

## CONTENTS:

- 1) The Quantile Function
- 2) Top SQL Command Cheat Sheet
- 3) View Functions

# The Quantile Function

## The Quantile Function

### Overview

"There is flattery in friendship."  
- William Shakespeare

### The Quantile Function and Syntax

The syntax for the Quantile function:

```
QUANTILE (<partitions>, <column-name> ,<sort-key> [DESC | ASC])
[QUALIFY QUANTILE (<column-name>) {< | > | = | <= | >=}] <number-of-rows>]
```

A Quantile is used to divide rows into a number of categories or grouping of roughly the same number of rows in each group. The percentile is the QUANTILE most commonly used in business. This means that the request is based on a value of 100 for the number of partitions. It is also possible to have quartiles (based on 4), tertiles (based on 3) and deciles (based on 10).

By default, both the QUANTILE column and the QUANTILE value itself will be output in ascending sequence. As in some cases, the ORDER BY clause may be used to reorder the output for display. Here, the order of the output does not change the meaning of the output, unlike a summation where the values are being added together and all need to appear in the proper sequence.

### A Quantile Example

This example determines the percentile for every row in the Sales table based on the daily sales amount, and sorts it into sequence by the value being categorized, which is Daily\_Sales here.

```
SELECT Product_ID, Sale_Date, Daily_Sales
      ,QUANTILE (100, Daily_Sales ) AS "Quantile"
FROM   Sales_Table
WHERE  Product_ID < 3000
AND    Sale_Date > 1000930 ;
```

Product_ID	Sale_Date	Daily_Sales	Quantile
1000	10/02/2000	32800.50	0
2000	10/04/2000	32800.50	0
2000	10/02/2000	36021.93	25
1000	10/01/2000	40200.43	37
2000	10/03/2000	43200.18	50
1000	10/04/2000	54553.10	62
2000	10/01/2000	54850.29	75
1000	10/03/2000	64300.00	87

Notice that the amount of 32800.50 in the first two rows has the same percentile value. They are the same value, and will, therefore, be put into the same partition.

### A Quantile Example using DESC Mode

```
SELECT Product_ID ,Sale_Date ,Daily_Sales
      ,QUANTILE(100, Daily_Sales , Sale_Date DESC) AS "Quantile"
FROM   Sales_Table
WHERE  Product_ID < 3000
AND    Sale_Date >= 1000930 ;
```

Product_ID	Sale_Date	Daily_Sales	Quantile
1000	10/02/2000	32800.50	0
2000	10/04/2000	32800.50	10
1000	09/30/2000	36000.07	20
2000	10/02/2000	36021.93	30
1000	10/01/2000	40200.43	40

2000	10/03/2000	43200.18	50
2000	09/30/2000	49850.03	60
1000	10/04/2000	54553.10	70
2000	10/01/2000	54850.29	80
1000	10/03/2000	64300.00	90

Notice the first two rows. This is because the Sale date DESC, impacts the first two rows. Why? Since these rows have the same value, it uses the Sale\_Date column as a tiebreaker for the sequencing and makes them different from each other. Hence, they are assigned to different values in different partitions.

#### QUALIFY to find Products in the top Partitions

This example uses a QUALIFY to show only products that sell in the top 60 Percentile

```
SELECT Product_ID, Sale_Date, Daily_Sales
      ,QUANTILE(100, Daily_Sales, Sale_Date ) as "Percentile"
FROM   Sales_Table
QUALIFY "Percentile" >= 60 ;
```

Product_ID	Sale_Date	Daily_Sales	Percentile
2000	09/29/2000	48000.00	61
1000	09/28/2000	48850.40	66
2000	09/30/2000	49850.03	71
1000	09/29/2000	54500.22	76
1000	10/04/2000	54553.10	80
2000	10/01/2000	54850.29	85
3000	09/28/2000	61301.77	90
1000	10/03/2000	64300.00	95

Like the aggregate functions, OLAP functions must read all required rows before performing their operation. Therefore, the WHERE clause cannot be used. Where the aggregates use HAVING, the OLAP functions use QUALIFY. The QUALIFY evaluates the result to determine which ones to return.

#### QUALIFY to find Products in the top Partitions Sorted DESC

```
SELECT Product_ID, Sale_Date, Daily_Sales
      ,QUANTILE (100, Daily_Sales, Sale_Date ) as "Percentile"
FROM   Sales_Table
QUALIFY "Percentile" >= 70
ORDER BY "percentile" DESC ;
```

Product_ID	Sale_Date	Daily_Sales	Percentile
1000	10/03/2000	64300.00	95
3000	09/28/2000	61301.77	90
2000	10/01/2000	54850.29	85
1000	10/04/2000	54553.10	80
1000	09/29/2000	54500.22	76
2000	09/30/2000	49850.03	71

The ORDER BY changes the sequence of the rows being listed, not the meaning of the percentile. The above functions both determined that the highest number in the column is the highest percentile. The data value sequence ascends as the percentile ascends, or descends as the percentile descends. When the sort in the QUANTILE function is changed to ASC, the data value sequence changes to ascend as the percentile descends. The sequence of the percentile does not change. But, the data value sequence is changed to ascend (ASC) instead of the default, which is to descend (DESC).

#### QUALIFY to find Products in the top Partitions Sorted ASC

```
SELECT Product_ID ,Sale_Date, Daily_Sales
      ,QUANTILE (100, Daily_Sales ASC, Sale_Date) as "Percentile"
FROM   Sales_Table
QUALIFY "Percentile" >=70 ;
```

---

Product_ID	Sale_Date	Daily_Sales	Percentile
1000	10/02/2000	32800.50	71
2000	10/04/2000	32800.50	76
3000	10/01/2000	28000.00	80
3000	10/03/2000	21553.79	85
3000	10/02/2000	19678.94	90
3000	10/04/2000	15675.33	95

The example above uses the ASC to cause the data values to go contradictory to the percentile.

#### QUALIFY to find Products in top Partitions with Tiebreaker

```
SELECT Product_ID, Sale_Date, Daily_Sales
      ,QUANTILE (100, Daily_Sales ASC, Sale_Date ASC) as "Percentile"
FROM   Sales_Table
QUALIFY "Percentile" >= 70 ;
```

Product_ID	Sale_Date	Daily_Sales	Percentile
2000	10/04/2000	32800.50	71
1000	10/02/2000	32800.50	76
3000	10/01/2000	28000.00	80
3000	10/03/2000	21553.79	85
3000	10/02/2000	19678.94	90
3000	10/04/2000	15675.33	95

The next SELECT modifies the above query to incorporate the sale date as a tiebreaker and reverse the ordering for the two rows with sales of \$32,800.50.

#### Using Tertiles (Partitions of Four)

```
SELECT Product_ID, Sale_Date, Daily_Sales
      ,QUANTILE(4, Daily_Sales , Sale_Date ) AS "Quartiles"
FROM   Sales_Table
WHERE  Product_ID in (1000, 2000) ;
```

Product_ID	Sale_Date	Daily_Sales	Quartiles
1000	10/02/2000	32800.50	0
2000	10/04/2000	32800.50	0
1000	09/30/2000	36000.07	0
2000	10/02/2000	36021.93	0
1000	10/01/2000	40200.43	1
2000	09/28/2000	41888.88	1
2000	10/03/2000	43200.18	1
2000	09/29/2000	48000.00	2
1000	09/28/2000	48850.40	2
2000	09/30/2000	49850.03	2
1000	09/29/2000	54500.22	2
1000	10/04/2000	54553.10	3
2000	10/01/2000	54850.29	3
1000	10/03/2000	64300.00	3

Instead of 100, the example above uses a quartile (QUANTILE based on 4 partitions).

#### How Quantile Works

```
SELECT Product_ID, Sale_Date, Daily_Sales
      ,QUANTILE (4, Daily_Sales , Sale_Date ) AS "Quartiles"
FROM   Sales_Table WHERE Product_ID = 1000;
```

Product_ID	Sale_Date	Daily_Sales	Quartiles
1000	10/02/2000	32800.50	0
1000	09/30/2000	36000.07	0
1000	10/01/2000	40200.43	1

1000	09/28/2000	48850.40	1
1000	09/29/2000	54500.22	2
1000	10/04/2000	54553.10	2
1000	10/03/2000	64300.00	3

Assigning a different value to the <partitions> indicator of the QUANTILE function changes the number of partitions established. Each Quantile partition is assigned a number starting at 0, increasing to a value that is one less than the partition number specified. So, with a Quantile of 4, the partitions are 0 through 3 and for 10, the partitions are assigned 0 through 9. Then, all the rows are distributed as evenly as possible into each partition from highest to lowest values. Normally, extra rows with the lowest value begin back in the lowest numbered partitions.

