary) over (partition by gender) as total_salary,
       min(salary) over (partition by gender) as min_salary,
       max(salary) over (partition by gender) as max_salary,
       rank() over (partition by gender order by salary desc) as rank_sal,
       dense_rank() over (partition by gender order by salary desc) as dens
       row_number() over (partition by gender order by salary desc) as row_
from emp.employees
'''
pd.read_sql_query(query, con_mysql)
```

Out[48]:

	Name	Salary	Gender	gendertotal	avg_salary	total_salary	min_salary	max_salary	rank_sal	de
0	Jodi	7000	Female	4	5375.0	21500.0	4000	7000	1	
1	Pam	5500	Female	4	5375.0	21500.0	4000	7000	2	
2	Mary	5000	Female	4	5375.0	21500.0	4000	7000	3	
3	Sara	4000	Female	4	5375.0	21500.0	4000	7000	4	
4	Ben	6500	Male	6	5000.0	30000.0	3500	6500	1	
5	Tom	5500	Male	6	5000.0	30000.0	3500	6500	2	
6	Mark	5000	Male	6	5000.0	30000.0	3500	6500	3	
7	Ron	5000	Male	6	5000.0	30000.0	3500	6500	3	
8	John	4500	Male	6	5000.0	30000.0	3500	6500	5	
9	Todd	3500	Male	6	5000.0	30000.0	3500	6500	6	

In [49]:
```
# Running Total
query = '''
SELECT Name, Gender, Salary,
    SUM(Salary) OVER (ORDER BY ID) AS RunningTotal
FROM Employees
'''
pd.read_sql_query(query, con_mysql)
```

Out[49]:

	Name	Gender	Salary	RunningTotal
0	Mark	Male	5000	5000.0
1	John	Male	4500	9500.0
2	Pam	Female	5500	15000.0
3	Sara	Female	4000	19000.0
4	Todd	Male	3500	22500.0
5	Mary	Female	5000	27500.0
6	Ben	Male	6500	34000.0
7	Jodi	Female	7000	41000.0
8	Tom	Male	5500	46500.0
9	Ron	Male	5000	51500.0

In [51]:
```
# Running Total : Partition by Gender
query = '''
SELECT Name, Gender, Salary,
    SUM(Salary) OVER (partition by gender ORDER BY ID) AS RunningTotal
FROM Employees
'''
pd.read_sql_query(query, con_mysql)
```

Out[51]:

	Name	Gender	Salary	RunningTotal
0	Pam	Female	5500	5500.0
1	Sara	Female	4000	9500.0
2	Mary	Female	5000	14500.0
3	Jodi	Female	7000	21500.0
4	Mark	Male	5000	5000.0
5	John	Male	4500	9500.0
6	Todd	Male	3500	13000.0
7	Ben	Male	6500	19500.0
8	Tom	Male	5500	25000.0
9	Ron	Male	5000	30000.0

In [67]:
```
# Running Average
query = '''
SELECT Name, Gender, Salary,
    AVG(Salary) OVER (ORDER BY ID) AS RunningAVG
FROM Employees
'''
pd.read_sql_query(query, con_mysql)
```

Out[67]:

	Name	Gender	Salary	RunningAVG
0	Mark	Male	5000	5000.0000
1	John	Male	4500	4750.0000
2	Pam	Female	5500	5000.0000
3	Sara	Female	4000	4750.0000
4	Todd	Male	3500	4500.0000
5	Mary	Female	5000	4583.3333
6	Ben	Male	6500	4857.1429
7	Jodi	Female	7000	5125.0000
8	Tom	Male	5500	5166.6667
9	Ron	Male	5000	5150.0000

In [70]:
```
# Running Count , Sum & Average
query = '''
SELECT Name, Gender, Salary,
    AVG(Salary) OVER (ORDER BY ID) AS Avg_Sal,
    Sum(Salary) OVER (ORDER BY ID) AS Sum_Sal,
    count(Salary) OVER (ORDER BY ID) AS Count_Sal
FROM Employees
'''
pd.read_sql_query(query, con_mysql)
```

Out[70]:

	Name	Gender	Salary	Avg_Sal	Sum_Sal	Count_Sal
0	Mark	Male	5000	5000.0000	5000.0	1
1	John	Male	4500	4750.0000	9500.0	2
2	Pam	Female	5500	5000.0000	15000.0	3
3	Sara	Female	4000	4750.0000	19000.0	4
4	Todd	Male	3500	4500.0000	22500.0	5
5	Mary	Female	5000	4583.3333	27500.0	6
6	Ben	Male	6500	4857.1429	34000.0	7
7	Jodi	Female	7000	5125.0000	41000.0	8
8	Tom	Male	5500	5166.6667	46500.0	9
9	Ron	Male	5000	5150.0000	51500.0	10

In [72]:
```
# Running Count , Sum & Average
query = '''
SELECT Name, Gender, Salary,
