

Tera-Tom's 1000 page e-Book on Teradata

Teradata Architecture and SQL



This book is for Glen Fernandes use only.
Only Glen Fernandes can view this
document legally.

So easy a 7-year old boy (raised by wolves) could understand it

ISBN 978-0-9859164-0-4

Teradata*, NCR, BYNET and SQL Assistant are registered trademarks of Teradata Corporation, Dayton, Ohio, U.S.A., IBM, DB2 and Netezza are registered trademarks of IBM Corporation, ANSI is a registered trademark of the American National Standards Institute. Microsoft Windows, Windows 2003 Server, .NET, PDW, and SQL Server are trademarks of Microsoft. Ethernet is a trademark of Xerox. UNIX is a trademark of The Open Group. Linux is a trademark of Linus Torvalds. Java and Oracle is a trademark of Oracle. ParAccel is a trademark of ParAccel. Kognitio is a trademark of Kognitio. Greenplum is a trademark of EMC corporation. Nexus Query Chameleon is a trademark of Coffing Data Warehousing.

Coffing Data Warehousing shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book or from the use of programs or program segments that are included. The manual is not a publication of Teradata Corporation, nor was it produced in conjunction with Teradata Corporation.

Copyright © January 2012 by Coffing Publishing

ISBN 978-0-9833363-8-9

All rights reserved. No part of this book shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the publisher. No patent liability is assumed with respect to the use of information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions, neither is any liability assumed for damages resulting from the use of information contained herein.

About Tom Coffing



Tom Coffing, CEO of Coffing Data Warehousing and better known as Tera-Tom has written over 30 books on Teradata and has taught over 1,000 classes on Teradata throughout the United States, Canada, Mexico, Europe, Africa, India and China.

Tom directed Coffing Data Warehousing towards its most challenging transformation by creating the Nexus Query Chameleon, a business user's desktop tool designed for all aspects of Business Intelligence. The Nexus is the first desktop BI tool designed to access all computer vendor's databases including Teradata, Oracle, IBM, Netezza, Greenplum, Kognitio, ParAccel and Microsoft. This changed the computer industry because customers could now utilize multiple vendors in their previously proprietary environments and bring the data together via the Nexus.

Tom can be reached at 513 300-0341 or emailed at Tom.Coffing@CoffingDW.com. His website is CoffingDW.com.

About Warehouse William Coffing



William Coffing is a Teradata Certified Master who has written multiple books on Teradata including the Teradata User's Guide. William, better known as Warehouse William has taught Teradata internationally and throughout the United States and has been an integral part of programming the Nexus Query Chameleon.

William graduated from Eastern Michigan University with a Bachelor's Degree in Creative Writing and Communication/Theater Arts and then spent an additional year at the New York Film Academy. He has written and directed a feature film called "Switching Roles" and has directed multiple commercials.

Table of Contents

- [Chapter 1 - The Teradata Architecture](#)
- [Chapter 2 - Primary Index](#)
- [Chapter 3 - Hashing of the Primary Index](#)
- [Chapter 4 - Space](#)
- [Chapter 5 - Partition Primary Index \(PPI\)](#)
[Tables Subquery](#)
- [Chapter 6 - Secondary Indexes](#)
- [Chapter 7 - Columnar Tables](#)
- [Chapter 8 - Temporal Tables Create Functions](#)
- [Chapter 9 - How Joins work internally](#)
- [Chapter 10 - Join Indexes](#)
- [Chapter 11 - Basic SQL Functions](#)
- [Chapter 12 - The WHERE Clause](#)
- [Chapter 13 - Distinct Vs Group By](#)
- [Chapter 14 - The TOP Command](#)
- [Chapter 15 - Review](#)
- [Chapter 16 - HELP and SHOW](#)
- [Chapter 17 - Aggregation Function](#)
- [Chapter 18 - Join Functions](#)
- [Chapter 19 - Date Functions](#)
- [Chapter 20 - Format Functions](#)
- [Chapter 21 - OLAP Functions](#)
- [Chapter 22 - The Quantile Function](#)
- [Chapter 23 - Temporary Tables](#)
- [Chapter 24 - Sub-query Functions](#)
- [Chapter 25 - Substrings and Positioning Functions](#)
- [Chapter 26 - Interrogating the Data](#)
- [Chapter 27 - View Functions](#)
- [Chapter 28 - Macro Functions](#)
- [Chapter 29 - Set Operators Functions](#)
- [Chapter 30 - Table Create and Data Types \(DML\)](#)
- [Chapter 31 – Data Manipulation Language](#)
- [Chapter 32 - Stored Procedure Functions](#)
- [Chapter 33 - Trigger Functions](#)
- [Chapter 34 - Math Functions](#)
- [Chapter 35 - The SAMPLE Function](#)
- [Chapter 36 - Statistical Aggregate Functions](#)
- [Chapter 37 - Explain](#)
- [Chapter 38 - Collect Statistics](#)
- [Chapter 39 - Hashing Functions](#)
- [Chapter 40 - BTEQ – Batch Teradata Query](#)
- [Chapter 41 - Your Nexus Query Chameleon](#)

Chapter 1

The Teradata

Architecture

“Fall seven times – Stand up eight!”

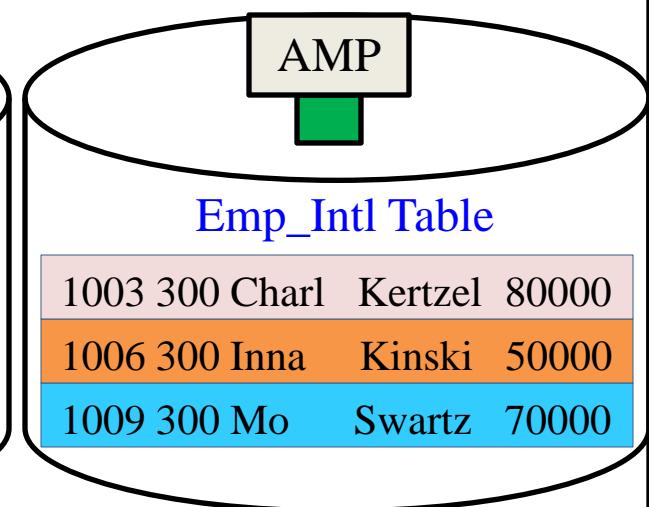
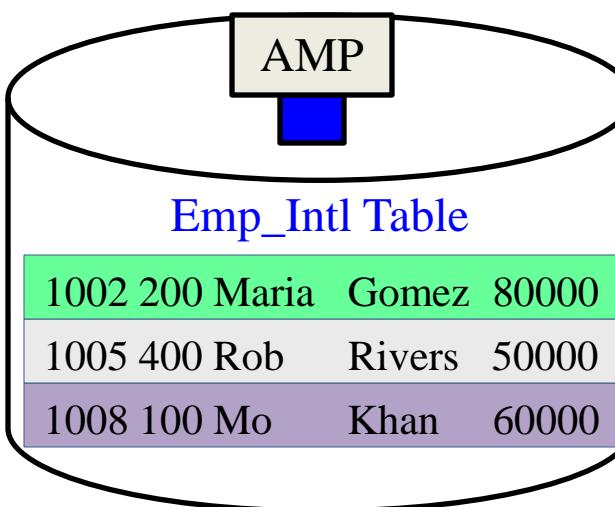
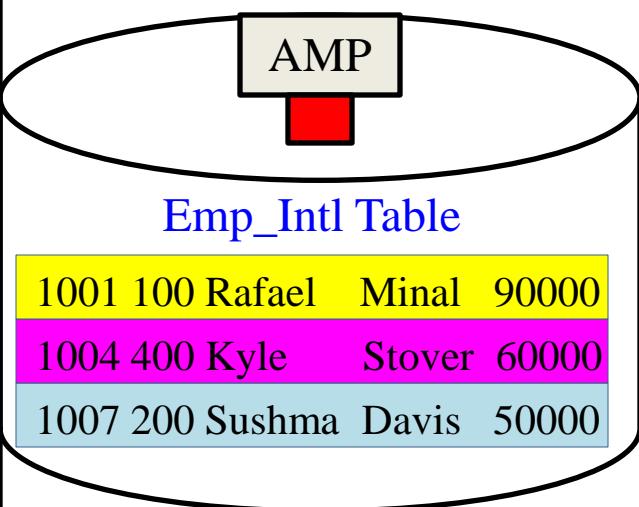
- Japanese Proverb

