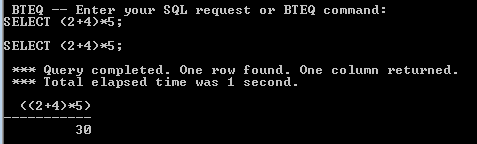
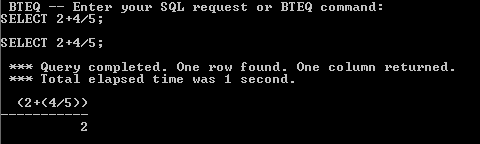
**Math Functions**

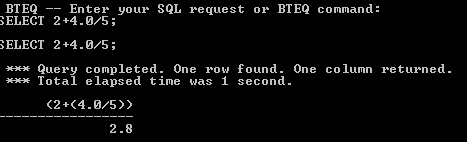
**What is the answer to this Math Question?**



**What is the answer to this Math Question?**



**What is the answer to this Math Question?**



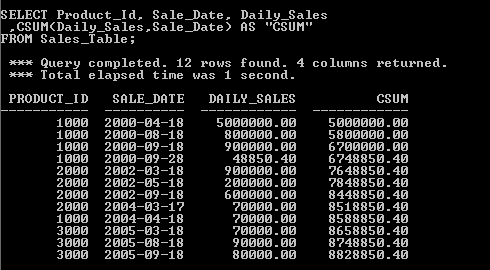
**OLAP Functions**

**Calculate the Cumulative Sum after Sorting the date**

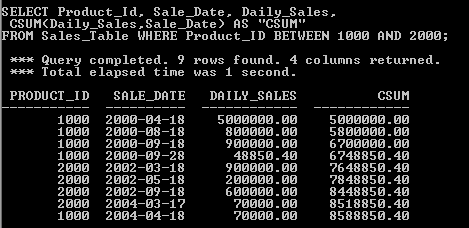
SELECT Product\_ID, Sale\_Date, Daily\_Sales

,CSUM(Daily\_Sales, Sale\_Date) AS “CSUM”

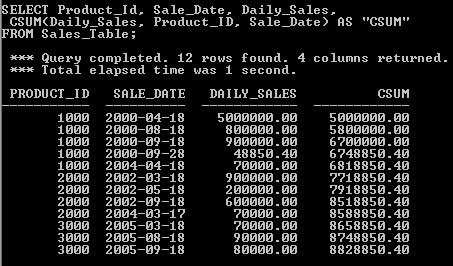
FROM Sales\_Table;



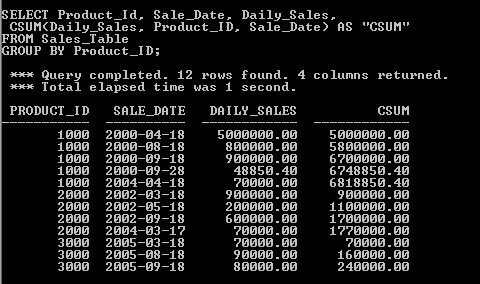
**OLAP Major Sort Key**



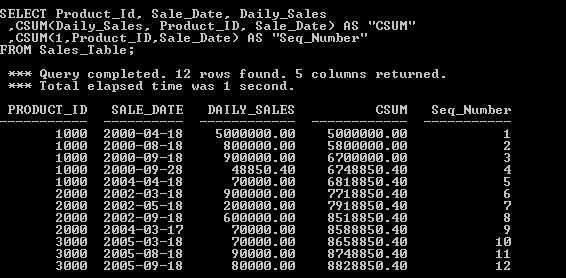
**The OLAP Major Key and Minor Key**



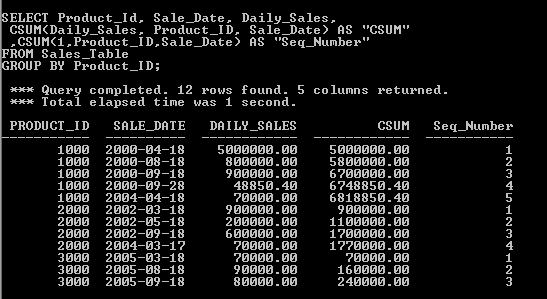
**GROUP BY In TERADATA OLAP Syntax**



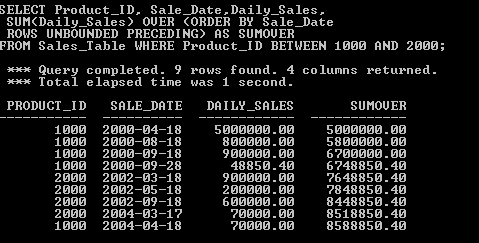
**CSUM the Number 1 to get Sequential Number**



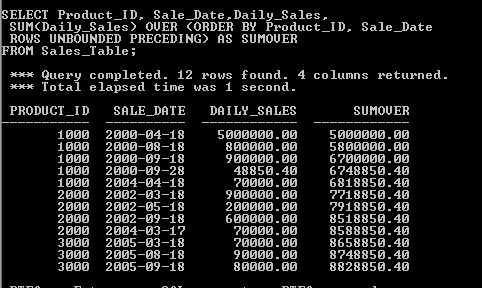
**Using GROUP BY**



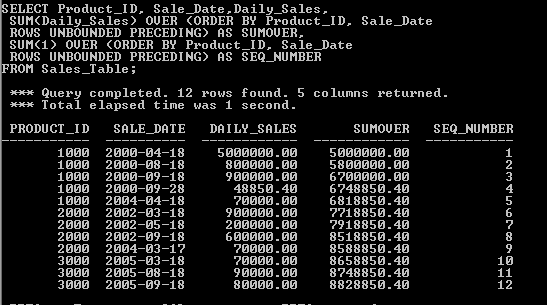
**ANSI CSUM**



**The ANSI CSUM Major and Minor Sort Keys**



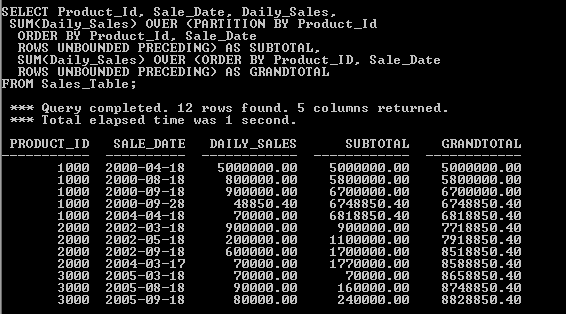
**ANSI CSUM Getting Sequential Number**



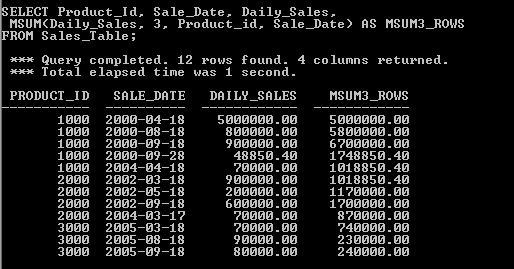
**The ANSI OLAP Reset with PARTITION BY**