    AVG(Salary) OVER (ORDER BY ID Range between UNBOUNDED PRECEDING AND CUR
    Sum(Salary) OVER (ORDER BY ID Range between UNBOUNDED PRECEDING AND CUR
    count(Salary) OVER (ORDER BY ID Range between UNBOUNDED PRECEDING AND C
FROM Employees
'''
pd.read_sql_query(query, con_mysql)
```

Out[72]:

	Name	Gender	Salary	Avg_Sal	Sum_Sal	Count_Sal
0	Mark	Male	5000	5000.0000	5000.0	1
1	John	Male	4500	4750.0000	9500.0	2
2	Pam	Female	5500	5000.0000	15000.0	3
3	Sara	Female	4000	4750.0000	19000.0	4
4	Todd	Male	3500	4500.0000	22500.0	5
5	Mary	Female	5000	4583.3333	27500.0	6
6	Ben	Male	6500	4857.1429	34000.0	7
7	Jodi	Female	7000	5125.0000	41000.0	8
8	Tom	Male	5500	5166.6667	46500.0	9
9	Ron	Male	5000	5150.0000	51500.0	10

In [71]:
```
# Sum , Average & Count
# UNBOUNDED PRECEDING : Firt row of Dataset
# UNBOUNDED FOLLOWING : Last row of Dataset
query = '''
SELECT Name, Gender, Salary,
    AVG(Salary) OVER (ORDER BY ID Range between UNBOUNDED PRECEDING AND UNB
    Sum(Salary) OVER (ORDER BY ID Range between UNBOUNDED PRECEDING AND UNB
    count(Salary) OVER (ORDER BY ID Range between UNBOUNDED PRECEDING AND U
FROM Employees
'''
pd.read_sql_query(query, con_mysql)
```

Out[71]:

	Name	Gender	Salary	Avg_Sal	Sum_Sal	Count_Sal
0	Mark	Male	5000	5150.0	51500.0	10
1	John	Male	4500	5150.0	51500.0	10
2	Pam	Female	5500	5150.0	51500.0	10
3	Sara	Female	4000	5150.0	51500.0	10
4	Todd	Male	3500	5150.0	51500.0	10
5	Mary	Female	5000	5150.0	51500.0	10
6	Ben	Male	6500	5150.0	51500.0	10
7	Jodi	Female	7000	5150.0	51500.0	10
8	Tom	Male	5500	5150.0	51500.0	10
9	Ron	Male	5000	5150.0	51500.0	10

Difference between Range & Row

In [78]:
```
engine_mysql = sqlalchemy.create_engine('mysql+pymysql://root:root@localhos
print("connecting with engine " + str(engine_mysql))
con_mysql = engine_mysql.connect()
```

```
connecting with engine Engine(mysql+pymysql://root:***@localhost:3306/em
p)
```

In [79]:
```
query = '''
SELECT * FROM Employees
'''
pd.read_sql_query(query, con_mysql)
```

Out[79]:

	Id	Name	Gender	Salary
0	1	Mark	Male	5000
1	2	John	Male	1000
2	3	Pam	Female	5500
3	4	Sara	Female	3000
4	5	Todd	Male	3000
5	6	Mary	Female	5000
6	7	Ben	Male	7000
7	8	Jodi	Female	7000
8	9	Tom	Male	5500
9	10	Ron	Male	5000

In [80]:
```
# Running Total with ROWS()
# ROWS & RANGE only differ when there are duplicates in your data
query = '''
SELECT Name, Salary,
    SUM(Salary) OVER(ORDER BY Salary
    ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS RunningTotal
FROM Employees
'''
pd.read_sql_query(query, con_mysql)
```

Out[80]:

	Name	Salary	RunningTotal
0	John	1000	1000.0
1	Sara	3000	4000.0
2	Todd	3000	7000.0
3	Mark	5000	12000.0
4	Mary	5000	17000.0
5	Ron	5000	22000.0
6	Pam	5500	27500.0
7	Tom	5500	33000.0
8	Ben	7000	40000.0
9	Jodi	7000	47000.0

In [81]:
```
# Running Total with RANGE()
query = '''
SELECT Name, Salary,
    SUM(Salary) OVER(ORDER BY Salary RANGE BETWEEN UNBOUNDED PRECEDING AND
FROM Employees
'''
pd.read_sql_query(query, con_mysql)
```

Out[81]:

	Name	Salary	RunningTotal
0	John	1000	1000.0
1	Sara	3000	7000.0
2	Todd	3000	7000.0
3	Mark	5000	22000.0
4	Mary	5000	22000.0
5	Ron	5000	22000.0
6	Pam	5500	33000.0
7	Tom	5500	33000.0
8	Ben	7000	47000.0
9	Jodi	7000	47000.0

NTILE

In [83]:
```
engine_mysql = sqlalchemy.create_engine('mysql+pymysql://root:root@localhos
print("connecting with engine " + str(engine_mysql))