Table of Contents Chapter 1 – The Teradata Architecture

- [Parallel Processing](#)
- [The Teradata Architecture](#)
- [All Teradata Tables are spread across All AMPs](#)
- [Each Table has a Column that is the Primary Index](#)
- [Each Table has a Column that is the Primary Index](#)
- [A Full Table Scan uses All-AMPs in Parallel](#)
- [Knowing the Primary Index of a Table is Vital](#)
- [Teradata Systems can Add AMPs for Linear Scalability](#)
- [Watch the Video on Teradata Architecture](#)

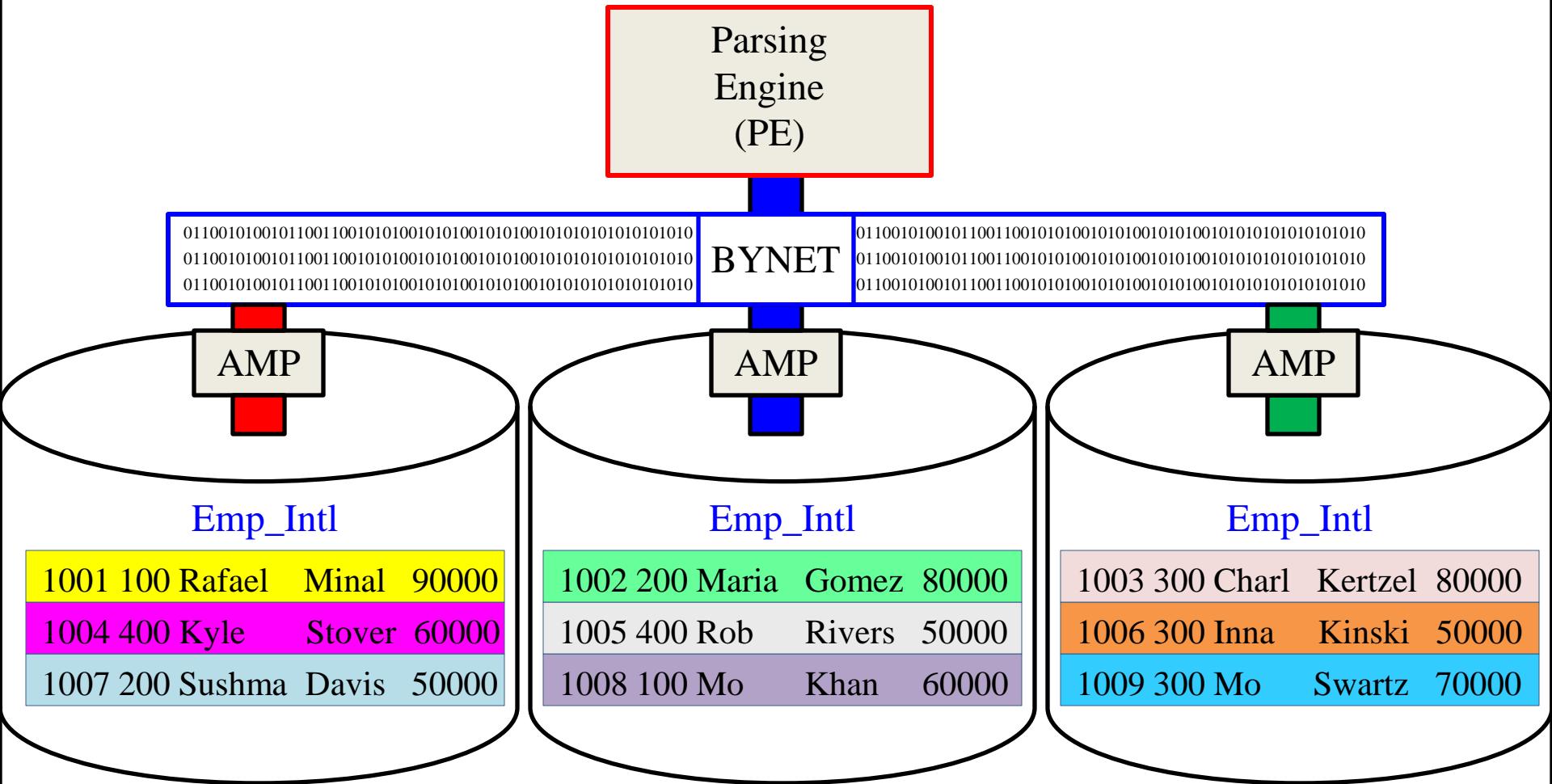
Parallel Processing

Emp_Intl				
Emp_No	Dept_No	First_Name	Last_Name	Salary
1001	100	Rafael	Minal	90000.00
1002	200	Maria	Gomez	80000.00
1003	300	Charl	Kertzel	70000.00
1004	400	Kyle	Stover	60000.00
1005	400	Rob	Rivers	50000.00
1006	300	Inna	Kinski	50000.00
1007	200	Sushma	Davis	50000.00
1008	100	Mo	Khan	60000.00
1009	300	Mo	Swartz	70000.00



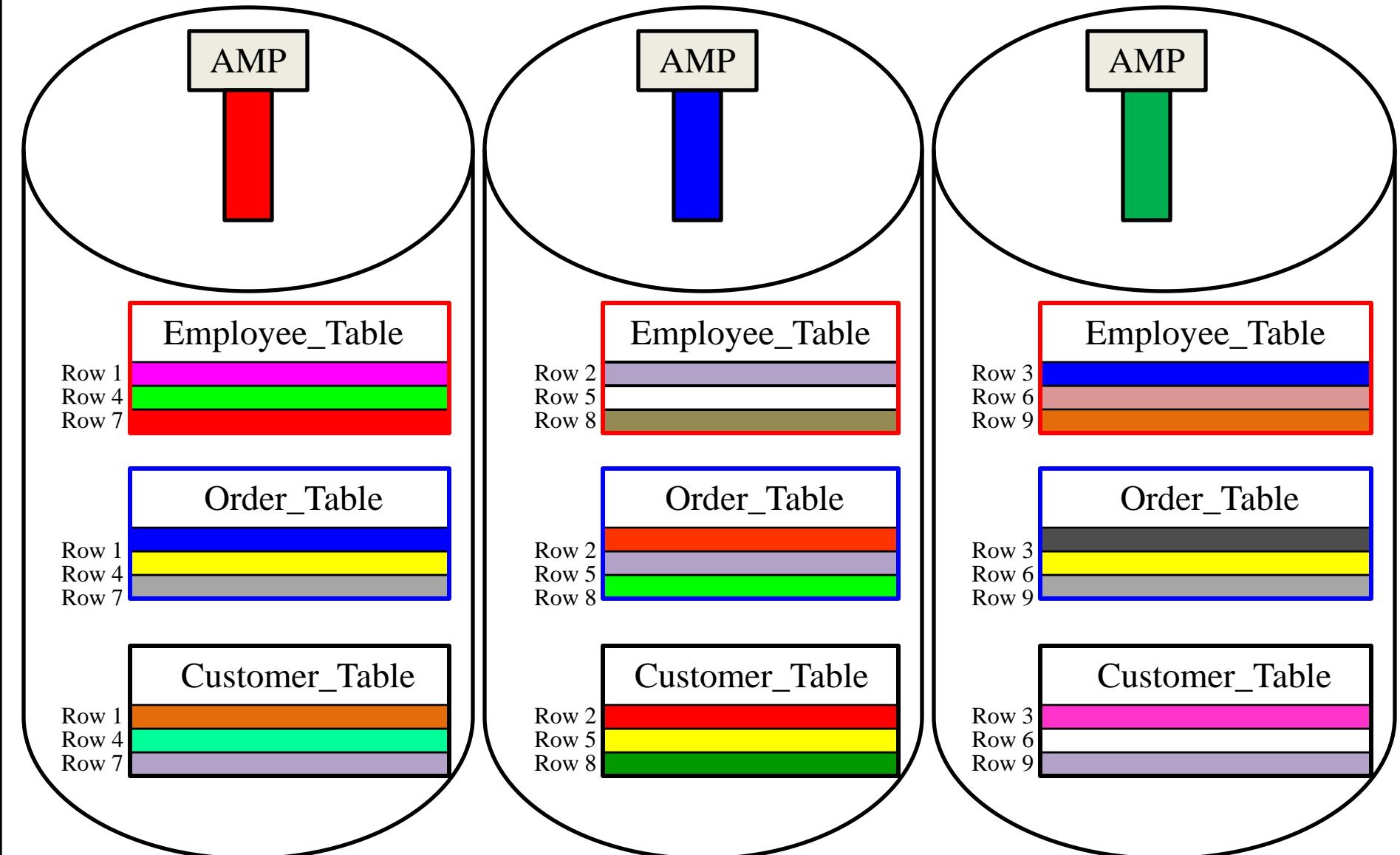
The rows of a Teradata table are spread across the AMPS so each AMP can then process an equal amount of the rows when a USER queries the table.