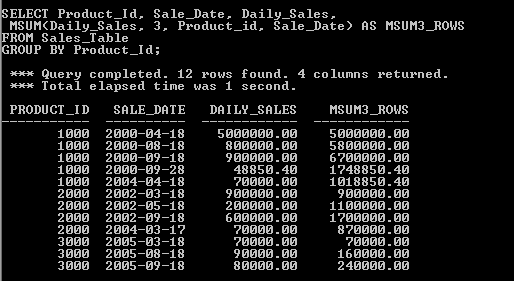
**PARTITION BY Single OLAP function**



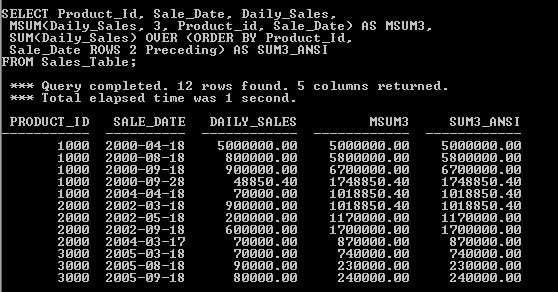
**MSUM**

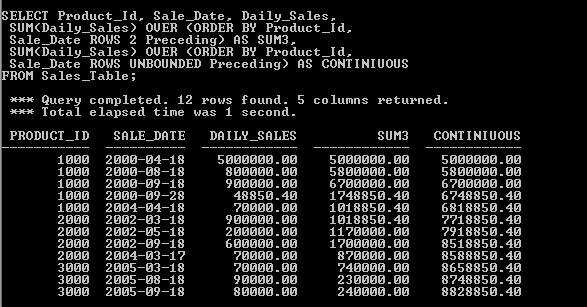


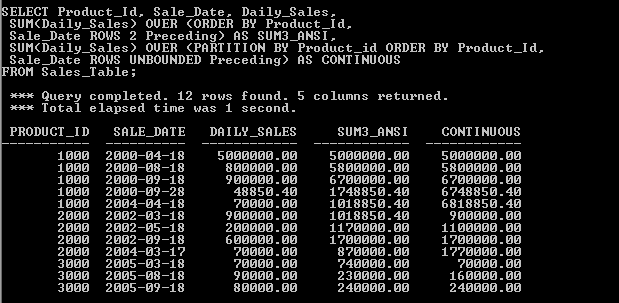
**GROUP BY in MSUM**



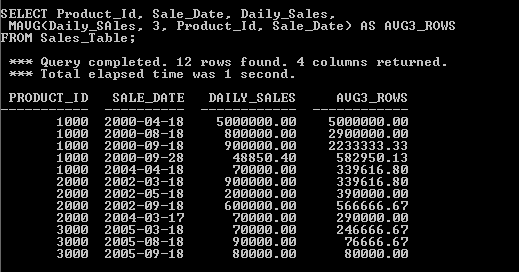
**Both Teradata Moving Sum and ANSI Version**



**MSUM for every 3 rows vs Continuous****Partition By using ANSI**



**Moving Average**

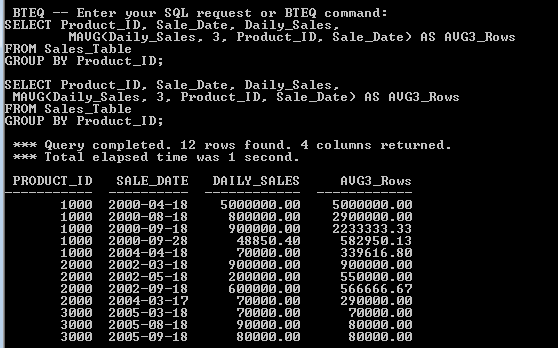


SELECT Product\_ID, Sale\_Date, Daily\_Sales,

MAVG(Daily\_Sales, 3, Product\_ID, Sale\_Date) AS AVG3\_Rows

FROM Sales\_Table

GROUP BY Product\_ID;



**Both the Teradata Moving Average and ANSI Version**

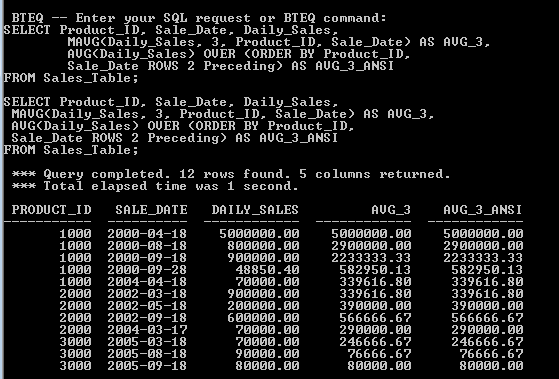
SELECT Product\_ID, Sale\_Date, Daily\_Sales,

MAVG(Daily\_Sales, 3, Product\_ID, Sale\_Date) AS AVG\_3,

AVG(Daily\_Sales) OVER (ORDER BY Product\_ID,

Sale\_Date ROWS 2 Preceding) AS AVG\_3\_ANSI

FROM Sales\_Table;