con_mysql = engine_mysql.connect()
```

```
connecting with engine Engine(mysql+pymysql://root:***@localhost:3306/em
p)
```

In [84]:
```
query = '''
SELECT * FROM Employees
'''
pd.read_sql_query(query, con_mysql)
```

Out[84]:

	Id	Name	Gender	Salary
0	1	Mark	Male	5000
1	2	John	Male	4500
2	3	Pam	Female	5500
3	4	Sara	Female	4000
4	5	Todd	Male	3500
5	6	Mary	Female	5000
6	7	Ben	Male	6500
7	8	Jodi	Female	7000
8	9	Tom	Male	5500
9	10	Ron	Male	5000

In [85]:
```
# NTILE - Divide dataset into three groups based on ORDER BY Column
query = '''
SELECT Name,
       Gender,
       Salary,
       NTILE(3) OVER (ORDER BY Salary) as groupping
FROM Employees
'''
pd.read_sql_query(query, con_mysql)
```

Out[85]:

	Name	Gender	Salary	groupping
0	Todd	Male	3500	1
1	Sara	Female	4000	1
2	John	Male	4500	1
3	Mark	Male	5000	1
4	Mary	Female	5000	2
5	Ron	Male	5000	2
6	Pam	Female	5500	2
7	Tom	Male	5500	3
8	Ben	Male	6500	3
9	Jodi	Female	7000	3

In [86]:
```
#LEAD & LAG by offset 1 or by one row
query = '''
SELECT Name, Gender, Salary,
    LEAD(Salary) OVER (ORDER BY Salary) AS Lead_Sal,
    LAG(Salary) OVER (ORDER BY Salary) AS Lag_Sal
FROM Employees
'''
pd.read_sql_query(query, con_mysql)
```

Out[86]:

	Name	Gender	Salary	Lead_Sal	Lag_Sal
0	Todd	Male	3500	4000.0	NaN
1	Sara	Female	4000	4500.0	3500.0
2	John	Male	4500	5000.0	4000.0
3	Mark	Male	5000	5000.0	4500.0
4	Mary	Female	5000	5000.0	5000.0
5	Ron	Male	5000	5500.0	5000.0
6	Pam	Female	5500	5500.0	5000.0
7	Tom	Male	5500	6500.0	5500.0
8	Ben	Male	6500	7000.0	5500.0
9	Jodi	Female	7000	NaN	6500.0

In [87]:
```
#LEAD  & lAG by 2 row
query = '''
SELECT Name, Gender, Salary,
    LEAD(Salary, 2, -1) OVER (ORDER BY Salary) AS Lead_2,
    LAG(Salary, 2, -1) OVER (ORDER BY Salary) AS Lag_1
FROM Employees
'''
pd.read_sql_query(query, con_mysql)
```

Out[87]:

	Name	Gender	Salary	Lead_2	Lag_1
0	Todd	Male	3500	4500	-1
1	Sara	Female	4000	5000	-1
2	John	Male	4500	5000	3500
3	Mark	Male	5000	5000	4000
4	Mary	Female	5000	5500	4500
5	Ron	Male	5000	5500	5000
6	Pam	Female	5500	6500	5000
7	Tom	Male	5500	7000	5000
8	Ben	Male	6500	-1	5500
9	Jodi	Female	7000	-1	5500

In [88]:
```
#LEAD/lAG by 1 row with partition by gender
query = '''
SELECT Name, Gender, Salary,
    LEAD(Salary, 1, -1) OVER (partition by Gender ORDER BY Salary) AS Lead_
    LAG(Salary, 1, -1) OVER (partition by Gender ORDER BY Salary) AS Lag_1
FROM Employees
'''
pd.read_sql_query(query, con_mysql)
```

Out[88]:

	Name	Gender	Salary	Lead_2	Lag_1
0	Sara	Female	4000	5000	-1
1	Mary	Female	5000	5500	4000
2	Pam	Female	5500	7000	5000
3	Jodi	Female	7000	-1	5500
4	Todd	Male	3500	4500	-1
5	John	Male	4500	5000	3500
6	Mark	Male	5000	5000	4500
7	Ron	Male	5000	5500	5000
8	Tom	Male	5500	6500	5000
9	Ben	Male	6500	-1	5500

In [89]:
```
#LEAD/lAG by 1 row with partition by gender
query = '''
SELECT Name, Gender, Salary,
    LEAD(Salary, 1, -1) OVER (partition by Gender ORDER BY Salary) AS Lead_
    LAG(Salary, 1, -1) OVER (partition by Gender ORDER BY Salary) AS Lag_1
FROM Employees
'''
pd.read_sql_query(query, con_mysql)
```

Out[89]:

	Name	Gender	Salary	Lead_2	Lag_1
0	Sara	Female	4000	5000	-1
1	Mary	Female	5000	5500	4000
2	Pam	Female	5500	7000	5000
3	Jodi	Female	7000	-1	5500
4	Todd	Male	3500	4500	-1
5	John	Male	4500	5000	3500
6	Mark	Male	5000	5000	4500
7	Ron	Male	5000	5500	5000
8	Tom	Male	5500	6500	5000