The Teradata Architecture



The Parsing Engine (PE) takes the User's SQL and builds a Plan for each AMP to follow to retrieve the data. Parallel Processing is all about each AMP doing an equal amount of the work. If they start at the same time and end the same time they are performing true Parallel Processing. All communication is done over the BYNET.

All Teradata Tables are spread across All AMPS

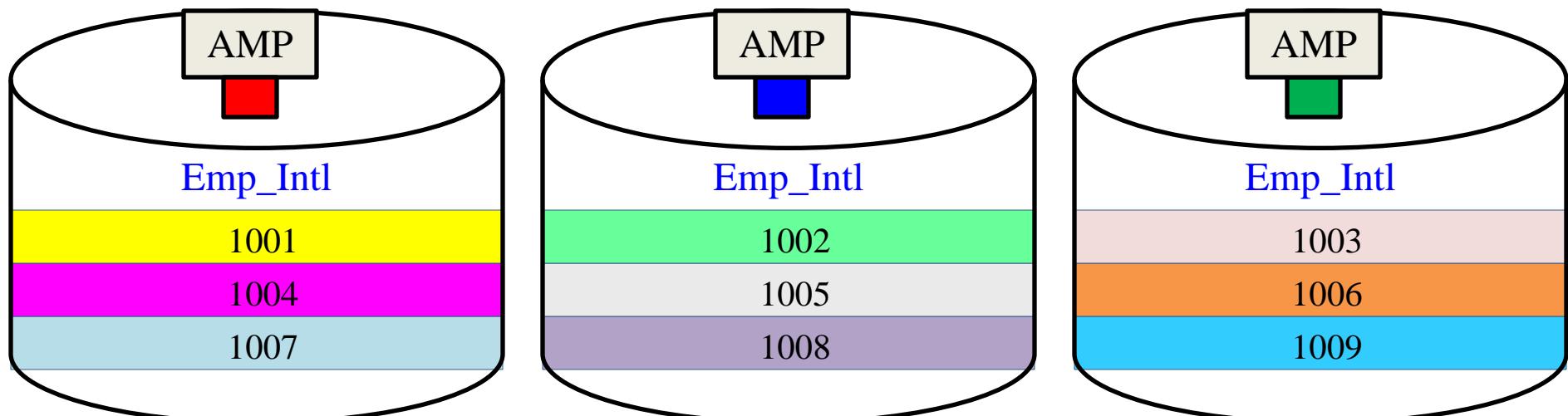


Each table dreams of spreading their rows equally across the AMPS. Above are three tables with each table holding 9 rows (3-rows per AMP).

Each Table has a Column that is the Primary Index

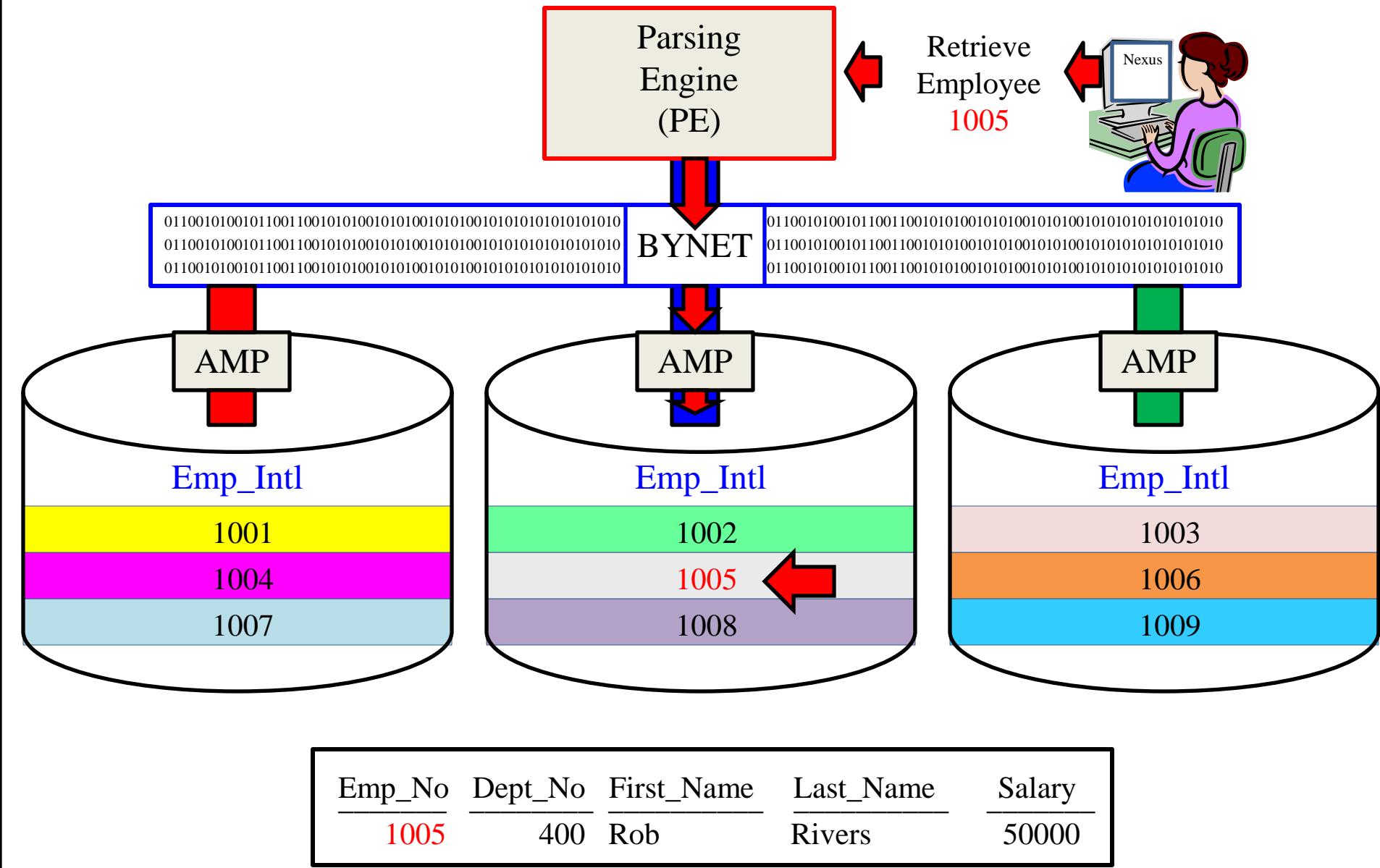
Emp_Intl					
Emp_No	Dept_No	First_Name	Last_Name	Salary	
1001	100	Rafael	Minal	90000.00	
1002	200	Maria	Gomez	80000.00	
1003	300	Charl	Kertzel	70000.00	
1004	400	Kyle	Stover	60000.00	
1005	400	Rob	Rivers	50000.00	
1006	300	Inna	Kinski	50000.00	
1007	200	Sushma	Davis	50000.00	
1008	100	Mo	Khan	60000.00	
1009	300	Mo	Swartz	70000.00	