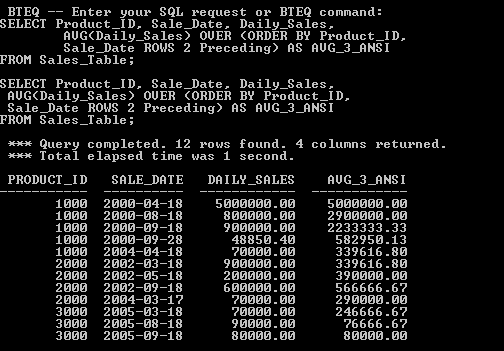
**How is the total calculated?**

SELECT Product\_ID, Sale\_Date, Daily\_Sales,

AVG(Daily\_Sales) OVER (ORDER BY Product\_ID,

Sale\_Date ROWS 2 Preceding) AS AVG\_3\_ANSI

FROM Sales\_Table;



**Moving average every 3 rows Vs. a continuous average**

SELECT Product\_ID, Sale\_Date, Daily\_Sales,

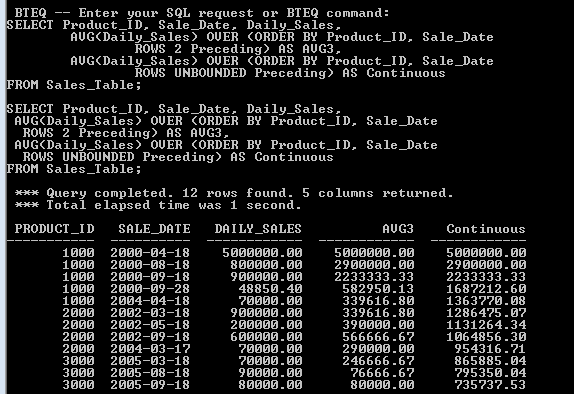
AVG(Daily\_Sales) OVER (ORDER BY Product\_ID, Sale\_Date

ROWS 2 Preceding) AS AVG3,

AVG(Daily\_Sales) OVER (ORDER BY Product\_ID, Sale\_Date

ROWS UNBOUNDED Preceding) AS Continuous

FROM Sales\_Table;



**Partition BY Resets an ANSI OLAP**

SELECT Product\_ID, Sale\_Date, Daily\_Sales,

AVG(Daily\_Sales) OVER (ORDER BY Product\_ID, Sale\_Date

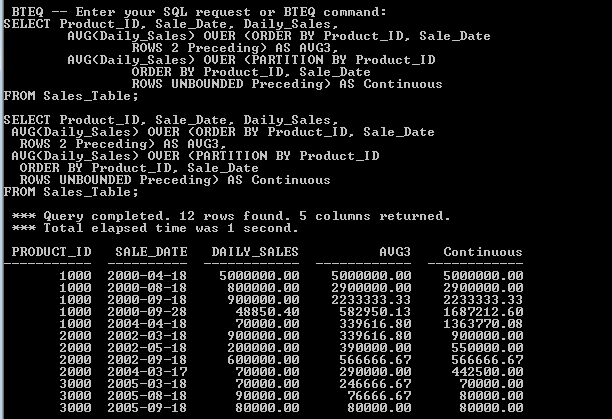
ROWS 2 Preceding) AS AVG3,

AVG(Daily\_Sales) OVER (PARTITION BY Product\_ID

ORDER BY Product\_ID, Sale\_Date

ROWS UNBOUNDED Preceding) AS Continuous

FROM Sales\_Table;

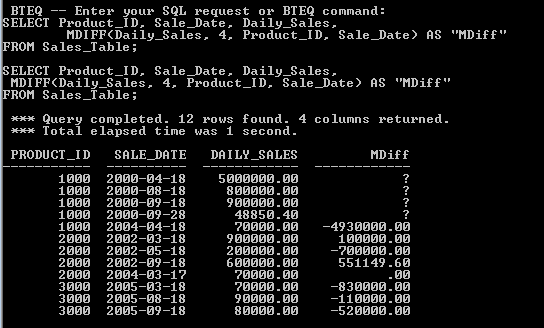


**The Moving Difference(MDIFF)**

SELECT Product\_ID, Sale\_Date, Daily\_Sales,

MDIFF(Daily\_Sales, 4, Product\_ID, Sale\_Date) AS "MDiff"

FROM Sales\_Table;



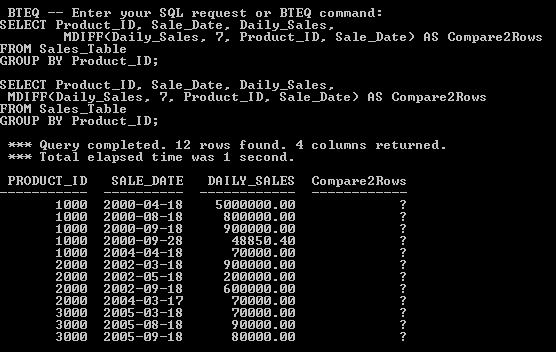
**Trouble Shooting the Moving Difference(MDIFF)**

SELECT Product\_ID, Sale\_Date, Daily\_Sales,

MDIFF(Daily\_Sales, 7, Product\_ID, Sale\_Date) AS Compare2Rows

FROM Sales\_Table

GROUP BY Product\_ID;



**Using the RESET WHEN option in Teradata(V13)**

SELECT Product\_ID, Sale\_Date, Daily\_Sales,

ROW\_NUMBER() OVER (PARTITION BY Product\_ID ORDER BY Sale\_Date

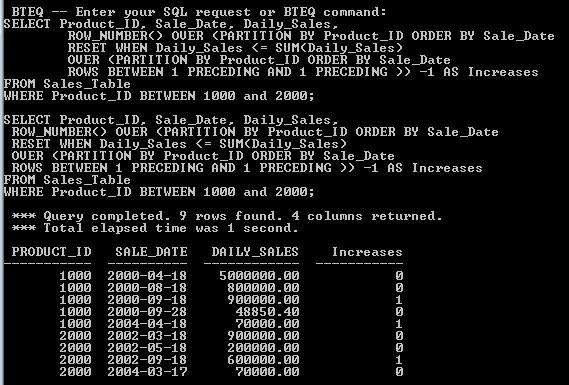
RESET WHEN Daily\_Sales <= SUM(Daily\_Sales)

OVER (PARTITION BY Product\_ID ORDER BY Sale\_Date

ROWS BETWEEN 1 PRECEDING AND 1 PRECEDING )) -1 AS Increases

FROM Sales\_Table

WHERE Product\_ID BETWEEN 1000 and 2000;