9	Ben	Male	6500	-1	5500

In [90]:
```
#First Value function
query = '''
SELECT Name, Gender, Salary,
FIRST_VALUE(Name) OVER (ORDER BY Salary) AS FirstValue
FROM Employees
'''
pd.read_sql_query(query, con_mysql)
```

Out[90]:

	Name	Gender	Salary	FirstValue
0	Todd	Male	3500	Todd
1	Sara	Female	4000	Todd
2	John	Male	4500	Todd
3	Mark	Male	5000	Todd
4	Mary	Female	5000	Todd
5	Ron	Male	5000	Todd
6	Pam	Female	5500	Todd
7	Tom	Male	5500	Todd
8	Ben	Male	6500	Todd
9	Jodi	Female	7000	Todd

In [91]:
```
#First Value function with Partition on gender
query = '''
SELECT Name, Gender, Salary,
FIRST_VALUE(Name) OVER (partition by gender ORDER BY Salary) AS FirstValue
FROM Employees
'''
pd.read_sql_query(query, con_mysql)
```

Out[91]:

	Name	Gender	Salary	FirstValue
0	Sara	Female	4000	Sara
1	Mary	Female	5000	Sara
2	Pam	Female	5500	Sara
3	Jodi	Female	7000	Sara
4	Todd	Male	3500	Todd
5	John	Male	4500	Todd
6	Mark	Male	5000	Todd
7	Ron	Male	5000	Todd
8	Tom	Male	5500	Todd
9	Ben	Male	6500	Todd

In [95]:
```
#Last Value function
query = '''
SELECT Name, Gender, Salary,
LAST_VALUE(Name) OVER (ORDER BY Salary ROWS BETWEEN UNBOUNDED PRECEDING AND
FROM Employees
'''
pd.read_sql_query(query, con_mysql)
```

Out[95]:

	Name	Gender	Salary	LastValue
0	Todd	Male	3500	Jodi
1	Sara	Female	4000	Jodi
2	John	Male	4500	Jodi
3	Mark	Male	5000	Jodi
4	Mary	Female	5000	Jodi
5	Ron	Male	5000	Jodi
6	Pam	Female	5500	Jodi
7	Tom	Male	5500	Jodi
8	Ben	Male	6500	Jodi
9	Jodi	Female	7000	Jodi

In [96]:
```
#Last Value function with Partition on gender
query = '''
SELECT Name, Gender, Salary,
LAST_VALUE(Name) OVER
(partition by gender ORDER BY Salary ROWS BETWEEN UNBOUNDED PRECEDING AND U
FROM Employees
'''
pd.read_sql_query(query, con_mysql)
```

Out[96]:

	Name	Gender	Salary	FirstValue
0	Sara	Female	4000	Jodi
1	Mary	Female	5000	Jodi
2	Pam	Female	5500	Jodi
3	Jodi	Female	7000	Jodi
4	Todd	Male	3500	Ben
5	John	Male	4500	Ben
6	Mark	Male	5000	Ben
7	Ron	Male	5000	Ben
8	Tom	Male	5500	Ben
9	Ben	Male	6500	Ben

In [105]:
```
# Connect to market_star_schemas database
engine_mysql = sqlalchemy.create_engine('mysql+pymysql://root:root@localhos
con_mysql = engine_mysql.connect()
```

In [106]:
```
# Rank function
query='''
select Cust_id,
       round(sales,2) as sales,
       rank() over (order by sales desc)
from market_star_schema.market_fact_full
'''
pd.read_sql_query(query,con_mysql)
```

Out[106]:

	Cust_id	sales	rank() over (order by sales desc)
0	Cust_1818	4701.69	1
1	Cust_1818	4233.15	2
2	Cust_1641	4072.01	3
3	Cust_1641	3410.16	4
4	Cust_839	3364.25	5
5	Cust_1818	2337.89	6
6	Cust_839	1410.93	7
7	Cust_708	465.90	8
8	Cust_1088	305.05	9
9	Cust_1818	164.02	10
10	Cust_1641	162.00	11
11	Cust_1818	136.81	12
12	Cust_1641	57.22	13
13	Cust_1818	42.27	14
14	Cust_26	14.76	15

In [28]:
```
# Rank
query = '''
select c.Customer_Name as 'Customer Name',
       round(sum(m.Sales)) as 'Total Sales',
       rank() over (order by sum(m.Sales) desc ) as 'Sales Rank'
from  market_star_schemas.market_fact_full as m
inner join
market_star_schemas.cust_dimen as c
on (c.Cust_id=m.Cust_id)
group by c.Customer_Name
'''
pd.read_sql_query(query,con_mysql)
```

Out[28]:

	Customer Name	Total Sales	Sales Rank
0	EMILY PHAN	117124.0	1
1	DEBORAH BRUMFIELD	97433.0	2
2	ROY SKARIA	92542.0	3
3	SYLVIA FOULSTON	88876.0	4
4	GRANT CARROLL	88417.0	5
...	...	...	...