Primary
Index



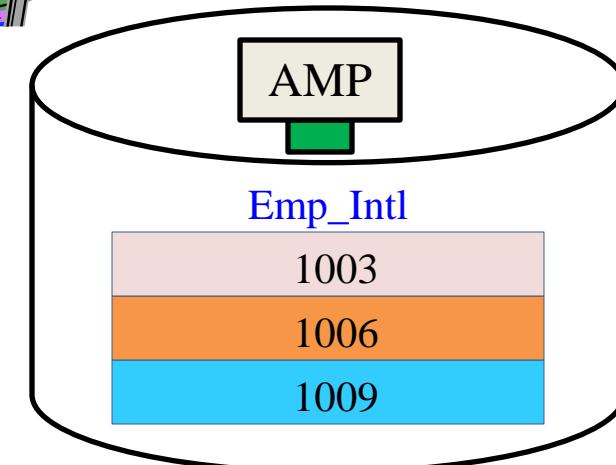
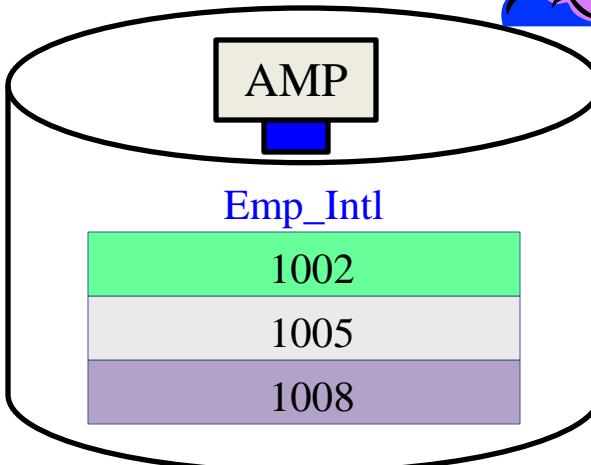
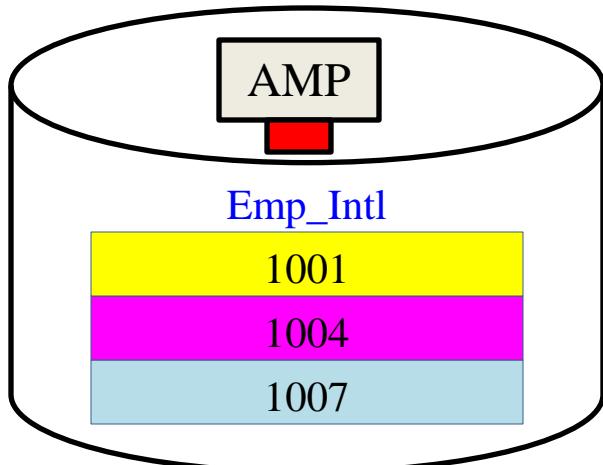
Rows are spread across AMPs based only on the value of the Primary Index column.

Each Table has a Column that is the Primary Index



Use the Primary Index column in your SQL WHERE clause and only 1-AMP retrieves.

A Full Table Scan uses All-AMPs in Parallel

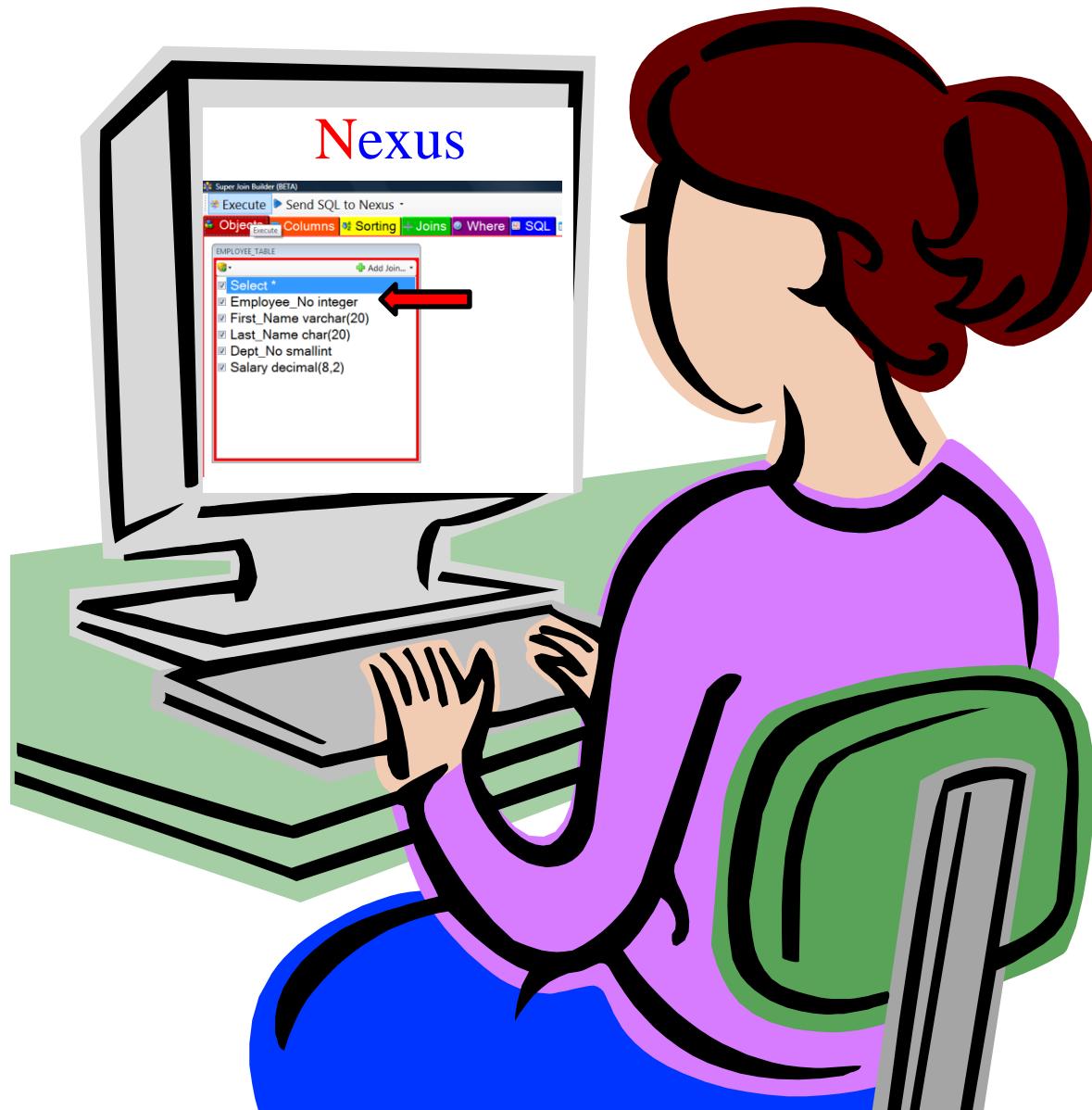


Each AMP
read
its 3-rows
simultaneously
(in Parallel)

Emp_No	Dept_No	First_Name	Last_Name	Salary
1001	100	Rafael	Minal	90000.00
1002	200	Maria	Gomez	80000.00
1003	300	Charl	Kertzel	70000.00
1004	400	Kyle	Stover	60000.00
1005	400	Rob	Rivers	50000.00
1006	300	Inna	Kinski	50000.00
1007	200	Sushma	Davis	50000.00
1008	100	Mo	Khan	60000.00
1009	300	Mo	Swartz	70000.00

A Full Table Scan has each AMP read each row it is responsible for from start to end.

