# Assignment No: 6

Name: Pratik Wani

- Database Information:
  - Created Database name as Assignment_6
  - Created one tables: Sales

SQLQuery1.sql - PRA...(PRATIK\dell (88))* | SQL_Coding_Challen...(PRATIK\dell (84))

```sql
create database assignment_6;
use assignment_6;

create table Sales (
    Region varchar(50),
    Product char(50),
    SalesAmount int
);

-- Insert some sample data
insert into Sales values('North','Electrical',400);
insert into Sales values('North','Mechanical',1200);
insert into Sales values('South','Electrical',800);
insert into Sales values('South','Mechanical',1700);
insert into Sales values('East','Electrical',900);
insert into Sales values('East','Mechanical',2000);
```

142 %

Messages

```
(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)
```

- Total Aggregations using SQL Queries:
  - When we use aggregations on the over() and order by clause we called it as total aggregation.
  - We can also use aggregations by partitioning the result set
  - If we did not use partition by then aggregation is applied on whole set.

SQLQuery1.sql - PRA...(PRATIK\dell (88))* | SQL_Coding_Challen...(PRATIK\dell (84))

```sql
insert into Sales values('East','Mechanical',2000);

--Total Aggregations using SQL Queries.
select
    Region,
    Product,
    SalesAmount,
    AVG(salesamount) OVER () grand_average,
    SUM(salesamount) OVER () grand_sum
from
    Sales;
```

142 %

Results | Messages

| | Region | Product | SalesAmount | grand_average | grand_sum |
|---|---|---|---|---|---|
| 1 | North | Electrical | 400 | 1166 | 7000 |
| 2 | North | Mechanical | 1200 | 1166 | 7000 |
| 3 | South | Electrical | 800 | 1166 | 7000 |
| 4 | South | Mechanical | 1700 | 1166 | 7000 |
| 5 | East | Electrical | 900 | 1166 | 7000 |
| 6 | East | Mechanical | 2000 | 1166 | 7000 |

- OVER and PARTITION BY Clause in SQL Queries:
  - The OVER clause is use to specifies the rows over which a window function.
  - It can include both the PARTITION BY clause and the ORDER BY clause.
  - PARTITION BY clause is used to divide the result set into groups based on some column.

SQLQuery1.sql - PRA...(PRATIK\dell (88))* | SQL_Coding_Challen...(PRATIK\dell (84))

```sql
--Over() and Partition by

select Region, Product, SalesAmount,
Rank() over(partition by Region order by SalesAmount) Rank
from sales;
```

142 %

Results | Messages

| | Region | Product | SalesAmount | Rank |
|---|---|---|---|---|
| 1 | East | Electrical | 900 | 1 |
| 2 | East | Mechanical | 2000 | 2 |
| 3 | North | Electrical | 400 | 1 |
| 4 | North | Mechanical | 1200 | 2 |
| 5 | South | Electrical | 800 | 1 |
| 6 | South | Mechanical | 1700 | 2 |

- Total Aggregation using OVER and PARTITION BY:
  - We can also use aggregations by partitioning the result set.
  - It applies the aggregation function based on the partition.

SQLQuery1.sql - PRA...(PRATIK\dell (88))* | SQL_Coding_Challen...(PRATIK\dell (84))

```sql
--Total Aggregation using OVER and PARTITION BY in SQL Queries

select
    Region,
    Product,
    SalesAmount,
    AVG(salesamount) OVER (partition by region) avg_by_region,
    SUM(salesamount) OVER (partition by product) total_by_product
from
    Sales;
```

142 %

Results | Messages

| | Region | Product | SalesAmount | avg_by_region | total_by_product |
|---|---|---|---|---|---|
| 1 | East | Electrical | 900 | 1450 | 2100 |
| 2 | North | Electrical | 400 | 800 | 2100 |
| 3 | South | Electrical | 800 | 1250 | 2100 |
| 4 | South | Mechanical | 1700 | 1250 | 4900 |
| 5 | North | Mechanical | 1200 | 800 | 4900 |
| 6 | East | Mechanical | 2000 | 1450 | 4900 |

- Creating Sub totals:
  - To create the sub totals we need Group by Rollup.
  - Subtotals is sum for Specific sections.
  - Here in these query Subtotal is created for region wise and Product wise.

SQLQuery1.sql - PRA...(PRATIK\dell (88))* | SQL_Coding_Challen...(PRATIK\dell (84))

```sql
--creation subtotals

select Region, Product,
        sum(SalesAmount) as total
from sales
Group by
    Rollup(Region, Product);
```

142 %

Results | Messages

| | Region | Product | total |
|---|---|---|---|
| 1 | East | Electrical | 900 |
| 2 | East | Mechanical | 2000 |
| 3 | East | NULL | 2900 |
| 4 | North | Electrical | 400 |
| 5 | North | Mechanical | 1200 |
| 6 | North | NULL | 1600 |
| 7 | South | Electrical | 800 |
| 8 | South | Mechanical | 1700 |
| 9 | South | NULL | 2500 |
| 10 | NULL | NULL | 7000 |