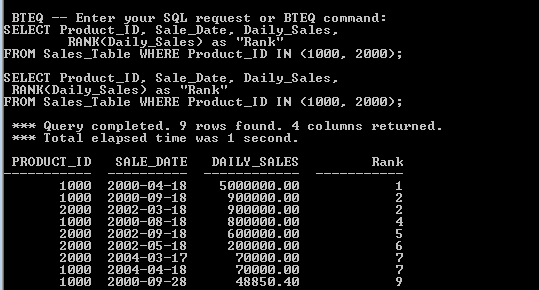
**How Many Months per Product\_ID has Revenue Increased?**

**The RANK Command**

SELECT Product\_ID, Sale\_Date, Daily\_Sales,

RANK(Daily\_Sales) as "Rank"

FROM Sales\_Table WHERE Product\_ID IN (1000, 2000);

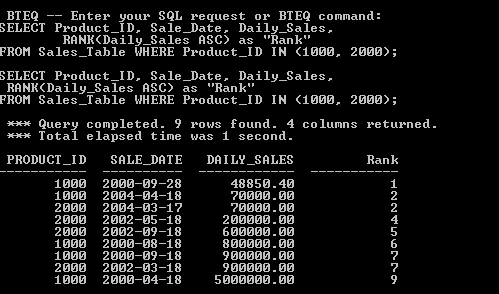


**How to get Rank to Sort in Ascending Order**

SELECT Product\_ID, Sale\_Date, Daily\_Sales,

RANK(Daily\_Sales ASC) as "Rank"

FROM Sales\_Table WHERE Product\_ID IN (1000, 2000);



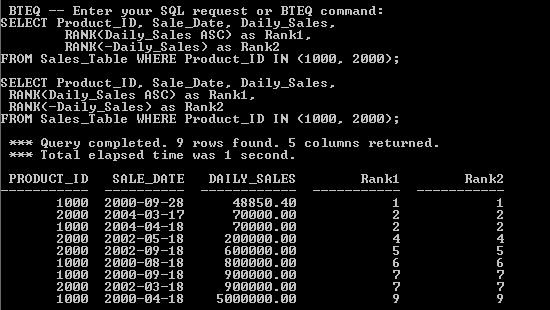
**Method 2**

SELECT Product\_ID, Sale\_Date, Daily\_Sales,

RANK(Daily\_Sales ASC) as Rank1,

RANK(-Daily\_Sales) as Rank2

FROM Sales\_Table WHERE Product\_ID IN (1000, 2000);

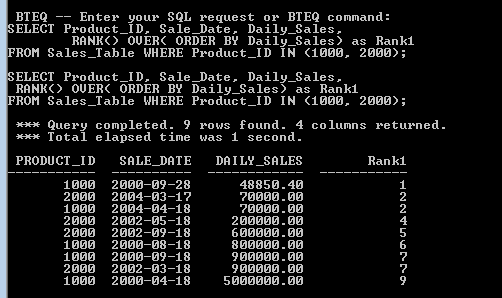


**RANK Using ANSI Syntax Defaults to Ascending order**

SELECT Product\_ID, Sale\_Date, Daily\_Sales,

RANK() OVER( ORDER BY Daily\_Sales) as Rank1

FROM Sales\_Table WHERE Product\_ID IN (1000, 2000);

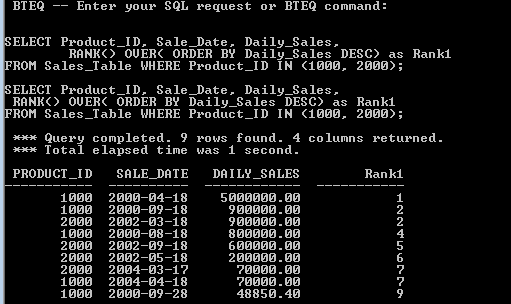


**RANK Using ANSI Syntax Defaults to Sort in Descending Order**

SELECT Product\_ID, Sale\_Date, Daily\_Sales,

RANK() OVER( ORDER BY Daily\_Sales DESC) as Rank1

FROM Sales\_Table WHERE Product\_ID IN (1000, 2000);

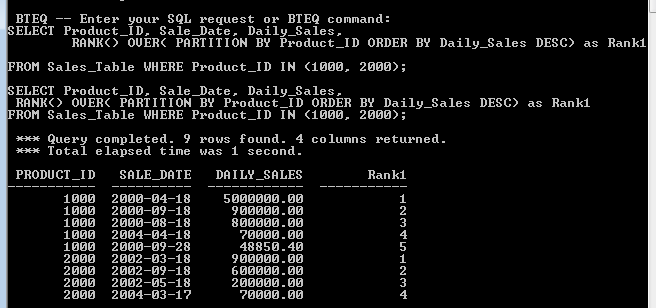


**RANK() OVER AND PARTITION BY**

SELECT Product\_ID, Sale\_Date, Daily\_Sales,

RANK() OVER( PARTITION BY Product\_ID ORDER BY Daily\_Sales DESC) as Rank1

FROM Sales\_Table WHERE Product\_ID IN (1000, 2000);



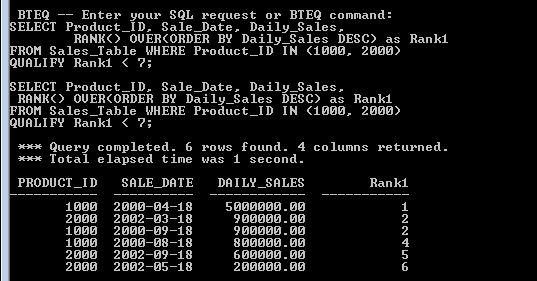
**RANK() OVER AND QUALIFY**

SELECT Product\_ID, Sale\_Date, Daily\_Sales,

RANK() OVER(ORDER BY Daily\_Sales DESC) as Rank1

FROM Sales\_Table WHERE Product\_ID IN (1000, 2000)

QUALIFY Rank1 < 7;