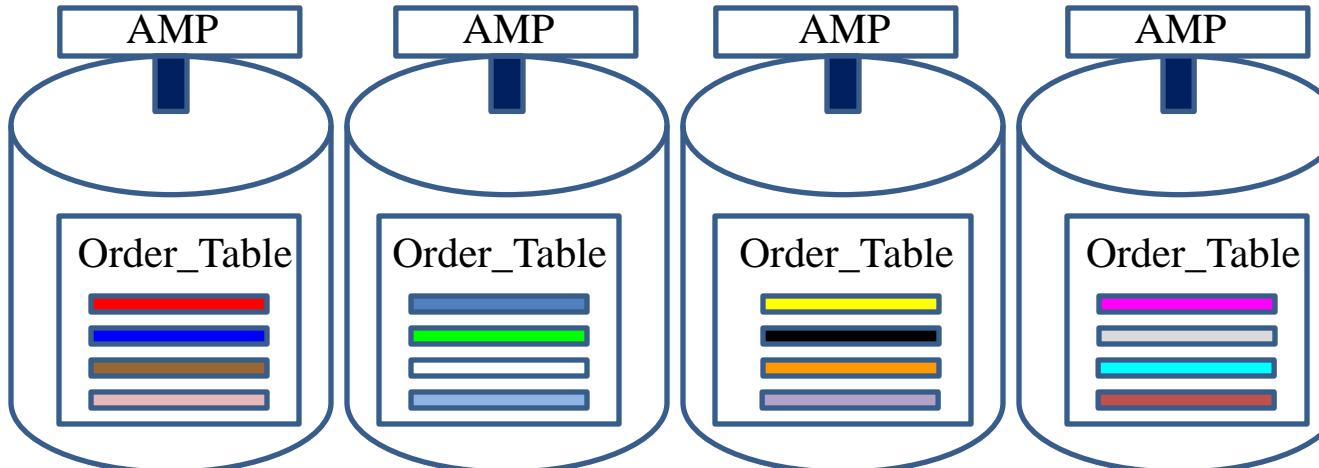
Knowing the Primary Index of a Table is Vital



Find out the Primary Index of a Table and you can always query using the Fast Path.

Teradata Systems can Add AMPs for Linear Scalability

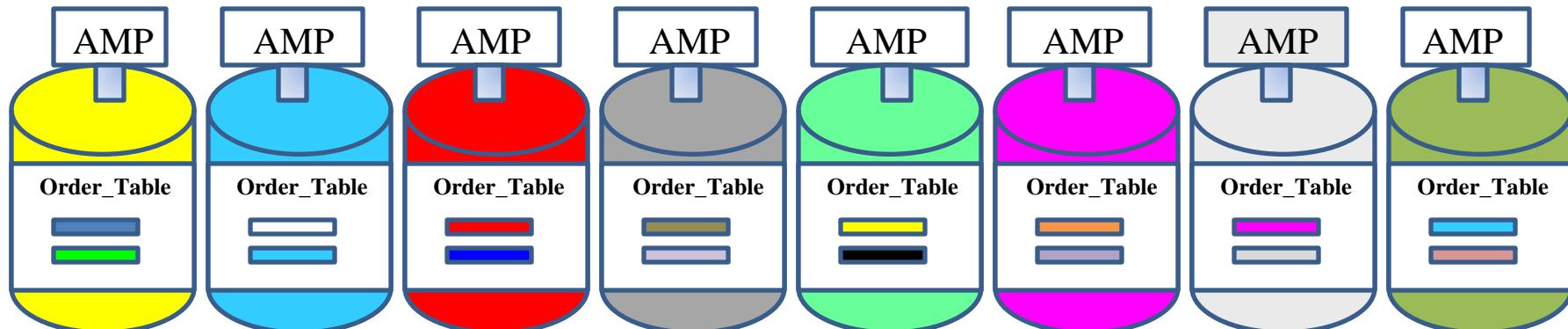
1



4-AMP Test
System.

Order_Table
has 16 rows.

2



8-AMP Production System – with the same 16 rows

If you double the size of your system (Double the AMPs) the system is twice as fast! System number one has only 4-AMPs, but system two has 8-AMPs and is twice as fast. When a customer buys more hardware they are adding AMPs to the system. Once the hardware is configured the AMPs will redistribute the data to include the new AMPs.

Watch the Video on Teradata Architecture



Tera-Tom Trivia

Tom Coffing was the first athlete to ever place at State in any sport at Lakota High School back in 1975. Tom went on to place 4th, 3rd, and 2nd in the state as a Lakota wrestler. Tom was later inducted into the first class of the Lakota Hall of Fame. Tom is pictured above at the ceremony.

Click on the **link** below or place it in your browser and watch the video on Teradata Architecture.

<http://www.coffingdw.com/TbasicsV12/architecture1.wmv>

Chapter 2

The Primary Index

“Civilized society is perpetually menaced with disintegration through this primary hostility of men towards one another.”

- Sigmund Freud

Table of Contents Chapter 2 – Primary Index

- [The Primary Index is defined when the table is CREATED](#)
- [A Unique Primary Index \(UPI\)](#)
- [Primary Index in the WHERE Clause - Single-AMP Retrieve](#)
- [A Non-Unique Primary Index \(NUPI\)](#)
- [Primary Index in the WHERE Clause - Single-AMP Retrieve](#)
- [A conceptual example of a Multi-Column Primary Index](#)
- [Primary Index in the WHERE Clause - Single-AMP Retrieve](#)
- [A conceptual example of a Table with NO PRIMARY INDEX](#)
- [A Full Table Scan is likely on a table with NO Primary Index](#)
- [Table CREATE Examples with four different Primary Indexes](#)
- [What happens when you forget the Primary Index?](#)
- [Why create a table with No Primary Index \(NoPI\)?](#)
- [Watch the Video on the Primary Index](#)

The Primary Index is defined when the table is CREATED

The screenshot shows the Nexus interface with a tree view on the left and a query editor on the right.