- Regexp:
  - Powerful tools for pattern matching and string manipulation..
  - Provide syntax for searching, matching, and manipulating text.
  - Here in these query Subtotal is created for region wise and Product wise.
  - '^' = Matches start of the line
  - '$' = Matches end of the line
  - [] = Defines a character class

SQLQuery1.sql - PRA...(PRATIK\dell (88))* | SQL_Coding_Challen...(PRATIK\dell (84))

```sql
--RegExp
--regex are not directly supported in ms server
--that's why we need to use Like or Pathindex

--Gives product starting with E
select distinct Product from sales where Product like'E%';

--Gives Region ending with h
select distinct Region from sales where Region like'%h';
```

142 %

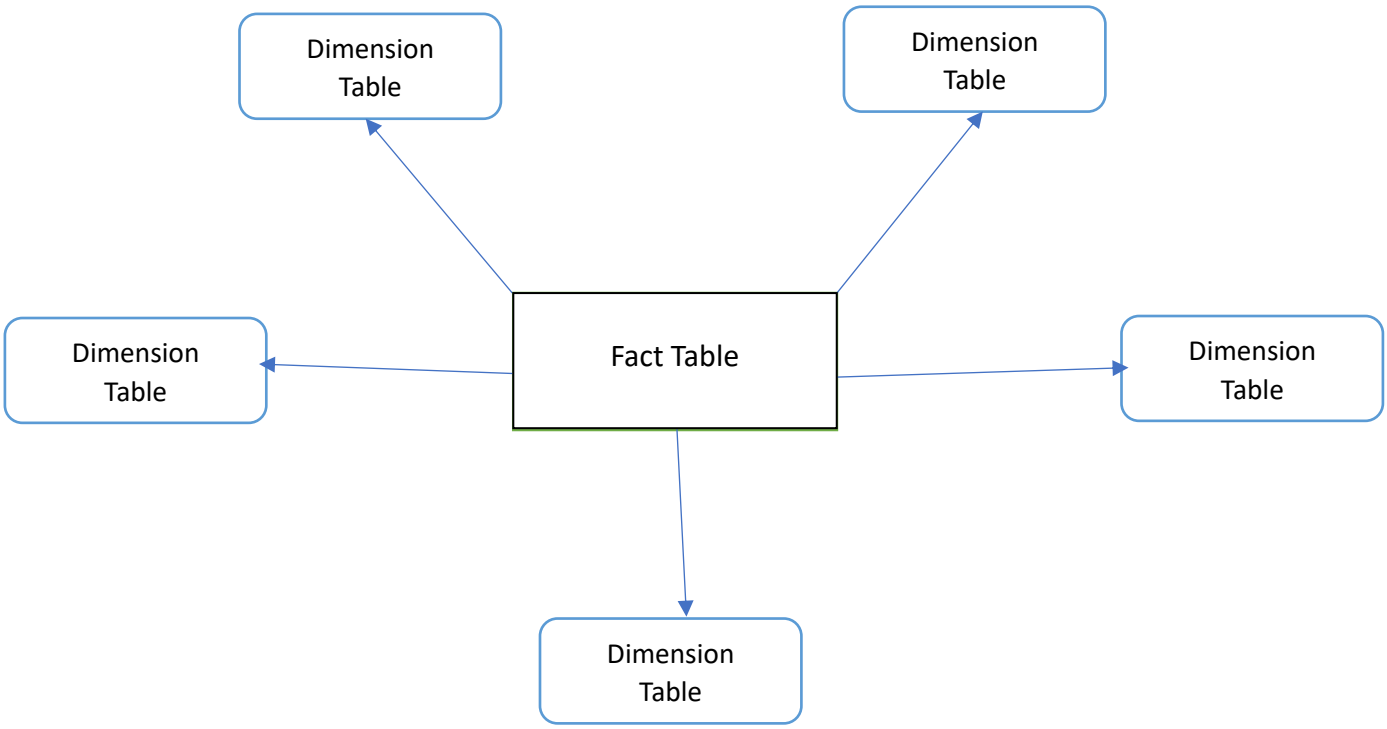
Results | Messages

| | Product |
|---|---|
| 1 | Electrical |

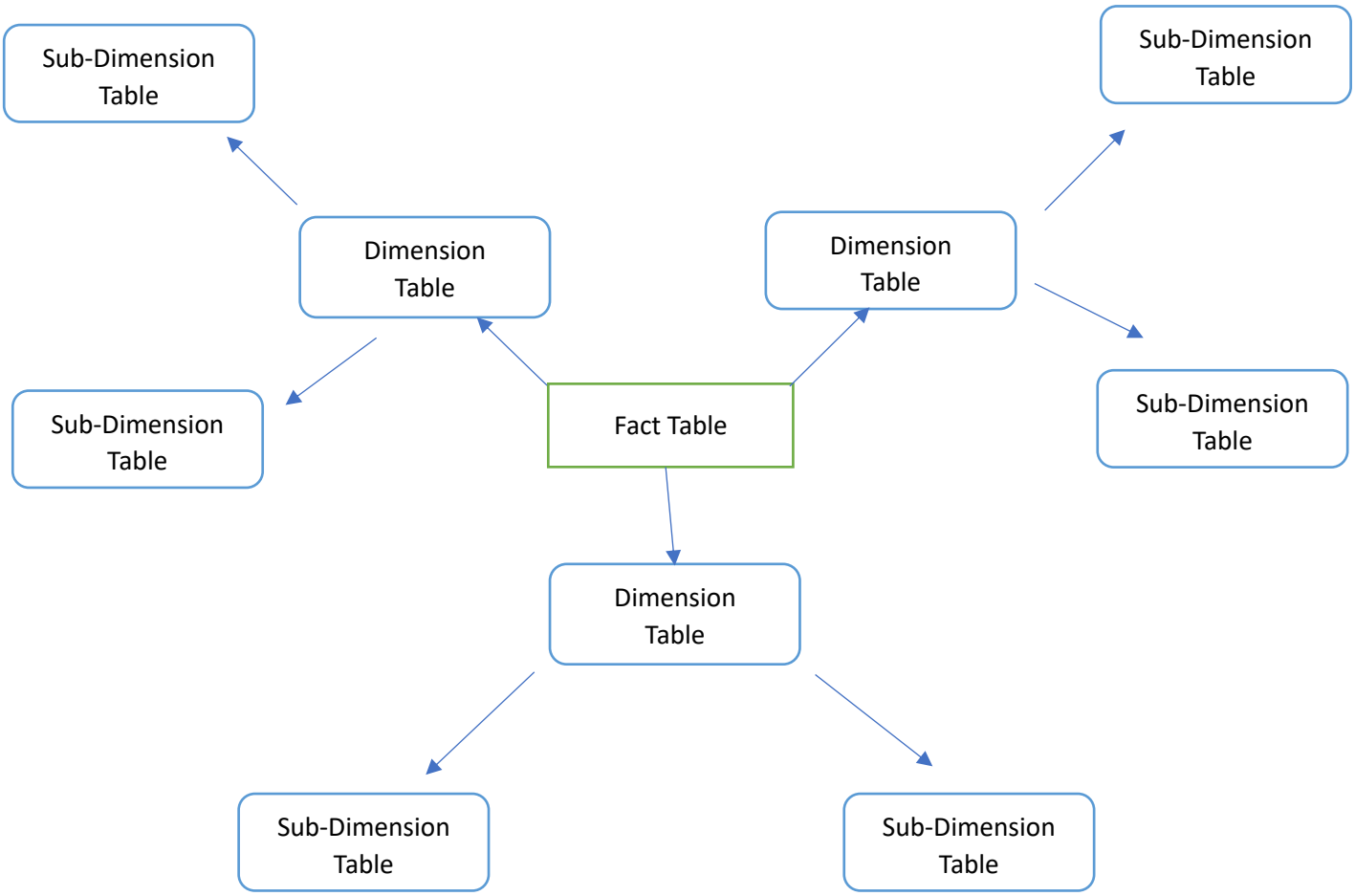
| | Region |
|---|---|
| 1 | North |
| 2 | South |

- Star schema:
  - It's a data model which is used to organize the data in the databases.
  - It consists of fact table connected to one or more dimension tables through foreign key relationships.
  - After designing the database it looks like a star that's the reason it known as star schema.

1. Fact Table:
   a. The fact table contains quantitative data.
   b. The data in fact table can be use for analytical purpose.
   c. Each row in fact table shows us specific event or transaction.
2. Dimension Tables:
   a. One or Many dimension table can be present in start Schema