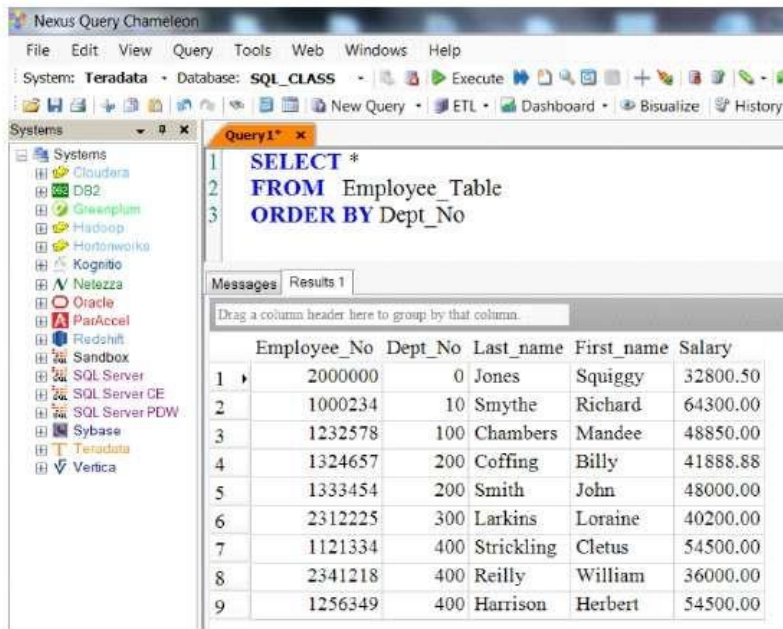
# Top SQL Commands Cheat Sheet

## Top SQL Commands Cheat Sheet

### Overview

"You miss 100 percent of the shots you never take."  
- Wayne Gretzky

### SELECT All Columns from a Table and Sort



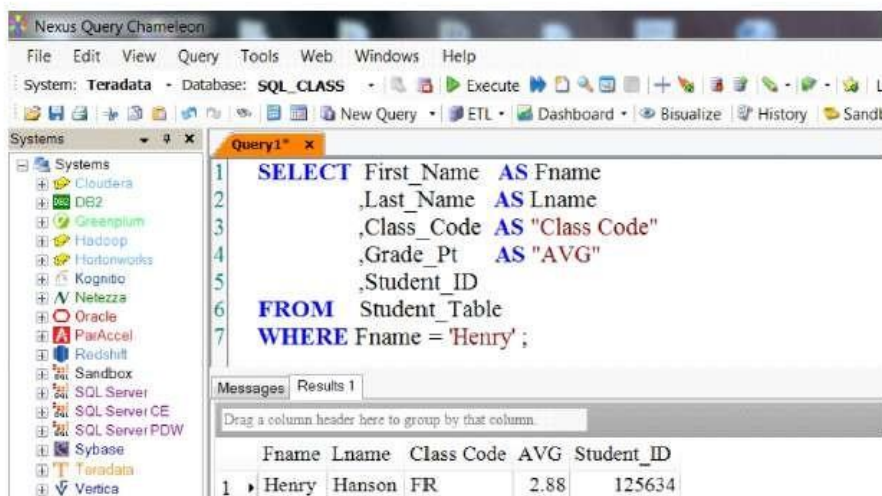
The screenshot shows the Nexus Query Chameleon interface. The query editor displays the following SQL code:

```
1 SELECT *
2 FROM Employee_Table
3 ORDER BY Dept_No
```

The Results tab shows the following data:

	Employee_No	Dept_No	Last_name	First_name	Salary
1	2000000	0	Jones	Squiggy	32800.50
2	1000234	10	Smythe	Richard	64300.00
3	1232578	100	Chambers	Mandee	48850.00
4	1324657	200	Coffing	Billy	41888.88
5	1333454	200	Smith	John	48000.00
6	2312225	300	Larkins	Loraine	40200.00
7	1121334	400	Strickling	Cletus	54500.00
8	2341218	400	Reilly	William	36000.00
9	1256349	400	Harrison	Herbert	54500.00

### Select Specific Columns and Limiting the Rows



The screenshot shows the Nexus Query Chameleon interface. The query editor displays the following SQL code:

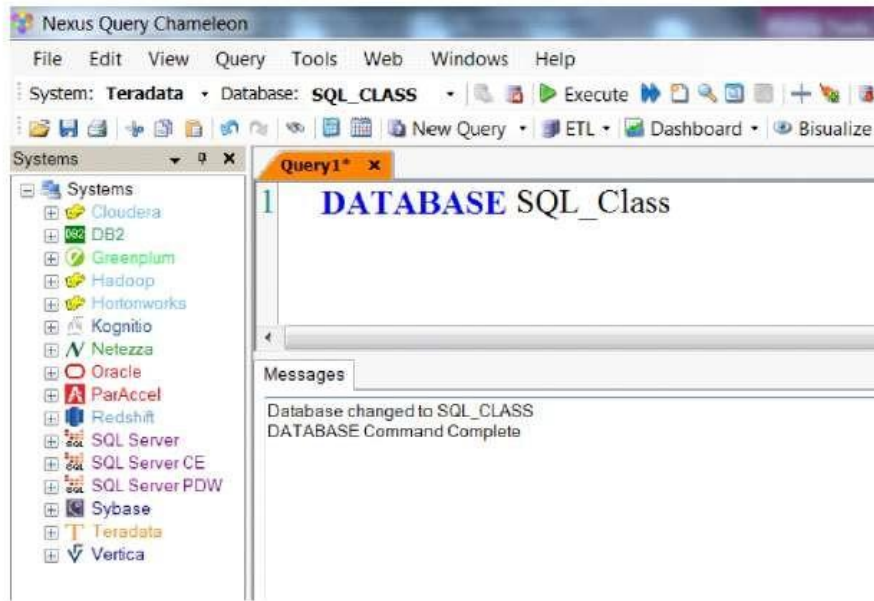
```
1 SELECT First_Name AS Fname
2       ,Last_Name AS Lname
3       ,Class_Code AS "Class Code"
4       ,Grade_Pt AS "AVG"
5       ,Student_ID
6 FROM Student_Table
7 WHERE Fname = 'Henry';
```

The Results tab shows the following data:

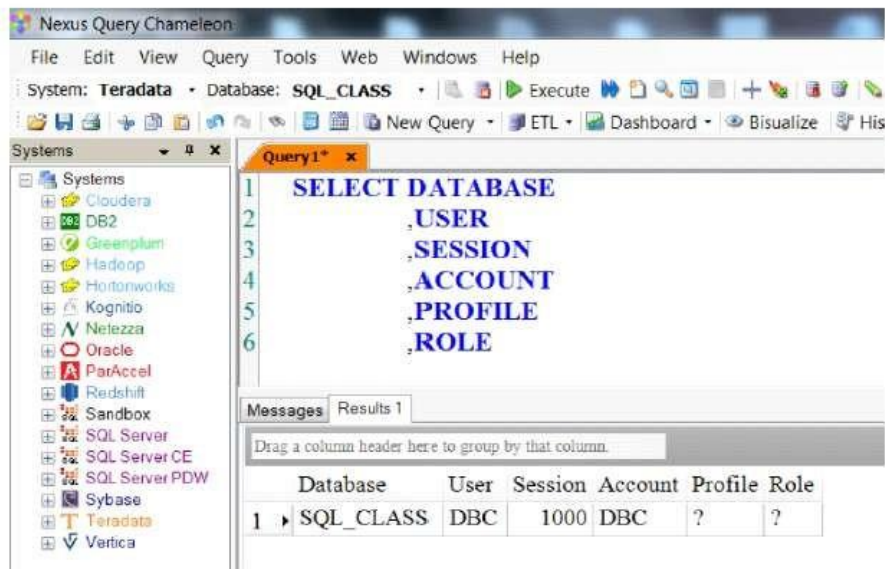
	Fname	Lname	Class Code	AVG	Student_ID
1	Henry	Hanson	FR	2.88	125634

### Changing your Default Database





Keywords that describe you



Select TOP Rows in a Rank Order

The screenshot shows the Nexus Query Chameleon interface. The 'Systems' pane on the left lists various databases, with 'Teradata' selected. The 'Query1' pane shows the following SQL query:

```

1 SELECT TOP 7 Last_Name,
2               Class_Code,
3               Grade_Pt
4 FROM Student_Table
5 ORDER BY Grade_Pt DESC ;
6

```

The 'Results' pane displays the following data:

	Last_Name	Class_Code	Grade_Pt
1	Thomas	FR	4.00
2	Bond	JR	3.95
3	Wilson	SO	3.80
4	Delaney	SR	3.35
5	Phillips	SR	3.00
6	Hanson	FR	2.88
7	Smith	SO	2.00

A Sample number of rows

The screenshot shows the Nexus Query Chameleon interface. The 'Systems' pane on the left lists various databases, with 'Teradata' selected. The 'Query1' pane shows the following SQL query:

```

1 SELECT *
2 FROM Employee_Table
3 SAMPLE 8

```

The 'Results' pane displays the following data:

	Employee_No	Dept_No	Last_name	First_name	Salary
1	2000000	0	Jones	Squiggy	32800.50
2	1256349	400	Harrison	Herbert	54500.00
3	1000234	10	Smythe	Richard	64300.00
4	1121334	400	Strickling	Cletus	54500.00
5	1232578	100	Chambers	Mandee	48850.00
6	2341218	400	Reilly	William	36000.00
7	1324657	200	Coffing	Billy	41888.88
8	2312225	300	Larkins	Loraine	40200.00

Getting a Sample Percentage of rows

The screenshot shows the Nexus Query Chameleon interface. The system is set to Teradata and the database is SQL\_CLASS. The query being executed is:

```
1 SELECT *
2 FROM Employee_Table
3 SAMPLE .50
```

The results are displayed in a table with the following columns: Employee\_No, Dept\_No, Last\_name, First\_name, and Salary.