Tree View:

- Systems
- DB2 DB2
- Greenplum
- Netezza
- Oracle
- Sandbox
- SQL SERVER
- Teradata
- SQL_CLASS
 - Tables
 - Addresses
 - Claims
 - Course_table
 - Customer_table
 - Department_table
 - department_table2
 - Emp_Job_table
 - Employee_table
 - employee_table2
 - Employee Table3

Query Editor:

```
CREATE TABLE Department_Table
(Dept_No          SMALLINT,
Department_Name  CHAR(20),
Mgr_No           INTEGER,
Budget           DECIMAL(10,2))
UNIQUE PRIMARY INDEX ( Dept_No );
```

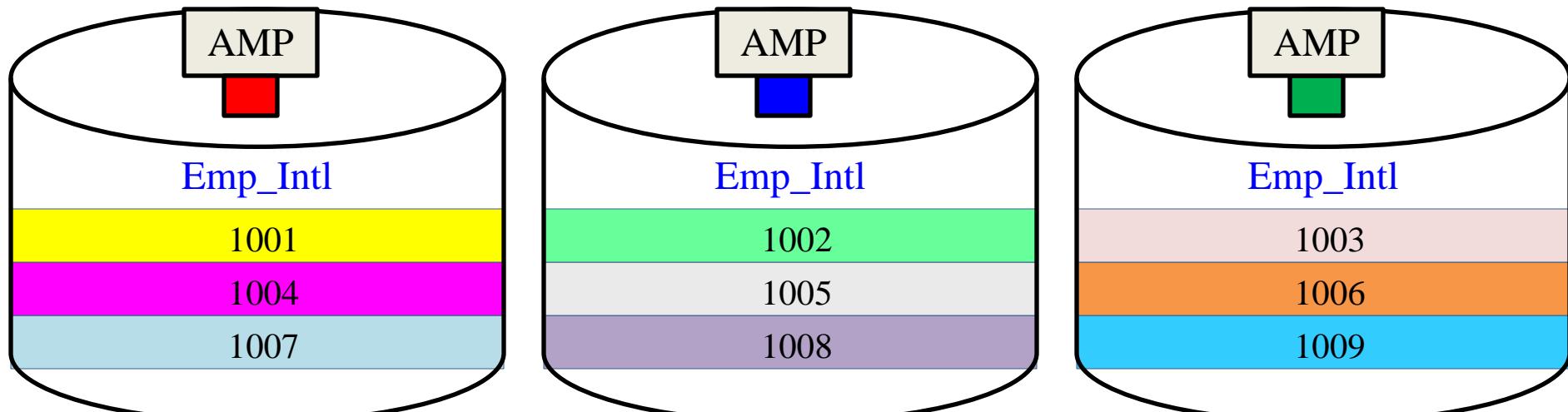
A red arrow points upwards from the text "The Primary Index is defined when the table is CREATED." to the word "INDEX" in the SQL code.

The Primary Index is defined when the table is CREATED. Above we have a UPI which stands for a Unique Primary Index (UPI).

A Unique Primary Index (UPI)

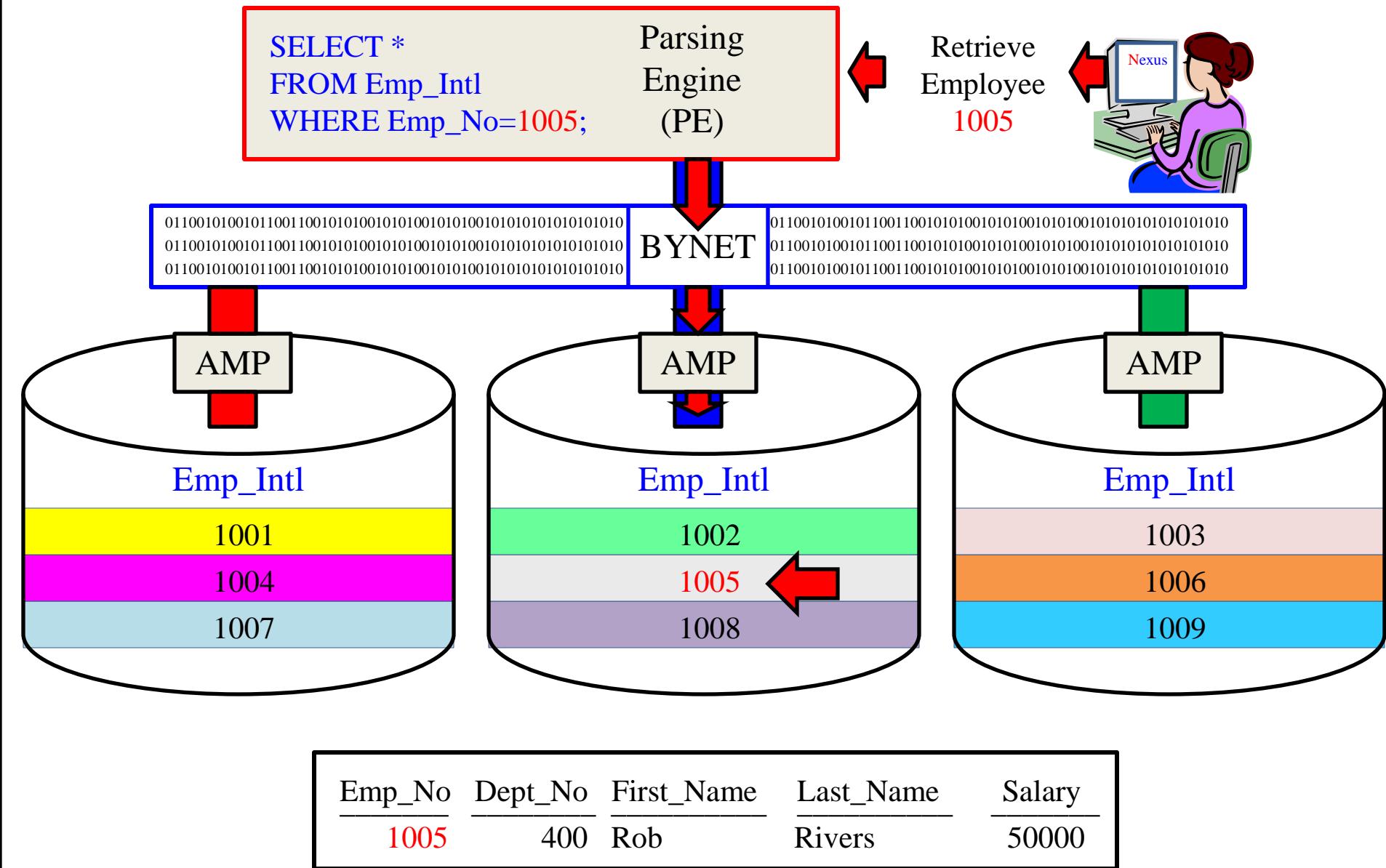
```
CREATE TABLE Emp_Intl  
(Emp_No          INTEGER,  
Dept_No         SMALLINT,  
First_Name      VARCHAR(12),  
Last_Name       CHAR(20),  
Salary          DECIMAL(10,2))  
UNIQUE Primary Index( Emp_No )
```

Unique Primary Index		Emp_Intl			
	Emp_No	Dept_No	First_Name	Last_Name	Salary
	1001	100	Rafael	Minal	90000.00
	1002	200	Maria	Gomez	80000.00
	1003	300	Charl	Kertzel	70000.00
	1004	400	Kyle	Stover	60000.00
	1005	400	Rob	Rivers	50000.00
	1006	300	Inna	Kinski	50000.00
	1007	200	Sushma	Davis	50000.00
	1008	100	Mo	Khan	60000.00
	1009	300	Mo	Swartz	70000.00



A Unique Primary Index (UPI) spreads the rows of a table evenly across the AMPS.