790	DOROTHY DICKINSON	198.0	791
791	KATRINA EDELMAN	181.0	792
792	NICOLE FJELD	153.0	793
793	NATALIE DECHERNEY	126.0	794
794	JEREMY FARRY	86.0	795

795 rows × 3 columns

In [29]:
```
# Rank with CTE Expression
query='''
with sales_rank as
(
select c.Customer_Name as 'Customer Name',
       round(sum(m.Sales)) as 'Total Sales',
       rank() over (order by sum(m.Sales) desc ) as 'SalesRank'
from  market_star_schemas.market_fact_full as m
inner join
market_star_schemas.cust_dimen as c
on (c.Cust_id=m.Cust_id)
group by c.Customer_Name
)
select * from sales_rank
where  sales_rank.SalesRank < 11
'''
pd.read_sql_query(query,con_mysql)
```

Out[29]:

	Customer Name	Total Sales	SalesRank
0	EMILY PHAN	117124.0	1
1	DEBORAH BRUMFIELD	97433.0	2
2	ROY SKARIA	92542.0	3
3	SYLVIA FOULSTON	88876.0	4
4	GRANT CARROLL	88417.0	5
5	ALEJANDRO GROVE	83562.0	6
6	DARREN BUDD	81577.0	7
7	JULIA BARNETT	80044.0	8
8	JOHN LUCAS	79696.0	9
9	LIZ MACKENDRICK	76306.0	10

In [31]:
```
# Rank & dense rank
query='''
select m.Ord_id,
       m.Discount,
       c.Customer_Name,
       rank() over (order by discount desc) as discount_rank,
       dense_rank() over (order by discount desc) as discount_dense_rank
from market_star_schemas.market_fact_full as m
inner join
market_star_schemas.cust_dimen as c
on (c.Cust_id=m.Cust_id)
where c.Customer_Name= 'Rick Wilson';
'''
pd.read_sql_query(query,con_mysql)
```

Out[31]:

	Ord_id	Discount	Customer_Name	discount_rank	discount_dense_rank
0	Ord_3855	0.10	RICK WILSON	1	1
1	Ord_5186	0.09	RICK WILSON	2	2
2	Ord_1209	0.09	RICK WILSON	2	2
3	Ord_3841	0.08	RICK WILSON	4	3
4	Ord_3855	0.07	RICK WILSON	5	4
5	Ord_3841	0.05	RICK WILSON	6	5
6	Ord_3845	0.05	RICK WILSON	6	5
7	Ord_1207	0.04	RICK WILSON	8	6
8	Ord_5186	0.03	RICK WILSON	9	7
9	Ord_5186	0.03	RICK WILSON	9	7
10	Ord_5186	0.03	RICK WILSON	9	7
11	Ord_1207	0.03	RICK WILSON	9	7
12	Ord_3828	0.03	RICK WILSON	9	7
13	Ord_1209	0.02	RICK WILSON	14	8
14	Ord_5202	0.01	RICK WILSON	15	9
15	Ord_3863	0.01	RICK WILSON	15	9
16	Ord_3867	0.01	RICK WILSON	15	9
17	Ord_5186	0.00	RICK WILSON	18	10
18	Ord_1213	0.00	RICK WILSON	18	10
19	Ord_3810	0.00	RICK WILSON	18	10
20	Ord_3851	0.00	RICK WILSON	18	10
21	Ord_3819	0.00	RICK WILSON	18	10

In [34]:
```
# Partition with Rank , Dense Rank , Row Number
query='''
with shipping as
(
select Ship_Mode,
       month(Ship_Date),
       count(*) as shipments
from market_star_schemas.shipping_dimen
group by Ship_Mode,month(Ship_Date)
)
select *,
       rank() over (partition by ship_mode order by shipments desc) as rank
       dense_rank() over (partition by ship_mode order by shipments desc) a
       row_number() over (partition by ship_mode order by shipments desc) a
from shipping;
'''
pd.read_sql_query(query,con_mysql)
```

Out[34]:

	Ship_Mode	month(Ship_Date)	shipments	rank_part	dense_rank_part	row_num
0	DELIVERY TRUCK	12	107	1	1	1
1	DELIVERY TRUCK	4	105	2	2	2
2	DELIVERY TRUCK	3	105	2	2	3
3	DELIVERY TRUCK	1	100	4	3	4