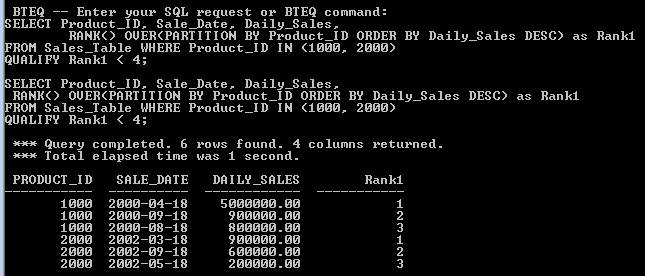
**RANK() OVER AND PARTITION BY WITH A QUALIFY**

SELECT Product\_ID, Sale\_Date, Daily\_Sales,

RANK() OVER(PARTITION BY Product\_ID ORDER BY Daily\_Sales DESC) as Rank1

FROM Sales\_Table WHERE Product\_ID IN (1000, 2000)

QUALIFY Rank1 < 4;



**QUALIFY and WHERE**

SELECT Product\_ID, Sale\_Date, Daily\_Sales,

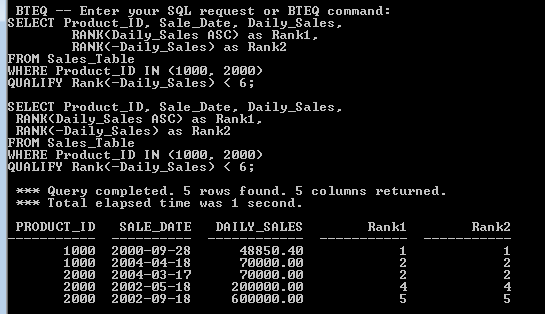
RANK(Daily\_Sales ASC) as Rank1,

RANK(-Daily\_Sales) as Rank2

FROM Sales\_Table

WHERE Product\_ID IN (1000, 2000)

QUALIFY Rank(-Daily\_Sales) < 6;



**How to simplify the QUALIFY statement**

SELECT Product\_ID, Sale\_Date, Daily\_Sales,

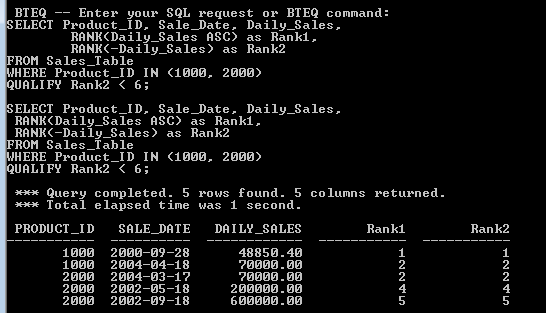
RANK(Daily\_Sales ASC) as Rank1,

RANK(-Daily\_Sales) as Rank2

FROM Sales\_Table

WHERE Product\_ID IN (1000, 2000)

QUALIFY Rank2 < 6;



**The QUALIFY statement without TIES**

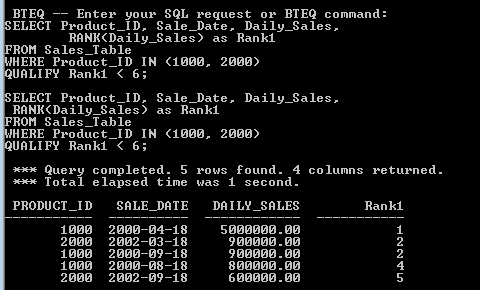
SELECT Product\_ID, Sale\_Date, Daily\_Sales,

RANK(Daily\_Sales) as Rank1

FROM Sales\_Table

WHERE Product\_ID IN (1000, 2000)

QUALIFY Rank1 < 6;



**The QUALIFY statement with TIES**

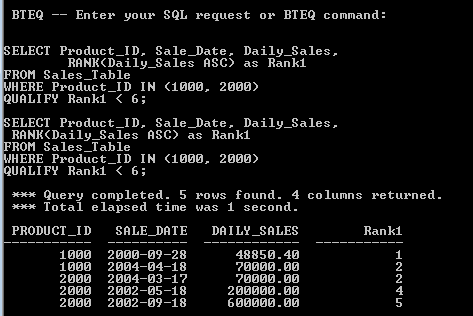
SELECT Product\_ID, Sale\_Date, Daily\_Sales,

RANK(Daily\_Sales ASC) as Rank1

FROM Sales\_Table

WHERE Product\_ID IN (1000, 2000)

QUALIFY Rank1 < 6;



**The QUALIFY statement with TIES Brings Back Extra Rows**

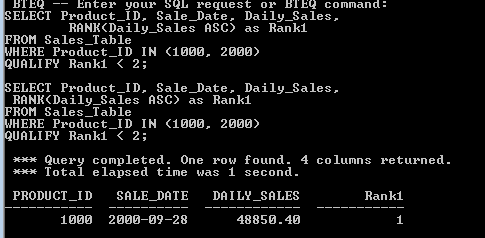
SELECT Product\_ID, Sale\_Date, Daily\_Sales,

RANK(Daily\_Sales ASC) as Rank1

FROM Sales\_Table

WHERE Product\_ID IN (1000, 2000)

QUALIFY Rank1 < 2;



**Mixing sort order by QUALIFY statement**

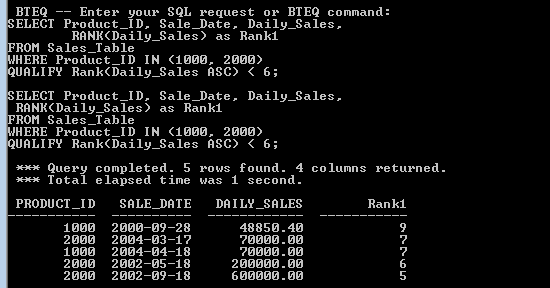
SELECT Product\_ID, Sale\_Date, Daily\_Sales,

RANK(Daily\_Sales) as Rank1

FROM Sales\_Table

WHERE Product\_ID IN (1000, 2000)

QUALIFY Rank(Daily\_Sales ASC) < 6;



**Rank to RESET**

SELECT Product\_ID, Sale\_Date, Daily\_Sales,

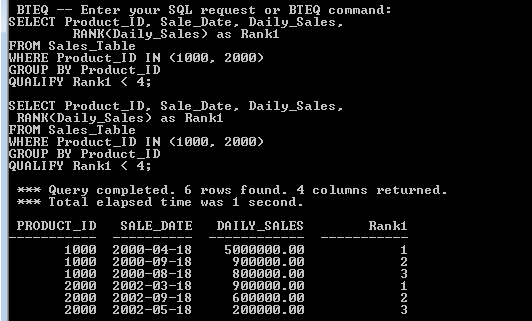
RANK(Daily\_Sales) as Rank1

FROM Sales\_Table

WHERE Product\_ID IN (1000, 2000)