Primary Index in the WHERE Clause - Single-AMP Retrieve

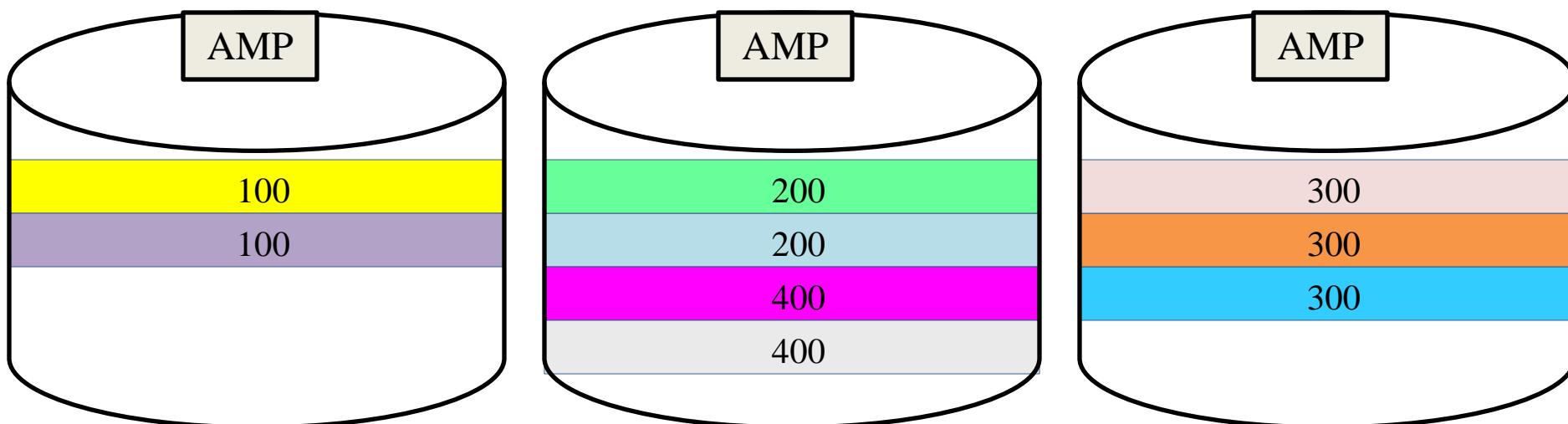


Use the Primary Index column in your SQL WHERE clause and only 1-AMP retrieves.

A Non-Unique Primary Index (NUPI)

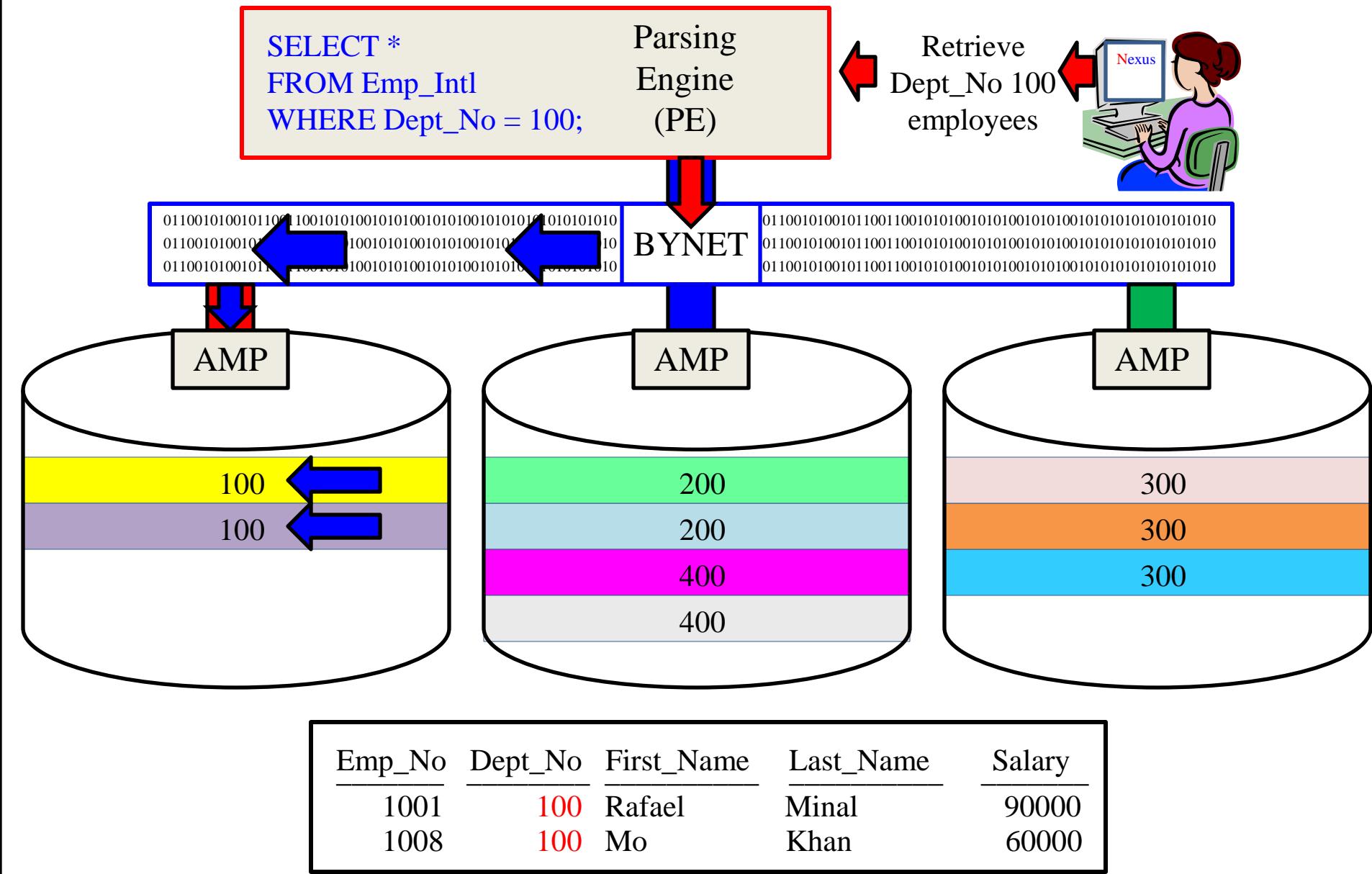
```
CREATE TABLE Emp_Intl
(Emp_No          INTEGER,
Dept_No          SMALLINT,
First_Name       VARCHAR(12),
Last_Name        CHAR(20),
Salary           DECIMAL(10,2))
Primary Index( Dept_No )
```

NUPI		Emp_Intl			
Emp_No	Dept_No	First_Name	Last_Name	Salary	
1001	100	Rafael	Minal	90000.00	
1002	200	Maria	Gomez	80000.00	
1003	300	Charl	Kertzel	70000.00	
1004	400	Kyle	Stover	60000.00	
1005	400	Rob	Rivers	50000.00	
1006	300	Inna	Kinski	50000.00	
1007	200	Sushma	Davis	50000.00	
1008	100	Mo	Khan	60000.00	
1009	300	Mo	Swartz	70000.00	



A Non-Unique Primary Index (NUPI) will have duplicates grouped together on the same AMP so data will always be skewed (uneven). The above skew is reasonable.