4	DELIVERY TRUCK	7	95	5	4	5
5	DELIVERY TRUCK	5	92	6	5	6
6	DELIVERY TRUCK	6	91	7	6	7
7	DELIVERY TRUCK	2	89	8	7	8
8	DELIVERY TRUCK	8	88	9	8	9
9	DELIVERY TRUCK	9	86	10	9	10
10	DELIVERY TRUCK	11	86	10	9	11
11	DELIVERY TRUCK	10	81	12	10	12
12	EXPRESS AIR	12	96	1	1	1
13	EXPRESS AIR	7	91	2	2	2
14	EXPRESS AIR	4	86	3	3	3
15	EXPRESS AIR	5	86	3	3	4
16	EXPRESS AIR	8	85	5	4	5
17	EXPRESS AIR	11	85	5	4	6
18	EXPRESS AIR	10	80	7	5	7
19	EXPRESS AIR	9	80	7	5	8
20	EXPRESS AIR	3	75	9	6	9
21	EXPRESS AIR	6	73	10	7	10
22	EXPRESS AIR	1	69	11	8	11
23	EXPRESS AIR	2	63	12	9	12
24	REGULAR AIR	8	512	1	1	1
25	REGULAR AIR	5	504	2	2	2
26	REGULAR AIR	9	488	3	3	3
27	REGULAR AIR	7	471	4	4	4
28	REGULAR AIR	12	469	5	5	5
29	REGULAR AIR	10	467	6	6	6
30	REGULAR AIR	11	467	6	6	7
31	REGULAR AIR	6	460	8	7	8
32	REGULAR AIR	3	459	9	8	9
33	REGULAR AIR	1	449	10	9	10
34	REGULAR AIR	4	434	11	10	11

	Ship_Mode	month(Ship_Date)	shipments	rank_part	dense_rank_part	row_num
35	REGULAR AIR	2	427	12	11	12

In [35]:
```
# Partition without CTE
query='''
select Ship_Mode,
       month(Ship_Date) as ship_month,
       count(*) as ship_count,
       rank() over (partition by Ship_Mode order by count(*) desc) as rank_
       dense_rank() over (partition by Ship_Mode order by count(*) desc) as
       row_number() over (partition by Ship_Mode order by count(*) desc) as
from market_star_schemas.shipping_dimen
group by Ship_Mode,month(Ship_Date)
'''
pd.read_sql_query(query,con_mysql)
```

Out[35]:

	Ship_Mode	ship_month	ship_count	rank_part	rank_part	rank_part
0	DELIVERY TRUCK	12	107	1	1	1
1	DELIVERY TRUCK	4	105	2	2	2
2	DELIVERY TRUCK	3	105	2	2	3
3	DELIVERY TRUCK	1	100	4	3	4
4	DELIVERY TRUCK	7	95	5	4	5
5	DELIVERY TRUCK	5	92	6	5	6
6	DELIVERY TRUCK	6	91	7	6	7
7	DELIVERY TRUCK	2	89	8	7	8
8	DELIVERY TRUCK	8	88	9	8	9
9	DELIVERY TRUCK	9	86	10	9	10
10	DELIVERY TRUCK	11	86	10	9	11
11	DELIVERY TRUCK	10	81	12	10	12
12	EXPRESS AIR	12	96	1	1	1
13	EXPRESS AIR	7	91	2	2	2
14	EXPRESS AIR	4	86	3	3	3
15	EXPRESS AIR	5	86	3	3	4
16	EXPRESS AIR	8	85	5	4	5
17	EXPRESS AIR	11	85	5	4	6
18	EXPRESS AIR	10	80	7	5	7
19	EXPRESS AIR	9	80	7	5	8
20	EXPRESS AIR	3	75	9	6	9
21	EXPRESS AIR	6	73	10	7	10
22	EXPRESS AIR	1	69	11	8	11
23	EXPRESS AIR	2	63	12	9	12
24	REGULAR AIR	8	512	1	1	1
25	REGULAR AIR	5	504	2	2	2
26	REGULAR AIR	9	488	3	3	3
27	REGULAR AIR	7	471	4	4	4
28	REGULAR AIR	12	469	5	5	5
29	REGULAR AIR	10	467	6	6	6
30	REGULAR AIR	11	467	6	6	7
31	REGULAR AIR	6	460	8	7	8
32	REGULAR AIR	3	459	9	8	9
33	REGULAR AIR	1	449	10	9	10
34	REGULAR AIR	4	434	11	10	11

	Ship_Mode	ship_month	ship_count	rank_part	rank_part	rank_part
35	REGULAR AIR	2	427	12	11	12

In [36]:
```
# Partition & Window
query='''
select Ship_Mode,
       month(Ship_Date) as ship_month,
       count(*) as ship_count,
       rank() over w as rank_part,
       dense_rank() over w as rank_part,
       row_number() over w as rank_part
from market_star_schemas.shipping_dimen
group by Ship_Mode,month(Ship_Date)