	Employee_No	Dept_No	Last_name	First_name	Salary
1	1000234	10	Smythe	Richard	64300.00
2	1121334	400	Strickling	Cletus	54500.00
3	1232578	100	Chambers	Mandee	48850.00
4	2341218	400	Reilly	William	36000.00
5	2312225	300	Larkins	Loraine	40200.00

Find Information about a Database

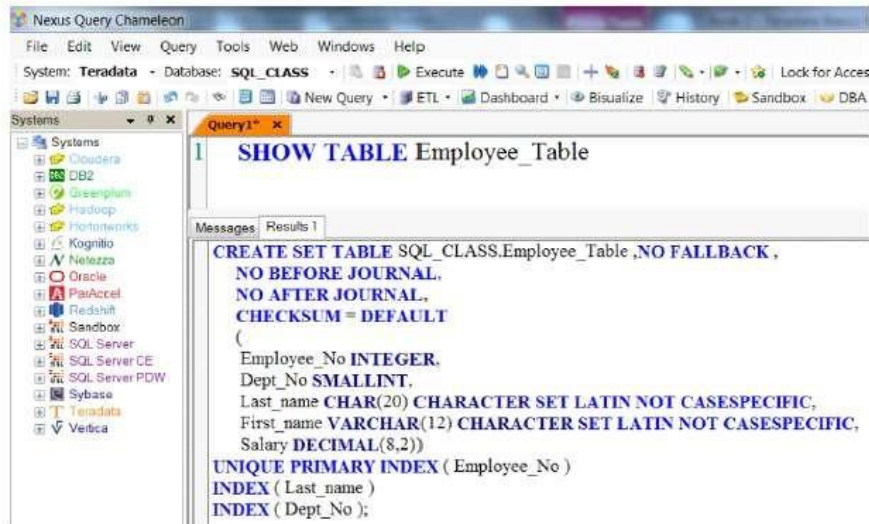
The screenshot shows the Nexus Query Chameleon interface. The system is set to Teradata and the database is SQL\_CLASS. The query being executed is:

```
1 HELP DATABASE SQL_Class
```

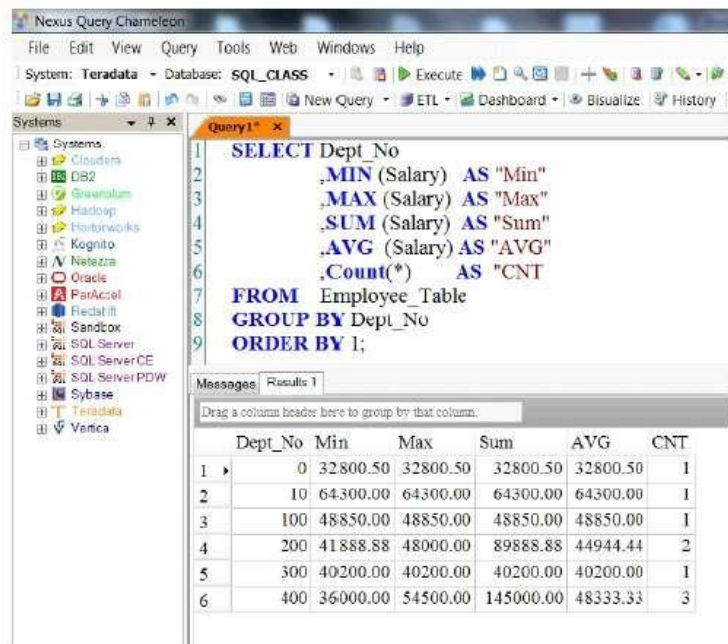
The results are displayed in a table with the following columns: Table/View/Macro name, Kind, Comment, Protection, and Creator Name.

	Table/View/Macro name	Kind	Comment	Protection	Creator Name
1	Addresses	T	?	N	DBC
2	AdvInsTLC	P	?	F	DBC
3	Claims	T	?	N	DBC
4	Course_table	T	?	N	DBC
5	CursorTest	P	?	F	DBC
6	Customer_table	T	?	N	DBC
7	Department_table	T	?	N	DBC
8	Employeea_table	T	?	N	DBC

Find information about a Table



### Using Aggregates



### Performing a Join



The screenshot shows the Nexus Query Chameleon interface. The query editor displays the following SQL code:

```

1 SELECT C.Customer_Number, C.Customer_Name,
2       O.Order_Number, O.Order_Total
3 FROM   Customer_Table as C,
4       Order_Table as O
5 WHERE  C.Customer_Number = O.Customer_Number ;

```

The results pane shows the following data:

	Customer_Number	Customer_Name	Order_Number	Order_Total
1	31323134	ACE Consulting	123552	5111.47
2	57896883	XYZ Plumbing	123777	23454.84
3	11111111	Billy's Best Choice	123512	8005.91
4	11111111	Billy's Best Choice	123456	12347.53
5	87323456	Databases N-U	123585	15231.62

### Performing a Join using ANSI Syntax

The screenshot shows the Nexus Query Chameleon interface. The query editor displays the following SQL code:

```

1 SELECT C.Customer_Number, C.Customer_Name,
2       O.Order_Number, O.Order_Total
3 FROM   Customer_Table as C
4 INNER JOIN
5       Order_Table as O
6 ON     C.Customer_Number = O.Customer_Number ;

```

The results pane shows the following data:

	Customer_Number	Customer_Name	Order_Number	Order_Total
1	31323134	ACE Consulting	123552	5111.47
2	57896883	XYZ Plumbing	123777	23454.84
3	11111111	Billy's Best Choice	123512	8005.91
4	11111111	Billy's Best Choice	123456	12347.53
5	87323456	Databases N-U	123585	15231.62

### Using Date, Time and Timestamp

The screenshot shows the Nexus Query Chameleon interface. The 'Systems' pane on the left lists various databases, with 'Teradata' selected. The 'Query1' pane shows the following SQL query:

```

1 SELECT Date
2      ,Current_Date
3      ,Time
4      ,Current_Time
5      ,Current_Timestamp(6)

```

The 'Results' pane displays the output of the query:

	Date	Date	Time	Current Time(0)	Current
1	09/13/2013	09/13/2013	15:10:04	15:10:04	09/13/

### Using Date Functions

The screenshot shows the Nexus Query Chameleon interface. The 'Systems' pane on the left lists various databases, with 'Teradata' selected. The 'Query1' pane shows the following SQL query:

```

1 SELECT Order_Date
2      ,Order_Date (FORMAT 'eeeebmmmbbdddyyyy')
3      ,Order_Date + 60 as "Due Date"
4      ,Order_Total (FORMAT '$$$$,$$$.$99')
5      ,ADD_MONTHS(Order_Date, 4) as Over_Due
6      ,EXTRACT(Month from Order_Date) as TheMonth
7 FROM   Order_Table
8 ORDER BY 1 ;

```

The 'Results' pane displays the output of the query:

	Order_Date	Order_Date	Due Date	Order_Total	Over_Due	TheMonth
1	05/04/1998	Monday May 04 1998	07/03/1998	\$12,347.53	09/04/1998	5
2	01/01/1999	Friday Jan 01 1999	03/02/1999	\$8,005.91	05/01/1999	1
3	09/09/1999	Thursday Sep 09 1999	11/08/1999	\$23,454.84	01/09/2000	9

### Using the System Calendar

The screenshot shows the Nexus Query Chameleon interface. The 'Systems' pane on the left lists various databases, with 'Teradata' selected. The main query editor displays the following SQL code:

```

1 Select Calendar_Date, Day_Of_Week, Day_Of_Month
2      ,Day_Of_Year, Day_Of_Calendar, Weekday_Of_Month
3      ,Week_Of_Month, Week_Of_Year, Week_Of_Calendar
4      ,Month_Of_Quarter, Month_Of_Year, Month_Of_Calendar
5      ,Quarter_Of_Year, Quarter_Of_Calendar, Year_Of_Calendar
6 FROM Sys_Calendar.Calendar
7 WHERE Calendar_Date = '1959-01-01'

```

The 'Results' pane at the bottom shows the output of the query:

	calendar_date	day_of_week	day_of_month	day_of_year	day_of_calendar	weekday_of_n
1	01/01/1959	5	1	1		21550