GROUP BY Product\_ID

QUALIFY Rank1 < 4;



**Name those Sort Orders**

**RANK(Daily\_Sales) as NON\_ANSI\_Rank**

SELECT Product\_ID, Sale\_Date, Daily\_Sales,

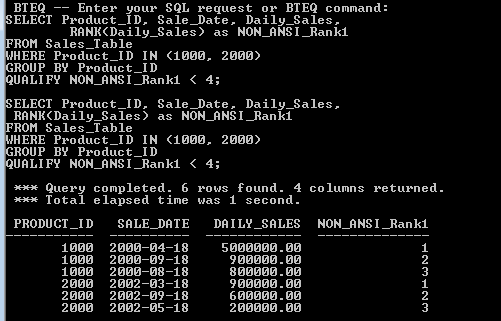
RANK(Daily\_Sales) as NON\_ANSI\_Rank1

FROM Sales\_Table

WHERE Product\_ID IN (1000, 2000)

GROUP BY Product\_ID

QUALIFY NON\_ANSI\_Rank1 < 4;



**PERCENT\_RANK() OVER**

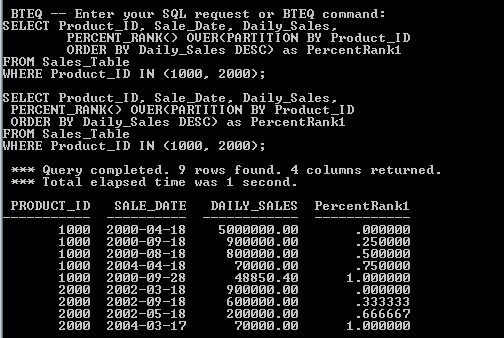
SELECT Product\_ID, Sale\_Date, Daily\_Sales,

PERCENT\_RANK() OVER(PARTITION BY Product\_ID

ORDER BY Daily\_Sales DESC) as PercentRank1

FROM Sales\_Table

WHERE Product\_ID IN (1000, 2000);



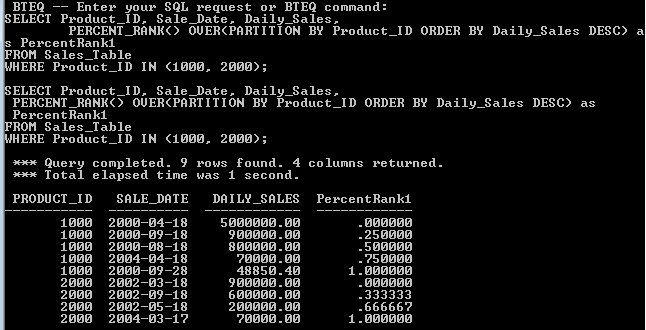
**What causes the Product\_ID to reset?**

SELECT Product\_ID, Sale\_Date, Daily\_Sales,

PERCENT\_RANK() OVER(PARTITION BY Product\_ID ORDER BY Daily\_Sales DESC) as PercentRank1

FROM Sales\_Table

WHERE Product\_ID IN (1000, 2000);



**Substrings and Positioning Functions**

1. THE CHARACTERS COMMAND COUNTS CHARACTER

SELECT FIRST\_NAME

,CHARACTERS(FIRST\_NAME) AS LNTHS

FROM EMPLOYEE\_TABLE

WHERE CHARACTERS(FIRST\_NAME) <7

ORDER BY 1;



1. THE CHARACTER COMMAND CAN COUNT SPACES TOO

SELECT ` R a m ` AS FIRST\_NAME

,CHARACTERS(` R a m `) AS LNTH

FROM EMPLOYEE\_TABLE;

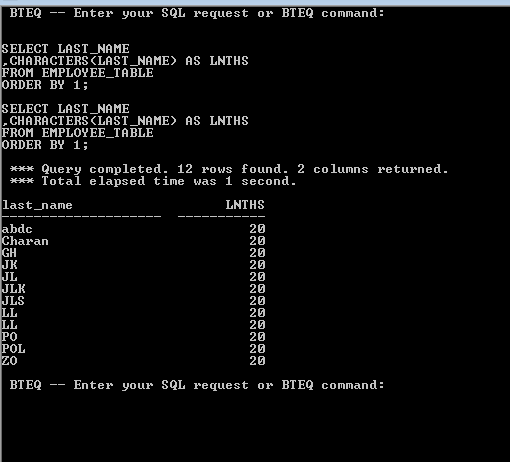
1. THE CHARACTER COMMAND AND CHAR(20) DATA

SELECT LAST\_NAME

,CHARACTERS(LAST\_NAME) AS LNTHS

FROM EMPLOYEE\_TABLE

ORDER BY 1;



1. TRIM COMMAND

SELECT LAST\_NAME

,CHARACTERS(TRIM(LAST\_NAME)) AS LNTHS

FROM EMPLOYEE\_TABLE

ORDER BY 1;