window w as (partition by Ship_Mode order by count(*) desc)
'''
pd.read_sql_query(query,con_mysql)
```

Out[36]:

	Ship_Mode	ship_month	ship_count	rank_part	rank_part	rank_part
0	DELIVERY TRUCK	12	107	1	1	1
1	DELIVERY TRUCK	4	105	2	2	2
2	DELIVERY TRUCK	3	105	2	2	3
3	DELIVERY TRUCK	1	100	4	3	4
4	DELIVERY TRUCK	7	95	5	4	5
5	DELIVERY TRUCK	5	92	6	5	6
6	DELIVERY TRUCK	6	91	7	6	7
7	DELIVERY TRUCK	2	89	8	7	8
8	DELIVERY TRUCK	8	88	9	8	9
9	DELIVERY TRUCK	9	86	10	9	10
10	DELIVERY TRUCK	11	86	10	9	11
11	DELIVERY TRUCK	10	81	12	10	12
12	EXPRESS AIR	12	96	1	1	1
13	EXPRESS AIR	7	91	2	2	2
14	EXPRESS AIR	4	86	3	3	3
15	EXPRESS AIR	5	86	3	3	4
16	EXPRESS AIR	8	85	5	4	5
17	EXPRESS AIR	11	85	5	4	6
18	EXPRESS AIR	10	80	7	5	7
19	EXPRESS AIR	9	80	7	5	8
20	EXPRESS AIR	3	75	9	6	9
21	EXPRESS AIR	6	73	10	7	10
22	EXPRESS AIR	1	69	11	8	11
23	EXPRESS AIR	2	63	12	9	12
24	REGULAR AIR	8	512	1	1	1
25	REGULAR AIR	5	504	2	2	2
26	REGULAR AIR	9	488	3	3	3
27	REGULAR AIR	7	471	4	4	4
28	REGULAR AIR	12	469	5	5	5
29	REGULAR AIR	10	467	6	6	6
30	REGULAR AIR	11	467	6	6	7
31	REGULAR AIR	6	460	8	7	8
32	REGULAR AIR	3	459	9	8	9
33	REGULAR AIR	1	449	10	9	10
34	REGULAR AIR	4	434	11	10	11

	Ship_Mode	ship_month	ship_count	rank_part	rank_part	rank_part
35	REGULAR AIR	2	427	12	11	12

Running Total & Moving Average

In [107]:
```
# Connect to market_star_schemas database
engine_mysql = sqlalchemy.create_engine('mysql+pymysql://root:root@localhos
con_mysql = engine_mysql.connect()
```

In [108]:
```
query='''
with ship_info as
(
select s.Ship_Date,
       sum(m.Shipping_Cost) as shipping_total
from market_star_schema.market_fact_full as m
inner join market_star_schema.shipping_dimen as s
on (m.Ship_id=s.Ship_id)
group by s.Ship_Date
)
select *,
       sum(shipping_total) over (order by ship_date rows UNBOUNDED PRECEDIN
       avg (shipping_total) over (order by ship_date rows 6 PRECEDING) as m
from ship_info;
'''
pd.read_sql_query(query,con_mysql)
```

Out[108]:

	Ship_Date	shipping_total	running_total	moving_avg
0	2009-08-07	27.23	27.23	27.230000
1	2010-05-26	0.99	28.22	14.110000
2	2010-07-27	2.50	30.72	10.240000
3	2010-07-28	9.75	40.47	10.117500
4	2010-11-11	14.30	54.77	10.954000
5	2011-02-26	3.37	58.14	9.690000
6	2011-05-30	0.50	58.64	8.377143
7	2011-10-31	4.86	63.50	5.181429
8	2011-12-26	36.09	99.59	10.195714
9	2011-12-27	61.76	161.35	18.661429
10	2011-12-31	8.30	169.65	18.454286

Using Lead to perform Date Diff between current & previous order date

In [109]:
```
query='''
with cust_order as
(
select c.Customer_Name as Customer_Name,
       m.Ord_id as Ord_id,
       o.Order_Date as Order_Date
from market_star_schema.market_fact_full as m
left outer join
market_star_schema.cust_dimen as c
on (m.Cust_id=c.Cust_id)
left outer join
market_star_schema.orders_dimen as o
on (o.Ord_id = m.Ord_id)
group by c.Customer_Name,m.Ord_id,o.Order_Date
),
date_summary as
(
select *,
       lead(order_date,1,'2001-01-01') over (order by order_date) as  next_
from cust_order
)