Using the System Calendar in a Query

The screenshot shows the Nexus Query Chameleon interface. The 'Systems' pane on the left lists various databases, with 'Teradata' selected. The main query editor displays the following SQL code:

```

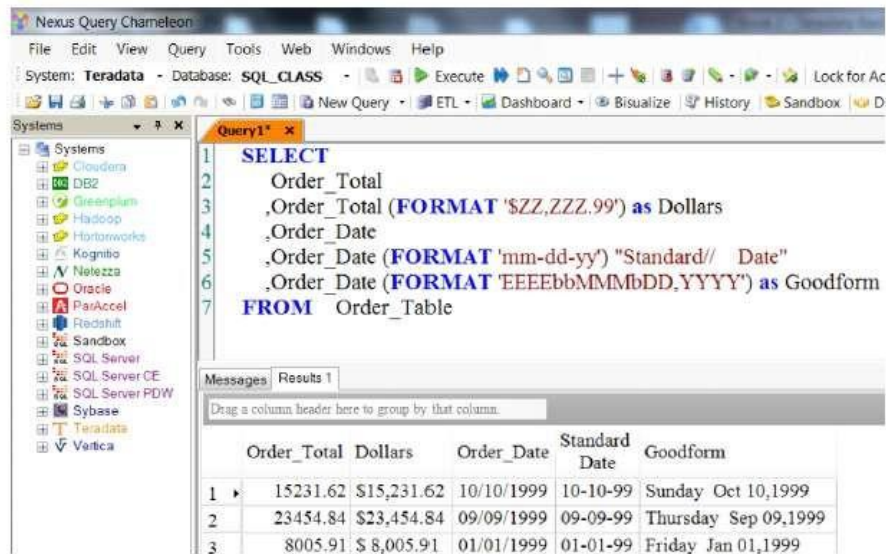
1 SELECT O.*
2 FROM Order_Table as O
3 INNER JOIN
4      Sys_Calendar.Calendar
5 ON Order_Date = Calendar_Date
6 AND Quarter_Of_Year = 4
7 AND Day_of_Week = 6
8 AND Week_of_Month = 0;

```

The 'Results' pane at the bottom shows the output of the query:

	Order_Number	Customer_Number	Order_Date	Order_Total
1	123552	31323134	10/01/1999	5111.47

Formatting Data



The screenshot shows the Nexus Query Chameleon interface. The query editor displays the following SQL code:

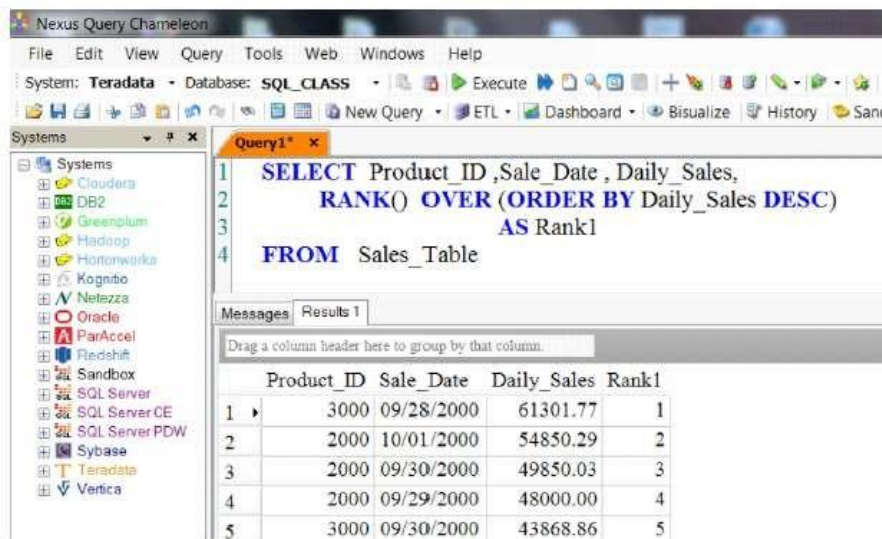
```

1 SELECT
2   Order_Total
3   ,Order_Total (FORMAT '$ZZ,ZZZ.99') as Dollars
4   ,Order_Date
5   ,Order_Date (FORMAT 'mm-dd-yy') "Standard// Date"
6   ,Order_Date (FORMAT 'EEEEbbMMbDD,YYYY') as Goodform
7 FROM Order_Table
  
```

The results pane shows the following data:

	Order_Total	Dollars	Order_Date	Standard Date	Goodform
1	15231.62	\$15,231.62	10/10/1999	10-10-99	Sunday Oct 10,1999
2	23454.84	\$23,454.84	09/09/1999	09-09-99	Thursday Sep 09,1999
3	8005.91	\$ 8,005.91	01/01/1999	01-01-99	Friday Jan 01,1999

### Using Rank



The screenshot shows the Nexus Query Chameleon interface. The query editor displays the following SQL code:

```

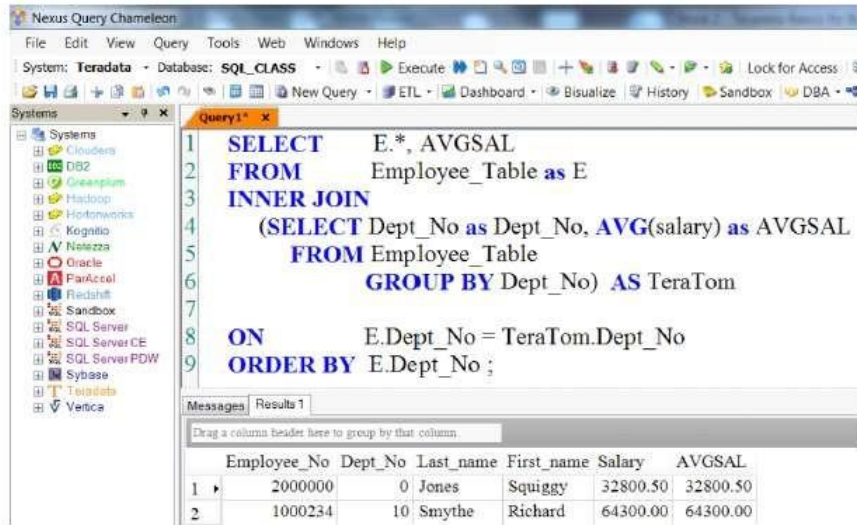
1 SELECT Product_ID,Sale_Date , Daily_Sales,
2        RANK() OVER (ORDER BY Daily_Sales DESC)
3        AS Rank1
4 FROM Sales_Table
  
```

The results pane shows the following data:

	Product_ID	Sale_Date	Daily_Sales	Rank1
1	3000	09/28/2000	61301.77	1
2	2000	10/01/2000	54850.29	2
3	2000	09/30/2000	49850.03	3
4	2000	09/29/2000	48000.00	4
5	3000	09/30/2000	43868.86	5

### Using a Derived Table





Nexus Query Chameleon interface showing a SQL query:

```

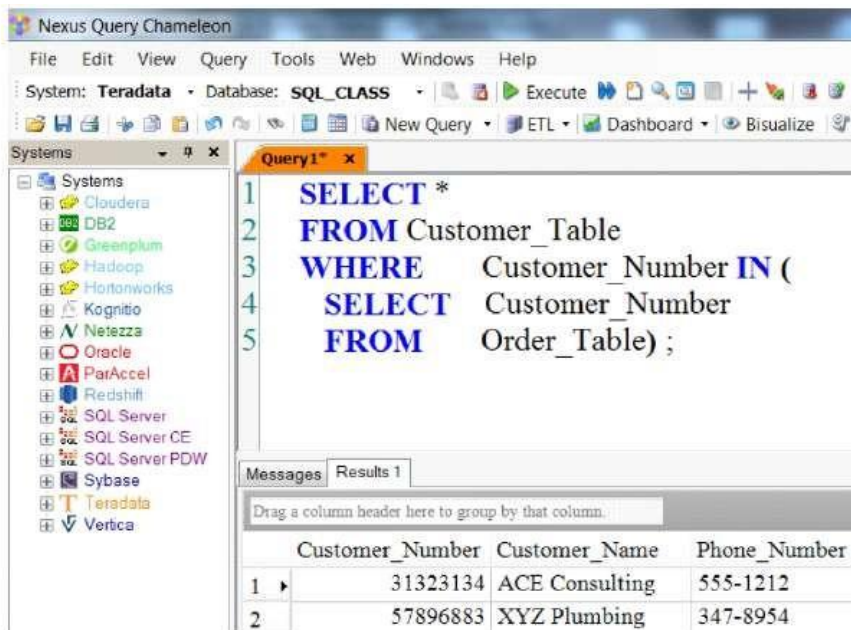
1 SELECT E.*, AVGSAL
2 FROM Employee_Table as E
3 INNER JOIN
4 (SELECT Dept_No as Dept_No, AVG(salary) as AVGSAL
5 FROM Employee_Table
6 GROUP BY Dept_No) AS TeraTom
7
8 ON E.Dept_No = TeraTom.Dept_No
9 ORDER BY E.Dept_No ;

```

Results 1:

	Employee_No	Dept_No	Last_name	First_name	Salary	AVGSAL
1	2000000	0	Jones	Squiggy	32800.50	32800.50
2	1000234	10	Smythe	Richard	64300.00	64300.00

### Using a Subquery



Nexus Query Chameleon interface showing a SQL query:

```

1 SELECT *
2 FROM Customer_Table
3 WHERE Customer_Number IN (
4 SELECT Customer_Number
5 FROM Order_Table) ;