TRIM IS USED FOR TROUBLESHOOTING CHARACTER COMMAND

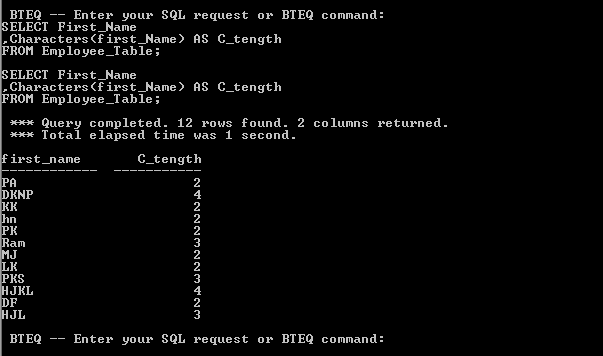
THE TRIM COMMAND WILL TRIM OFF ALL SPACES BEFORE AND AFTER THE LAST\_NAME

1. CHARACTER AND CHARACTER\_LENGTH EQUIVALENT

**SELECT** First\_Name

,Characters(first\_Name) **AS** C\_tength

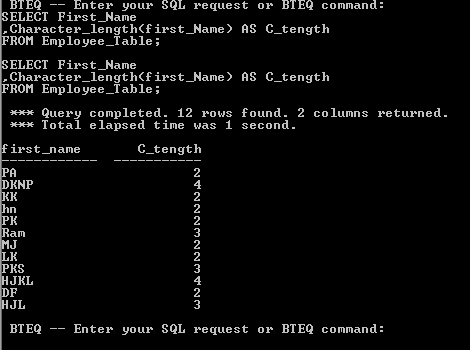
**FROM** Employee\_Table**;**



**SELECT** First\_Name

,Character\_length(first\_Name) **AS** C\_tength

**FROM** Employee\_Table**;**

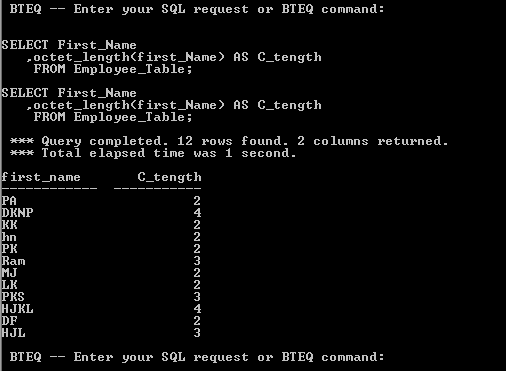


1. OCTET\_LENGTH

**SELECT** First\_Name

,octet\_length(first\_Name) **AS** C\_tength

**FROM** Employee\_Table**;**



1. The TRIM Command trims both Leading and Trailing Spaces

Select Last\_Name

,trim(Last\_Name) AS No\_Space

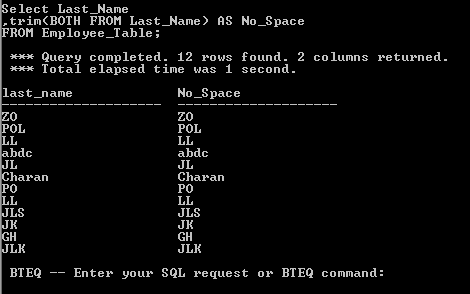
FROM Employee\_Table;



Select Last\_Name

,trim(BOTH FROM Last\_Name) AS No\_Space

FROM Employee\_Table;