   b. It contain attributes that provide details about the data in the fact table.
   c. Fact table and dimension table are connected using foreign key relationships



- Snowflaking schema:
  - Snowflaking schema is data model that is an extension of a star schema.
  - Dimension tables are broken down into subdimensions.
  - Dimension tables are broken using concept of normalization
  - After designing the database it looks like a Snowflake that's the reason it known as snowflaking schema.



## Materialized view:

- Also known as a snapshot, It's a database object that contains the results of a precomputed query.
- It's different from the normal view as view only stores the query and every we invoke the view the query is recomputed every time.
- But materialized view actually stores the result set of particular query.
- We need to keep the materialized view updated.
- In m. s. server there is no direct syntax to create materialized view but we can create it by using UNIQUE CLUSTERD INDEX

SQLQuery1.sql - PRA...(PRATIK\dell (88))* | SQL_Coding_Challen...(PRATIK\dell (84))

```sql
--Materialized view

CREATE VIEW [dbo].myindex4
WITH SCHEMABINDING
AS
SELECT Region, Product
FROM [dbo].[Sales]
GROUP BY Region, Product;

CREATE UNIQUE CLUSTERED INDEX myindex ON [dbo].myindex4 (Region, Product);
```

142 %

Messages

```
Commands completed successfully.

Completion time: 2024-01-26T09:55:35.4581388+05:30
```