```

Results 1:

	Customer_Number	Customer_Name	Phone_Number
1	31323134	ACE Consulting	555-1212
2	57896883	XYZ Plumbing	347-8954

### Correlated Subquery

The screenshot shows the Nexus Query Chameleon interface. The 'Systems' pane on the left lists various databases, with Teradata selected. The main query editor displays the following SQL code:

```

1 SELECT *
2 FROM Employee_Table as EE
3 WHERE Salary > (
4     SELECT AVG(Salary)
5     FROM Employee_Table as EEEE
6     WHERE EE.Dept_No = EEEE.Dept_No);

```

The 'Results' pane shows the output of the query:

	Employee_No	Dept_No	Last_name	First_name	Salary
1	1333454	200	Smith	John	48000.00
2	1256349	400	Harrison	Herbert	54500.00
3	1121334	400	Strickling	Cletus	54500.00

### Using Substring

The screenshot shows the Nexus Query Chameleon interface. The 'Systems' pane on the left lists various databases, with Teradata selected. The main query editor displays the following SQL code:

```

1 SELECT First Name,
2 SUBSTRING (First_Name FROM 2 for 3) AS Quiz
3 FROM Employee_Table;
4

```

The 'Results' pane shows the output of the query:

	First_name	Quiz
1	Squiggy	qui
2	John	ohn
3	Richard	ich
4	Herbert	erb
5	Mandee	and

### Basic CASE Statement

The screenshot shows the Nexus Query Chameleon interface. The query editor displays the following SQL code:

```

1 SELECT Course_Name, credits
2 ,CASE Credits
3     WHEN 1 THEN 'One Credit'
4     WHEN 2 THEN 'Two Credits'
5     WHEN 3 THEN 'Three Credits'
6     Else 'Credits not found'
7 END AS CreditAlias
8 FROM Course_Table ORDER BY 1 ;

```

The Results pane shows the following data:

Course_name	Credits	CreditAlias
1 Advanced SQL	3	Three Credits
2 Database Administration	4	Credits not found
3 Introduction to SQL	3	Three Credits
4 Physical Database Design	4	Credits not found

### Advanced CASE Statement

The screenshot shows the Nexus Query Chameleon interface. The query editor displays the following SQL code:

```

1 SELECT Course_Name
2 ,CASE
3     WHEN Credits <= 1 THEN 'One'
4     WHEN Credits = 2 THEN 'Two'
5     WHEN Credits < 4 THEN 'Three'
6     WHEN Course_Name like 'Tera%' Then '4'
7     Else 'Don't know'
8 END AS CreditAlias
9 FROM Course_Table
10 ORDER BY 1;

```

The Results pane shows the following data:

Course_name	CreditAlias
1 Advanced SQL	Three
2 Database Administration	Don't know
3 Introduction to SQL	Three

### Using an Access Lock in your SQL

The screenshot shows the Nexus Query Chameleon interface. The 'Systems' pane on the left lists various databases, with Teradata selected. The main query editor displays the following SQL:

```
1 Locking Row For ACCESS
2 SELECT * FROM Employee_Table
```

The 'Results' tab shows a table with 6 rows and 6 columns: Employee\_No, Dept\_No, Last\_name, First\_name, and Salary. The data is as follows:

	Employee_No	Dept_No	Last_name	First_name	Salary
1	2000000	0	Jones	Squiggy	32800.50
2	1333454	200	Smith	John	48000.00
3	1000234	10	Smythe	Richard	64300.00
4	1256349	400	Harrison	Herbert	54500.00
5	1232578	100	Chambers	Mandee	48850.00
6	1121334	400	Strickling	Cletus	54500.00

#### Collect Statistics

The screenshot shows the Nexus Query Chameleon interface with the following SQL commands entered in the query editor:

```
1 COLLECT STATISTICS
2 COLUMN (Employee_No)
3 ON Employee_Table;
4
5 COLLECT STATISTICS
6 ON Employee_Table
7 COLUMN Dept_No;
```

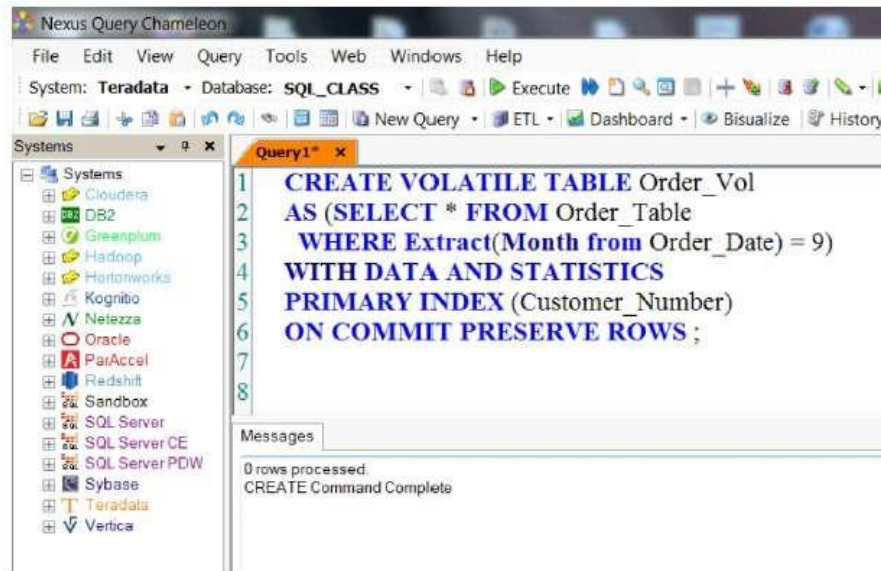
The 'Messages' pane at the bottom shows the following output:

```
1 rows processed.
COLLECT STATISTICS Command Complete

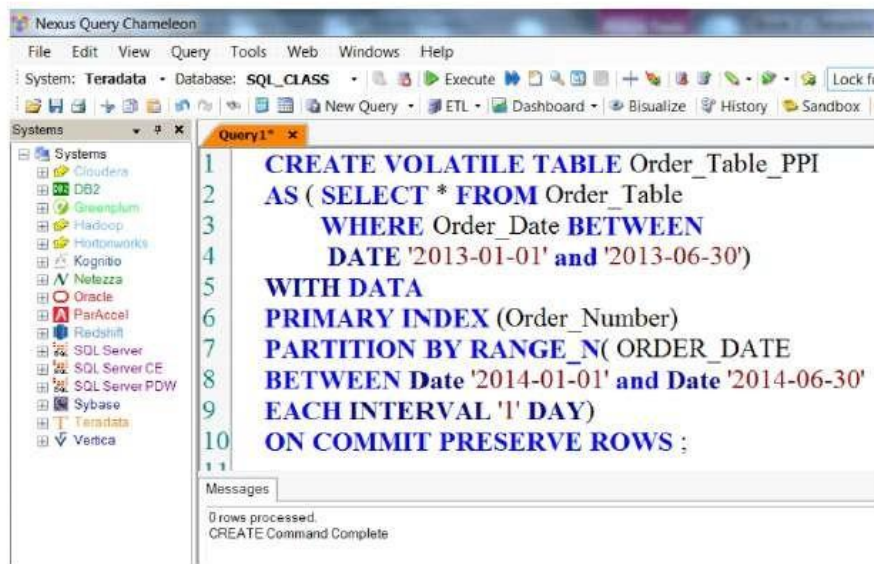
1 rows processed.
COLLECT STATISTICS Command Complete
```

#### CREATING a Volatile Table with a Primary Index

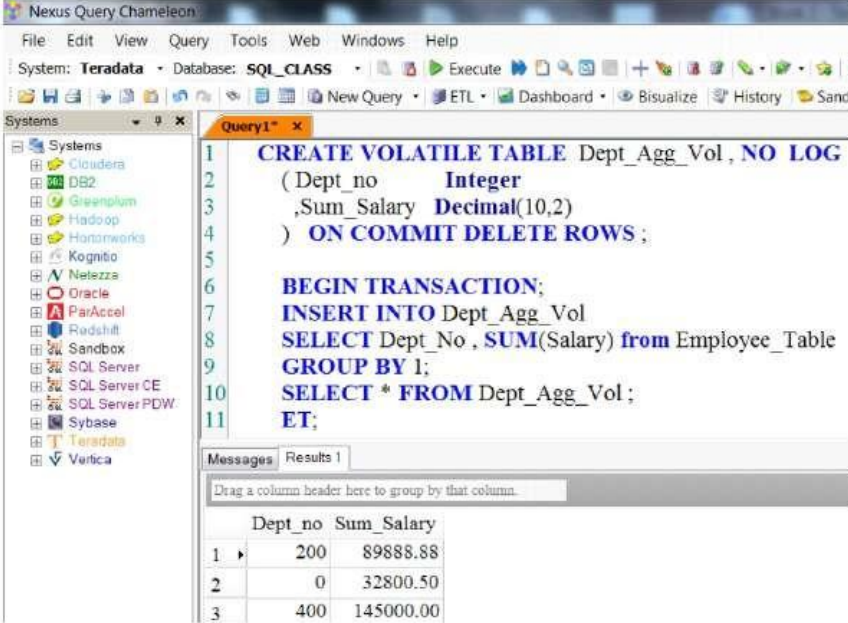




CREATING a Volatile Table that is Partitioned (PPI)