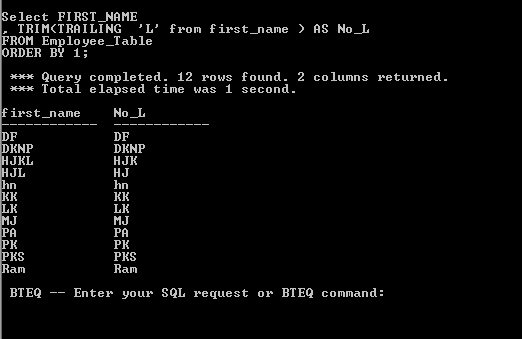


1. **Select** FIRST\_NAME

, TRIM(TRAILING 'L' **from** first\_name ) **AS** No\_L

**FROM** Employee\_Table

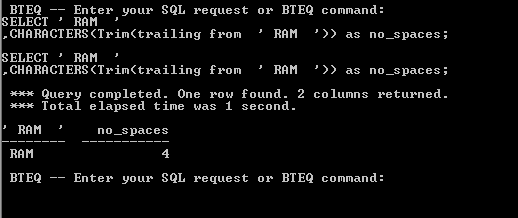
**ORDER** **BY** 1**;**



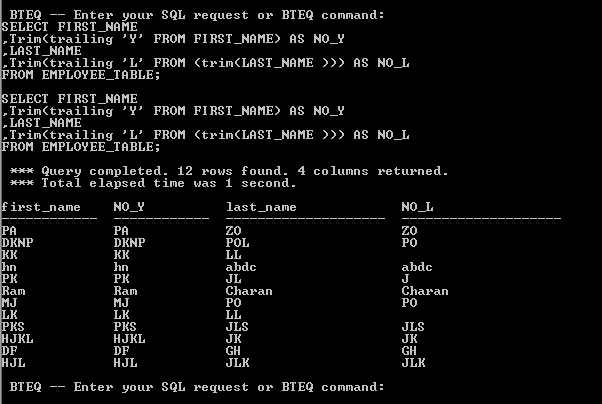
1. TRIM COMBINED WITH CHARACTER COMMAND



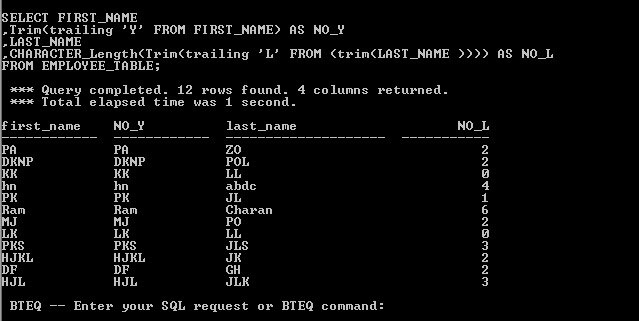
1. TRIM COMBINED WITH TRAIL



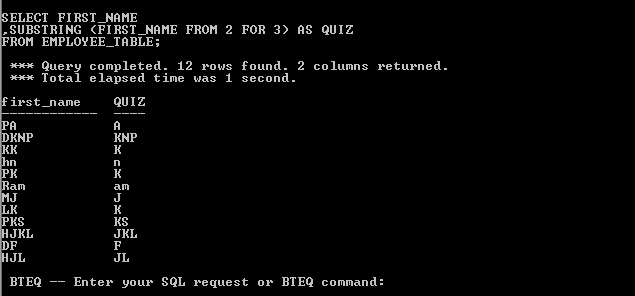
1. TRIMING TRAILING LETTERS



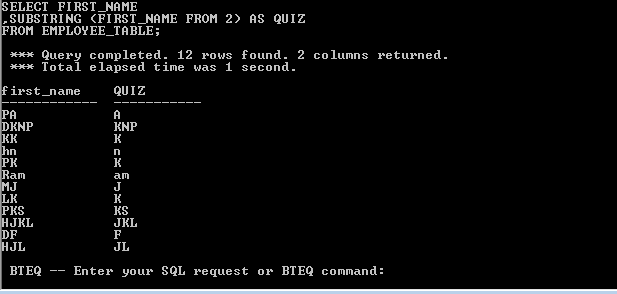
1. HOW TO TRIM TRAILING LETTER AND USE CHARACTER COMMAND



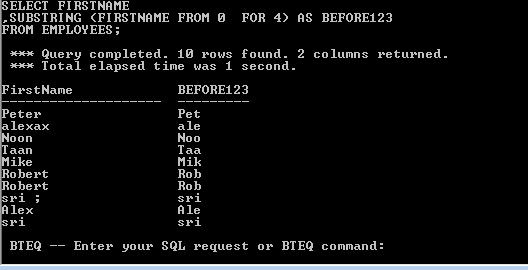
1. THE SUBSTRING COMMAND



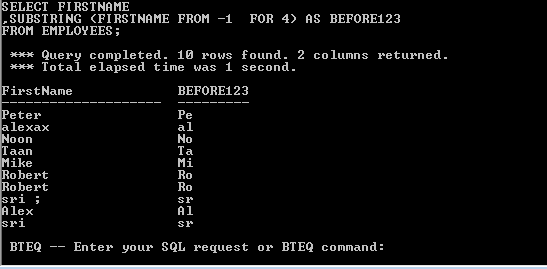
1. SUBSTRING WITH NO END VALUE



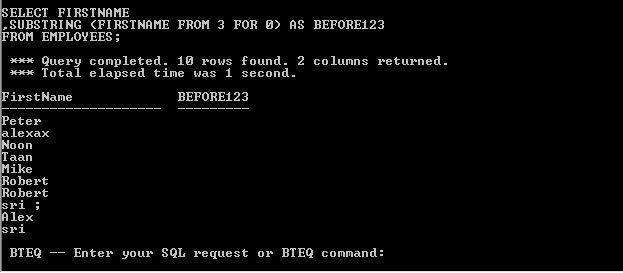
1. USING SUBSTRING TO MOVE BACKWARD



1. SUBSTRING WITH STARTING POSITION AS -1

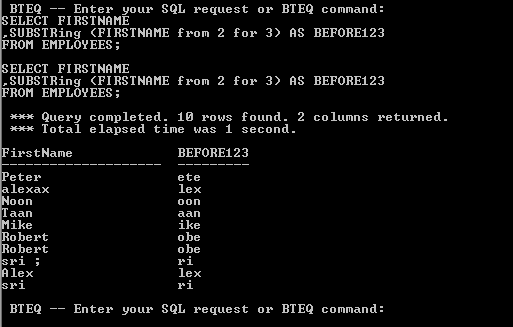


1. SUBSTRING WITH ENDING POSITION FOR 0

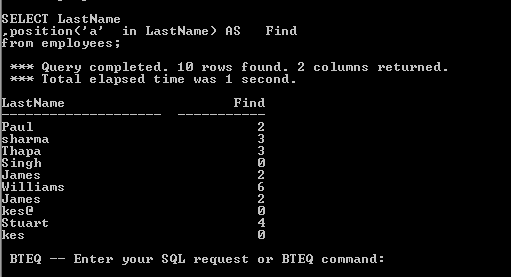


1. SUBSTRING and SUBSTR are equal, but use different syntax

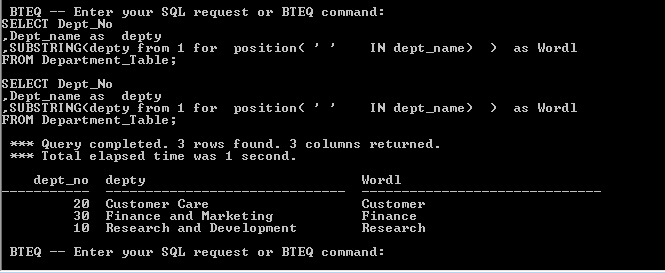


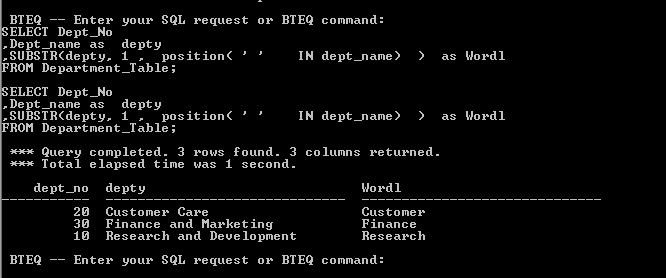


1. THE POSITION COMMAND FINDS A LETTER POSITION

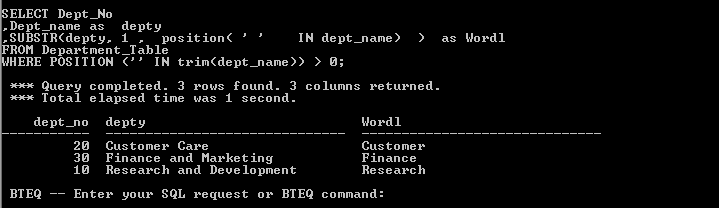


1. THE POSITION COMMAND WITH SUBSTRING



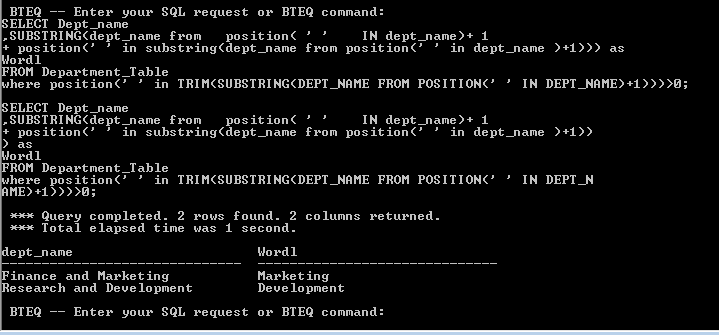


1. USING THE SUBQUERY TO FIND THE SECOND WORD ON



1. WHY TWO ROWS

THEY LOOK FOR TWO SPACES



1. CONCATENATION