select *,
       datediff(next_date,order_date) as date_diff
from date_summary;
'''
pd.read_sql_query(query,con_mysql)
```

Out[109]:

	Customer_Name	Ord_id	Order_Date	next_date	date_diff
0	AARON BERGMAN	Ord_5485	2009-01-07	2009-07-07	181
1	AARON BERGMAN	Ord_5406	2009-07-07	2010-05-26	323
2	AARON HAWKINS	Ord_4743	2010-05-26	2010-07-27	62
3	AARON BERGMAN	Ord_5446	2010-07-27	2010-09-11	46
4	AARON BERGMAN	Ord_5456	2010-09-11	2011-02-24	166
5	AARON HAWKINS	Ord_2978	2011-02-24	2011-05-28	93
6	AARON BERGMAN	Ord_31	2011-05-28	2011-10-30	155
7	AARON HAWKINS	Ord_1925	2011-10-30	2011-12-25	56
8	AARON HAWKINS	Ord_2207	2011-12-25	2011-12-29	4
9	AARON HAWKINS	Ord_4725	2011-12-29	2001-01-01	-4014

CASE STATEMENT

In [110]:
```
query='''
select Market_fact_id,
       Profit,
       case when Profit < -500 then 'Huge Loss'
            when Profit between -500 and 0 then 'Bearable Loss'
            when Profit between 0 and 500 then 'Decent Profit'
            when Profit > 500 then 'Huge Profit'
            END
            as 'Profit Category'
from market_star_schema.market_fact_full
'''
pd.read_sql_query(query,con_mysql)
```

Out[110]:

	Market_fact_id	Profit	Profit Category
0	1	-30.51	Bearable Loss
1	2	4.56	Decent Profit
2	3	1148.90	Huge Profit
3	4	729.34	Huge Profit
4	5	1219.87	Huge Profit
5	6	-47.64	Bearable Loss
6	7	1.32	Decent Profit
7	8	1137.91	Huge Profit
8	9	45.84	Decent Profit
9	10	-27.72	Bearable Loss
10	11	1675.98	Huge Profit
11	12	79.34	Decent Profit
12	13	23.12	Decent Profit
13	14	-693.23	Huge Loss
14	15	-317.48	Bearable Loss

SQL Function

In [ ]:
```
use market_star_schema

DELIMITER $$

CREATE FUNCTION profit_cat(profit INT)
RETURNS VARCHAR(30) DETERMINISTIC

BEGIN
DECLARE msg varchar(30);
IF profit < -500 THEN
 SET msg = 'Huge Loss';
ELSEIF profit between -500 AND 0  THEN
 SET msg = 'Bearable Loss';
ELSEIF profit between 0 AND 500 THEN
 SET msg = 'Decent Profit';
ELSE
 SET msg= 'Huge Profit';
END IF;
RETURN msg;

END ;
$$
DELIMITER ;
```

In [111]:
```
query='''
SELECT profit_cat(-600)
'''
pd.read_sql_query(query,con_mysql)
```

Out[111]:

	profit_cat(-600)
0	Huge Loss

In [112]:
```
query='''
SELECT profit_cat(-200)
'''
pd.read_sql_query(query,con_mysql)
```

Out[112]:

	profit_cat(-200)
0	Bearable Loss

In [113]:
```
query='''
SELECT profit_cat(600)
'''
pd.read_sql_query(query,con_mysql)
```

Out[113]:

	profit_cat(600)
0	Huge Profit

SQL PROCEDURE

In [ ]:
```
use market_star_schema

DELIMITER $$

CREATE PROCEDURE get_sales(sales_input INT)

BEGIN
SELECT DISTINCT cust_id,
                round(sales) as sales_amount
FROM market_fact_full
WHERE round(sales) > sales_input
ORDER BY sales;

END $$
DELIMITER ;
```

In [114]:
```
query='''
CALL get_sales(350)
'''
pd.read_sql_query(query,con_mysql)
```

Out[114]:

	cust_id	sales_amount
0	Cust_708	466.0
1	Cust_839	1411.0
2	Cust_1818	2338.0
3	Cust_839	3364.0
4	Cust_1641	3410.0
5	Cust_1641	4072.0
6	Cust_1818	4233.0
7	Cust_1818	4702.0

END