CREATING a Volatile Table that is deleted after the Query



The screenshot shows the Nexus Query Chameleon interface. The left pane lists various database systems, with Teradata selected. The main pane displays a SQL query for Query1:

```

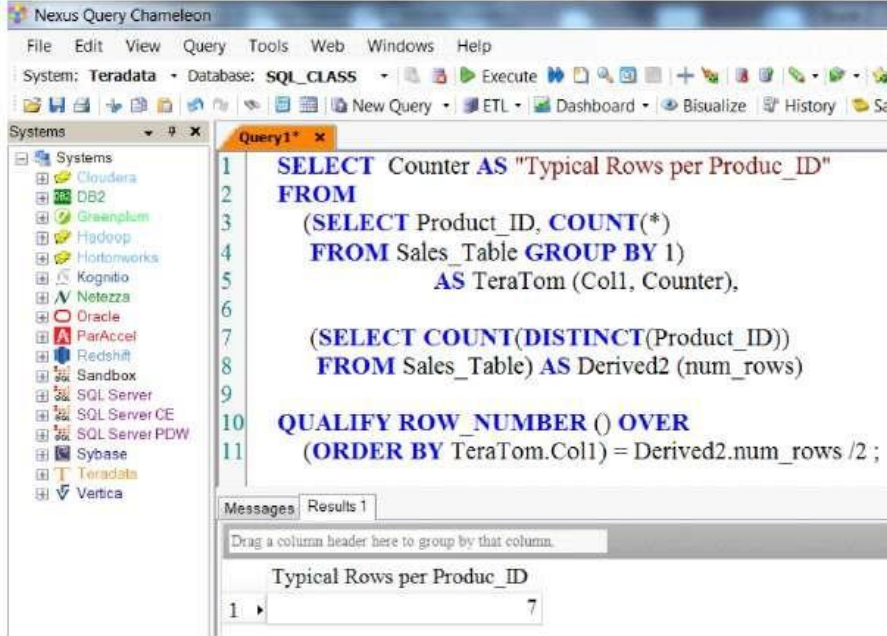
1 CREATE VOLATILE TABLE Dept_Agg_Vol, NO LOG
2 ( Dept_no      Integer
3   ,Sum_Salary  Decimal(10,2)
4   ) ON COMMIT DELETE ROWS ;
5
6 BEGIN TRANSACTION;
7 INSERT INTO Dept_Agg_Vol
8 SELECT Dept_No , SUM(Salary) from Employee_Table
9 GROUP BY 1;
10 SELECT * FROM Dept_Agg_Vol ;
11 ET;

```

Below the query editor, the 'Results 1' tab shows a table with the following data:

Dept_no	Sum_Salary
1	200 89888.88
2	0 32800.50
3	400 145000.00

Finding the Typical Rows per Value for specific column



The screenshot shows the Nexus Query Chameleon interface. The left pane lists various database systems, with Teradata selected. The main pane displays a SQL query for Query1:

```

1 SELECT Counter AS "Typical Rows per Produc_ID"
2 FROM
3 (SELECT Product_ID, COUNT(*)
4  FROM Sales_Table GROUP BY 1)
5  AS TeraTom (Col1, Counter),
6
7 (SELECT COUNT(DISTINCT(Product_ID))
8  FROM Sales_Table) AS Derived2 (num_rows)
9
10 QUALIFY ROW_NUMBER () OVER
11 (ORDER BY TeraTom.Col1) = Derived2.num_rows /2 ;

```

Below the query editor, the 'Results 1' tab shows a table with the following data:

Typical Rows per Produc_ID	
1	7

Finding out how much Space you have

The screenshot shows the Nexus Query Chameleon interface. The query editor contains the following SQL:

```

1 SELECT UserName
2      , CreatorName
3      , PermSpace
4      , SpoolSpace
5      , TempSpace
6      , LastAlterTimeStamp
7 FROM DBC.Users
8 WHERE UserName = USER ;

```

The results pane shows a single row of data for the user 'DBC':

UserName	CreatorName	PermSpace	SpoolSpace	TempSpace	LastAlter
1 DBC	DBC	2711469682.56	3885663682.56	3885663682.56	02/08/201

How much Space you have Per AMP

The screenshot shows the Nexus Query Chameleon interface with a query to find space per AMP:

```

1 LOCKING ROW FOR ACCESS
2 SELECT Vproc          AS "AMP# "
3      , DatabaseName   AS "Database"
4      , AccountName    AS "Account"
5      , (MaxPerm)       AS "Perm"
6      , (MaxSpool)      AS "Spool"
7      , (MaxTemp)       AS "Temp"
8 FROM DBC.DiskSpace
9 WHERE DatabaseName = 'sql01';

```

The results pane shows two rows of data:

AMP#	Database	Account	Perm	Spool	Temp
1 0	SQL01	DBC	2500000.00	10000000.00	500000.00
2 1	SQL01	DBC	2500000.00	10000000.00	500000.00

Finding your Space



The screenshot shows the Nexus Query Chameleon interface. The left pane displays a tree of systems, including Teradata. The main query editor shows the following SQL:

```

1 LOCKING ROW FOR ACCESS
2 SELECT SUM(MaxPerm) AS "Perm"
3       ,SUM(MaxSpool) AS "Spool"
4       ,SUM(MaxTemp) AS "Temp"
5 FROM DBC.DiskSpace
6 WHERE DatabaseName = USER;

```

The Messages pane shows a warning: "Drag a column header here to group by that column." The Results pane displays the following data:

	Perm	Spool	Temp
1	2711469682.56	3885663682.56	3885663682.56

### Finding Space Skew in Tables in a Database

The screenshot shows the Nexus Query Chameleon interface. The left pane displays a tree of systems, including Teradata. The main query editor shows the following SQL:

```

1 SELECT
2   Vproc
3   ,CAST (TableName AS CHAR(20))
4   ,CurrentPerm
5   ,PeakPerm
6 FROM DBC.TableSizeV
7 WHERE DatabaseName = 'SQL_Class'
8 ORDER BY TableName, Vproc ;
9

```

The Messages pane shows a warning: "Drag a column header here to group by that column." The Results pane displays the following data:

	Vproc	TableName	CurrentPerm	PeakPerm
1	0	Addresses	1536.00	1536.00
2	1	Addresses	1024.00	1024.00
3	0	AdvInsTLC	17920.00	17920.00
4	1	AdvInsTLC	17920.00	17920.00

### Finding the Number of rows per AMP for a Column



The screenshot shows the Nexus Query Chameleon interface. The 'Systems' pane on the left lists various databases, with Teradata selected. The main query editor displays the following SQL code:

```

1 LOCKING ROW FOR ACCESS SELECT
2   HASHAMP (HASHBUCKET
3     (HASHROW (Employee_No))) AS "AMP #"
4   ,COUNT(*)
5 FROM SQL_Class.Employee_Table
6 GROUP BY 1
7 ORDER BY 1 ;

```

The 'Results' pane shows the output of the query:

AMP #	Count(*)
1	3
2	6

### Finding Account Information

The screenshot shows the Nexus Query Chameleon interface. The 'Systems' pane on the left lists various databases, with Teradata selected. The main query editor displays the following SQL code:

```

1 SELECT *
2 FROM DBC.AccountInfoV
3 ORDER BY 1;

```

The 'Results' pane shows the output of the query:

UserName	AccountName	UserOrProfile
AndreProfile	\$MSQL12	Profile
CoffingTest	DBC	User
Compressed	\$MComp	User
console	\$H-remote-console-use	User
Crashdumps	Crashdumps	User

### Ordered Analytics

Nexus Query Chameleon

File Edit View Query Tools Web Windows Help

System: Teradata Database: DBC

Execute Lock for Access

New Query ETL Dashboard Visualize History Sandbox DBA Joins

Systems

- Cloudera
- DB2
- Greenplum
- Hadoop
- Hortonworks
- Kognito
- Netezza
- Oracle
- PostgreSQL
- Redshift
- SQL Server
- SQL Server CE
- SQL Server PDW
- Sybase
- Teradata
- Vertica

Query1 Super Join Builder (BETA) Join Builder

```
1 SELECT Product_ID, Sale_Date, Daily_Sales,  
2 SUM(Daily_Sales) OVER ( ORDER BY Product_ID, Sale_Date  
3 ROWS UNBOUNDED PRECEDING) "CSUM",  
4 SUM(Daily_Sales) OVER ( ORDER BY Product_ID, Sale_Date  
5 ROWS 2 PRECEDING) "MSUM",  
6 AVG(Daily_Sales) OVER ( ORDER BY Product_ID, Sale_Date  
7 ROWS 2 PRECEDING) "MAVG",  
8 Daily_Sales - SUM(Daily_Sales) OVER  
9 ( ORDER BY Product_ID, Sale_Date  
10 ROWS BETWEEN 2 PRECEDING AND 2 PRECEDING)  
11 AS "MDIFF"  
12 FROM SQL_CLASS.Sales_table ;
```

Messages Results 1

Drag a column header here to group by that column

	Product_ID	Sale_Date	Daily_Sales	CSUM	MSUM	MAVG	MDIFF
1	2000	09/28/2000	41888.88	41888.88	41888.88	41888.88	?
2	2000	09/29/2000	48000.00	89888.88	89888.88	44944.44	?
3	2000	09/30/2000	49850.03	139738.91	139738.91	46579.64	7961.15
4	2000	10/01/2000	54850.29	194589.20	152700.32	50900.11	6850.29

# View Functions

## View Functions

### Overview

"It is easier to go down a hill than up it, but the view is much better at the top."  
- Arnold Bennett

### Creating a Simple View

Employee_Table				
Employee_No	Dept_No	Last_Name	First_Name	Salary
2000000	?	Jones	Squiggy	32800.50
1000234	10	Smythe	Richard	32800.00
1232578	100	Chambers	Mandee	48850.00
1324657	200	Coffing	Billy	41888.88
1333454	200	Smith	John	48000.00
2312225	300	Larkins	Loraine	40200.00
1256349	400	Harrison	Herbert	54500.00
2341218	400	Reilly	William	36000.00
1121334	400	Strickling	Cletus	54500.00

```
CREATE View Employee_V AS
SELECT      Employee_No
           ,First_Name
           ,Last_Name
           ,Dept_No
FROM Employee_Table ;
```

The purposes of views are to restrict access to certain columns, derive columns or Join Tables, and to restrict access to certain rows (if a WHERE clause is used).

### Basic Rules for Views

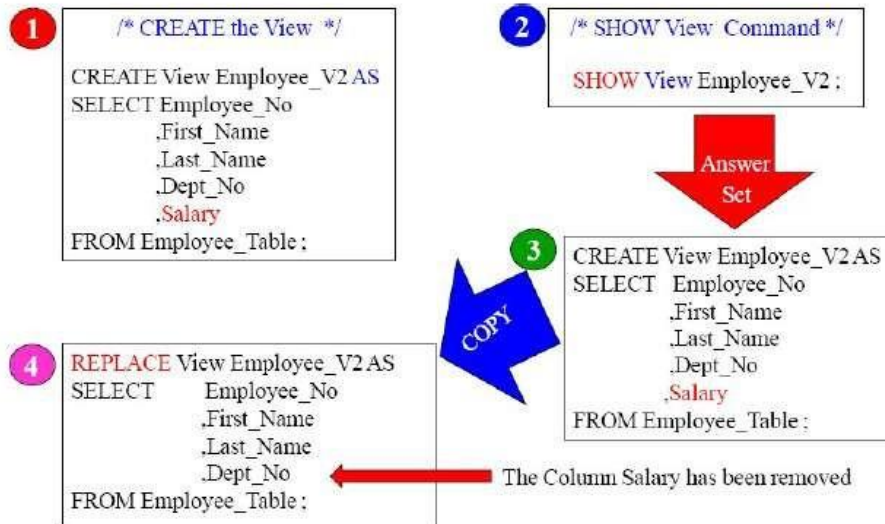
1. No **ORDER BY** inside the View CREATE (some exceptions exist)
2. All **Aggregation** needs to have an **ALIAS**
3. Any **Derived** columns (such as Math) needs an **ALIAS**

<pre>CREATE View Department_Salaries AS SELECT      Dept_No            ,SUM(Salary) as SumSal            ,SUM(Salary) / 12 as MonthSal FROM Employee_Table GROUP BY 1;</pre>	<pre>SEL * FROM Department_Salaries Order By 1 ;</pre>
--	--

Dept_No	SumSal	MonthSal
?	32800.50	2733.38
10	64300.00	5358.33
100	48850.00	4070.83
200	89888.88	7490.74
300	40200.00	3350.00
400	145000.00	12083.33

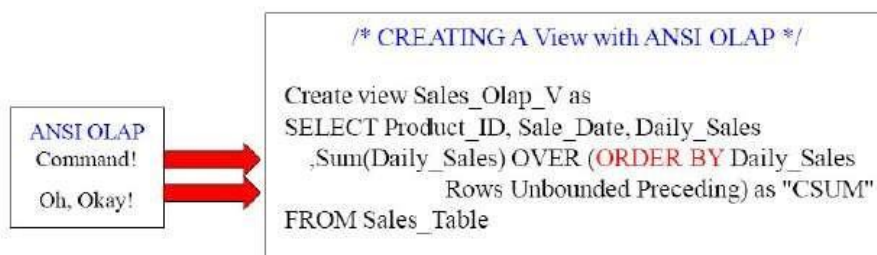
Above, are the basic rules of Views with excellent examples.

### How to Modify a View



The REPLACE Keyword will allow a user to change a view.

#### Exceptions to the ORDER BY Rule inside a View



There are EXCEPTIONS to the ORDER BY rule. The TOP command allows a view to work with an ORDER BY inside. ANSI OLAP statements also work inside a View.

#### How to Get HELP with a View

1

HELP View Command

HELP View Emp\_View ;

Column Name	Type	Comment	Nullable	Format	Title
Employee_No	?		?	?	?
Dept_No	?		?	?	?
Last_Name	?		?	?	?
First_Name	?		?	?	?
Salary	?		?	?	?
Max Length	Decimal Total Digits	Decimal Fractional Digits	Range Low	Range High	
?	?	?	?	?	?
?	?	?	?	?	?
?	?	?	?	?	?
?	?	?	?	?	?
UpperCase	Table/View?	Default value	Char Type	IdCol Type	
?	?	?	?	?	?
?	?	?	?	?	?
?	?	?	?	?	?
?	?	?	?	?	?

The Help View command does little but show you the columns.

### Views sometimes CREATED for Formatting or Row Security

```
CREATE VIEW empl_200_v AS
  SELECT Employee_No AS Emp_No
         ,Last_Name   AS Last
         ,salary/12 (format '$$$$,$$9.99') AS Monthly_Salary
  FROM   Employee_Table
 WHERE  Dept_No = 200 ;
```

SELECTING from A View

```
SELECT *
FROM Empl_200_v
ORDER BY Monthly_Salary ;
```

Emp_No	Last_Name	Monthly_Salary
1324657	Coffing	\$3,490.74
1333454	Smith	\$4,000.00

Views are designed to do many things. In the example above, this view formats and derives data, limits columns, and also limits the rows coming back with a WHERE.

### Another Way to Alias Columns in a View CREATE

```
CREATE VIEW empl_200_v (Emp_Nbr, Last, Monthly_Salary)
  AS SELECT Employee_No
         ,Last_Name
         ,Salary/12 (format '$$$$,$$9.99')
  FROM   Employee_Table
 WHERE  Dept_No = 200 ;
```

Aliases  
are here



```
SELECT *
FROM Empl_200_v
ORDER BY Monthly_Salary ;
```

Emp_No	Last_Name	Monthly_Salary
1324657	Coffing	\$3,490.74
1333454	Smith	\$4,000.00

Will this View CREATE Error? No! It won't error because it's Aliased above!

### Resolving Aliasing Problems in a View CREATE

```
CREATE VIEW empl_200_v (Emp_Nbr, Last, Monthly_Salary)
AS SELECT Employee_No
      ,Last_Name
      ,Salary/12 (format '$$$$,$$9.99') as Sal_Monthly
FROM Employee_Table
WHERE Dept_No = 200 ;
```

```
SELECT *
FROM Empl_200_v
ORDER BY 3 ;
```

Emp_No	Last_Name	Monthly_Salary
1324657	Coffing	\$3,490.74
1333454	Smith	\$4,000.00

← First Alias!

The ALIAS for Salary / 12 that'll be used in this example is MONTHLY\_SALARY. It came first at the top, even though it is aliased in the SELECT list also.

### Resolving Aliasing Problems in a View CREATE

```
CREATE VIEW empl_200_v (Emp_Nbr, Last, Monthly_Salary)
AS SELECT Employee_No
      ,Last_Name
      ,Salary/12 (format '$$$$,$$9.99') as Sal_Monthly
FROM Employee_Table
WHERE Dept_No = 200 ;
```

```
SELECT *
FROM Empl_200_v
ORDER BY Sal_Monthly ;
```

What will happen in the above query?

### Resolving Aliasing Problems in a View CREATE

```
CREATE VIEW empl_200_v (Emp_Nbr, Last, Monthly_Salary)
AS SELECT Employee_No
      ,Last_Name
      ,Salary/12 (format '$$$$,$$9.99') as Sal_Monthly
FROM Employee_Table
```

```
WHERE Dept No = 200 ;
```

```
SELECT *
FROM Empl_200_v
ORDER BY Sal_Monthly ;
```

Doesn't  
Recognize

ERROR

If you ALIAS at the top, then that is the only ALIAS that the query can recognize. So, it is a good idea to alias at the top or the bottom, but not do both.

### CREATING Views for Complex SQL such as Joins

```
CREATE VIEW Customer_Order_v AS
SELECT Customer_Name AS Customer
,Order_Number
,Order_Total (FORMAT '$$$,$$9.99' ) AS Total_Amount
FROM Customer_Table AS Cust
,Order_Table AS Ord
WHERE Cust.Customer_Number = Ord.Customer_Number ;
```

```
SELECT * FROM Customer_Order_v
ORDER BY 1 ;
```

Customer	Order_Number	Total_Amount
Ace Consulting	123552	\$5,111.47
Billy's Best Choice	123456	\$12,346.53
Billy's Best Choice	123512	\$8,005.91
Databases N-U	123585	\$15,231.62
XYZ Plumbing	123777	\$23,454.84

A huge reason for Views, other than security, is to make Complex SQL easy for users. This view already has the Inner Join built into it, but users just SELECT.

### WHY certain columns need Aliasing in a View

```
CREATE VIEW Aggreg_Order_v AS
SELECT Customer_Number
,Order_Date/100+190000 (format '9999-99') AS Yr_Mth_Orders
,COUNT(Order_Total) AS Order_Cnt
,SUM(Order_Total) AS Order_Sum
,AVG(Order_Total) AS Order_Avg
FROM Order_Table
GROUP BY Customer_Number, Yr_Mth_Orders ;
```

```
SELECT Customer_Number
,Order_Sum
FROM Aggreg_Order_v ;
```

Customer_Number	Order_Sum
31323134	5111.47
87323456	15231.62
11111111	8005.91
11111111	12347.53
57896883	23454.84

When you CREATE a view, you have to ALIAS any aggregation or derived data (such as math). Why? So you can SELECT it later without having to do a SELECT \*. Here, we only chose two columns and used their ALIAS to retrieve them.



## Aggregates on View Aggregates

```
CREATE VIEW Aggreg_Order_v AS
SELECT Customer_Number
      ,Order_Date/100+190000 (format '9999-99') AS Yr_Mth_Orders
      ,COUNT(Order_Total) AS Order_Cnt
      ,SUM(Order_Total) AS Order_Sum
      ,AVG(Order_Total) AS Order_Avg
FROM Order_Table
GROUP BY Customer_Number, Yr_Mth_Orders ;
```

```
SELECT Customer_Number
      ,Order_Sum
FROM Aggreg_Order_v ;
```

```
SELECT SUM (Order_Sum)
FROM Aggreg_Order_v ;
```

Customer_Number	Order_Sum
31323134	5111.47
87323456	15231.62
11111111	8005.91
11111111	12347.53
57896883	23454.84

SUM(Order_Sum)
64151.37

The examples above show how we put a SUM on the aggregate Order\_Sum.

## Locking Row for Access

Lock  
Placed



```
CREATE VIEW Emp_HR_v AS
Locking Row for ACCESS
SELECT Employee_No
      ,Dept_No
      ,Last_Name
      ,First_Name
FROM Employee_Table ;
```

```
SELECT * FROM Emp_HR_v;
```

The Employee\_Table used above will automatically use an **ACCESS Lock**, which allows **ACCESS** during **UPDATES** or **table loads**.

Most views utilize the Locking row for ACCESS command. This is because they want to be able to read while a table is being updated and loaded into. If the user knows a dirty read won't have a huge effect on their job, why not make a view lock with an ACCESS Lock, thus preventing unnecessary waiting?

## Creating Views for Temporal Tables

```
CREATE VIEW SQL01.Prop_As_Is
AS
Locking row for access
CURRENT VALIDTIME
SELECT Cust_No
      ,Prop_No
BEGIN(Prop_Val_Time) AS Beg_Val_Time,

CREATE VIEW SQL01.Prop_As_Was
AS
Locking row for access
NONSEQUENCED VALIDTIME
SELECT Cust_No
      ,Prop_No
BEGIN(Prop_Val_Time) AS Beg_Val_Time,
```

```
END(Prop_Val_Time) AS End_Val_Time,  
FROM Property_Owners;
```

```
END(Prop_Val_Time) AS End_Val_Time,  
FROM Property_Owners;
```

```
SELECT * FROM SQL01.Prop As Is ;
```

```
SELECT * FROM SQL01.Prop As Was ;
```

You can create views that will allow users to see the way things are or the way things were. Above, are two excellent examples

### Altering a Table

```
CREATE VIEW Emp_HR_v AS  
SELECT Employee_No  
      ,Dept_No  
      ,Last_Name  
      ,First_Name  
FROM Employee_Table ;
```

-- Altering the actual Table

```
ALTER TABLE Employee_Table  
ADD Mgr_No INTEGER ;
```



Will the View  
STILL run?

-- Select from the View

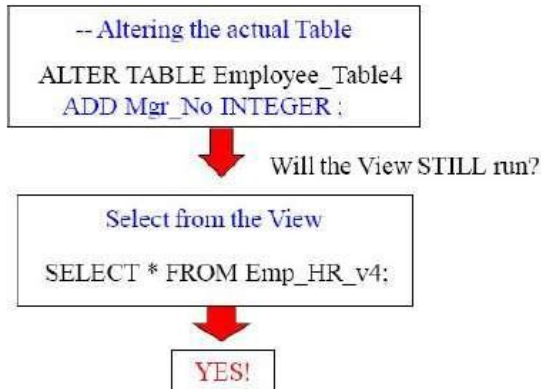
```
SELECT * FROM Emp_HR_v;
```



This view will run after the table has added an additional column!

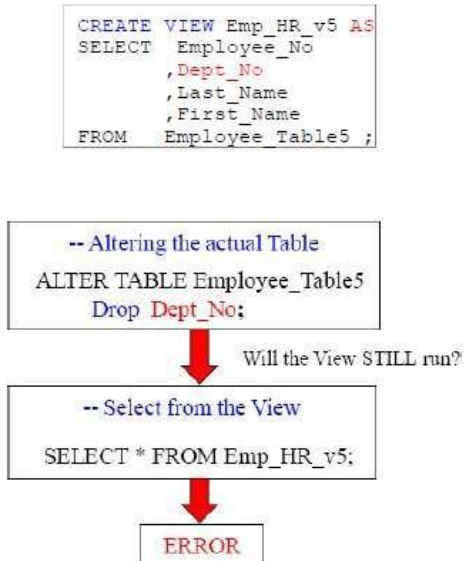
### Altering a Table after a View has been created

```
CREATE VIEW Emp_HR_v4 AS  
SELECT *  
FROM Employee_Table4 ;
```



This view runs after the table has added an additional column, but it won't include Mgr\_No in the view results even though there is a SELECT \* in the view. The View includes only the columns present when the view was CREATED.

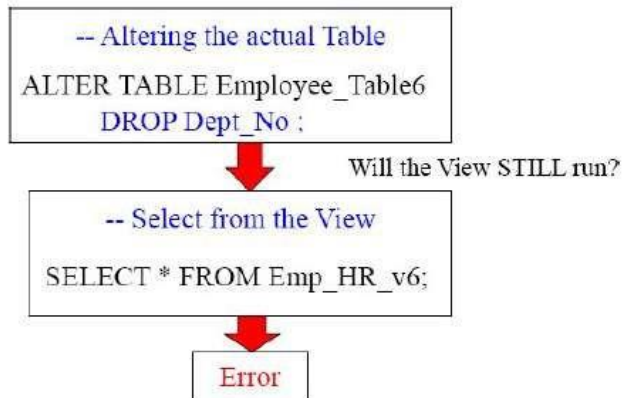
#### A View that errors After an ALTER



This view will NOT run after the table has dropped a column referenced in the view.

#### Troubleshooting a View

```
CREATE VIEW Emp_HR_v6 AS  
SELECT *  
FROM Employee_Table6 ;
```



This view will NOT run after the table has dropped a column referenced in the view, even though the View was CREATED with a SELECT \*. At View CREATE Time, the columns present were the only ones the view considered responsible for. Dept\_No was one of those columns. Once Dept\_No was dropped, the view no longer works.

#### Updating Data in a Table through a View

```

CREATE VIEW Emp_HR_v8 AS
SELECT *
FROM Employee_Table8;

```

```

--Updating the table through the View

UPDATE Emp_HR_v8
SET Salary = 88888.88
WHERE Employee No = 2000000;

```

```

--SELECT from the actual Table

SELECT *
FROM Employee_Table8
WHERE Employee No = 2000000;

```

Employee_No	Dept_No	Last_Name	First_Name	Salary
2000000	?	Jones	Squiggy	88888.88

You can UPDATE a table through a View if you have the RIGHTS to do so.

#### Maintenance Restrictions on a Table through a View

There are a few restrictions that disallow maintenance activity on a view with an INSERT, UPDATE or DELETE request. A view cannot be used for maintenance if it:

1. Performs a join operation - more than one table
2. Selects the same column twice - wouldn't know which one to use
3. Derives data - because it does not undo the math or calculation
4. Performs aggregation - because this eliminates detail data
5. Uses OLAP functions - because OLAP data is calculated
6. Uses a DISTINCT or GROUP BY - eliminates duplicate rows

Perform maintenance on a table through a view, but see the restrictions above first.