Primary Index in the WHERE Clause - Single-AMP Retrieve

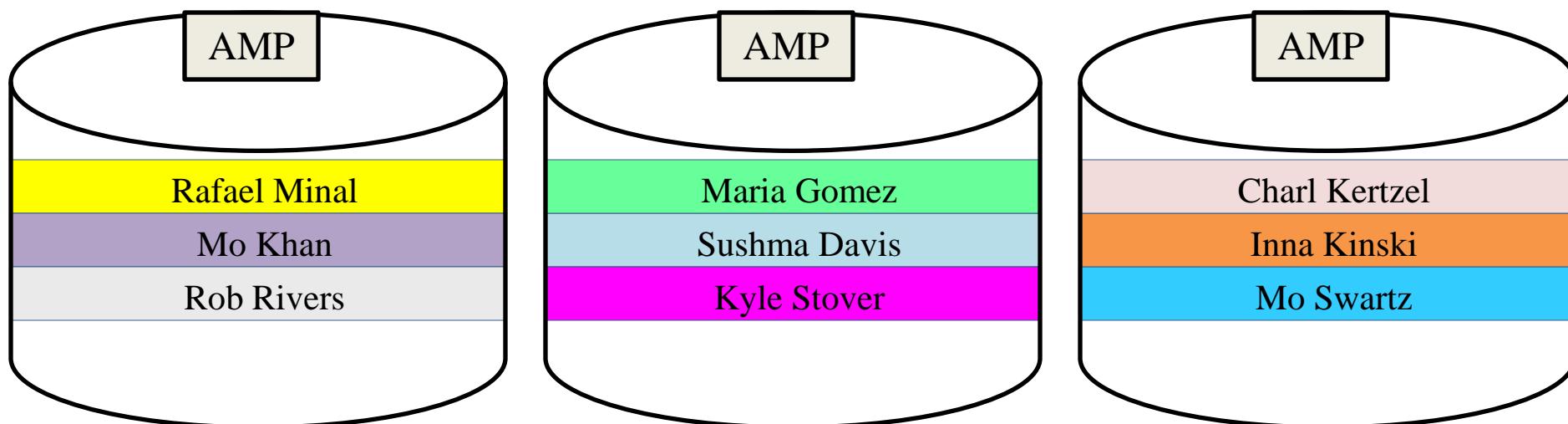


Use the Primary Index column in your SQL WHERE clause and only 1-AMP retrieves.

A conceptual example of a Multi-Column Primary Index

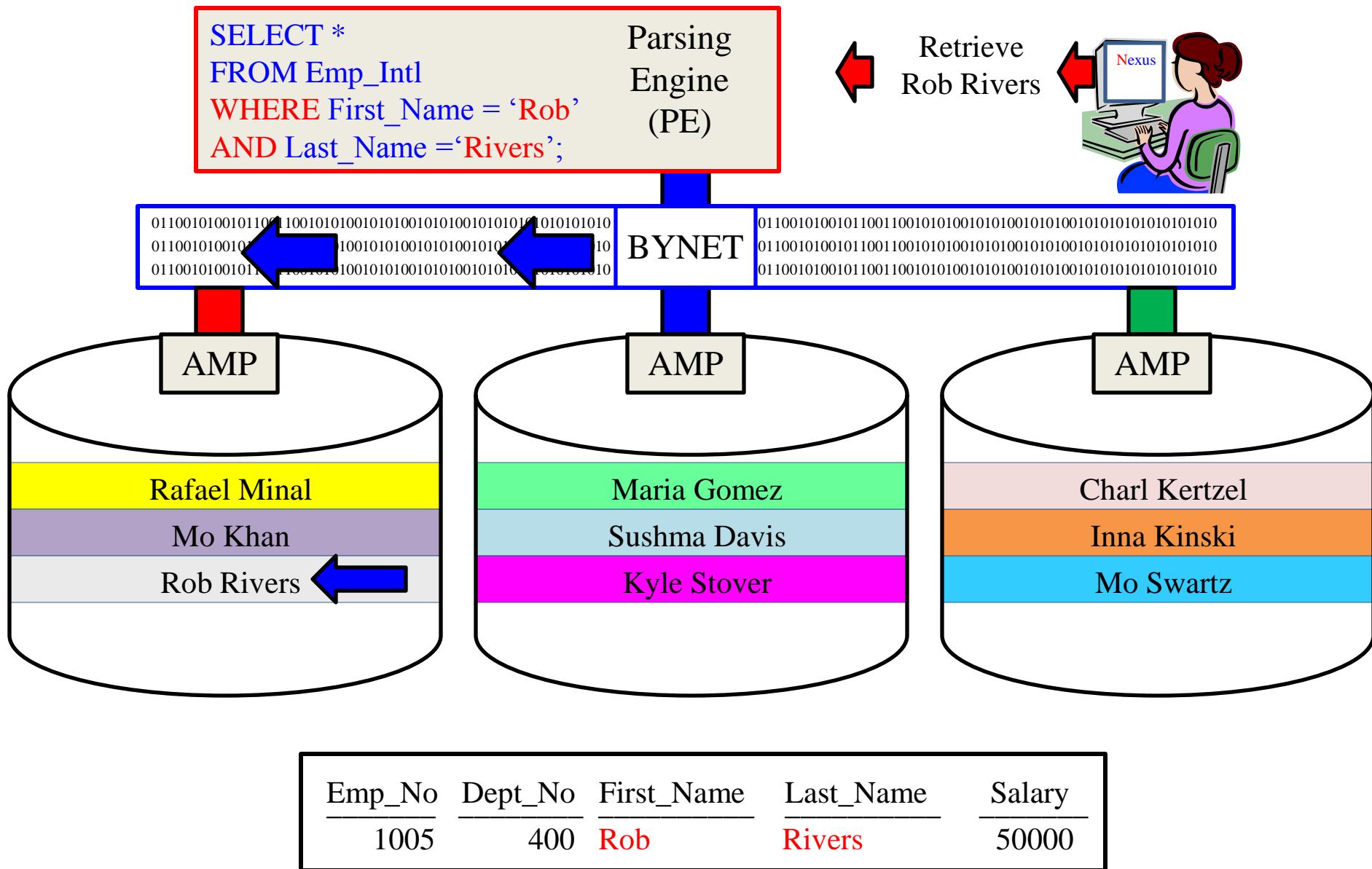
```
CREATE TABLE Emp_Intl
(Emp_No          INTEGER,
Dept_No         SMALLINT,
First_Name      VARCHAR(12),
Last_Name       CHAR(20),
Salary          DECIMAL(10,2))
Primary Index( First_Name, Last_Name)
```

Multi-Column Primary Index					
Emp_No	Dept_No	First_Name	Last_Name	Salary	
1001	100	Rafael	Minal	90000.00	
1002	200	Maria	Gomez	80000.00	
1003	300	Charl	Kertzel	70000.00	
1004	400	Kyle	Stover	60000.00	
1005	400	Rob	Rivers	50000.00	
1006	300	Inna	Kinski	50000.00	
1007	200	Sushma	Davis	50000.00	
1008	100	Mo	Khan	60000.00	
1009	300	Mo	Swartz	70000.00	



A table can have only one Primary Index, but you can combine up to 64 columns together max to form one Multi-Column Primary Index.

Primary Index in the WHERE Clause - Single-AMP Retrieve



Use the Primary Index column in your SQL WHERE clause and only 1-AMP retrieves.

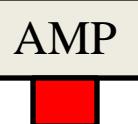
A conceptual example of a Table with NO PRIMARY INDEX

```
CREATE TABLE Emp_Intl  
(Emp_No          INTEGER,  
Dept_No         SMALLINT,  
First_Name      VARCHAR(12),  
Last_Name       CHAR(20),  
Salary          DECIMAL(10,2)  
) NO Primary Index
```



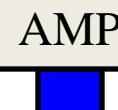
No Primary Index

Emp_Intl					
Emp_No	Dept_No	First_Name	Last_Name	Salary	
1001	100	Rafael	Minal	90000.00	
1002	200	Maria	Gomez	80000.00	
1003	300	Charl	Kertzel	70000.00	
1004	400	Kyle	Stover	60000.00	
1005	400	Rob	Rivers	50000.00	
1006	300	Inna	Kinski	50000.00	
1007	200	Sushma	Davis	50000.00	
1008	100	Mo	Khan	60000.00	
1009	300	Mo	Swartz	70000.00	



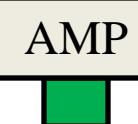
Emp_Intl Table

1001	100	Rafael	Minal	90000
1004	400	Kyle	Stover	60000
1007	200	Sushma	Davis	50000



Emp_Intl Table

1002	200	Maria	Gomez	80000
1005	400	Rob	Rivers	50000
1008	100	Mo	Khan	60000

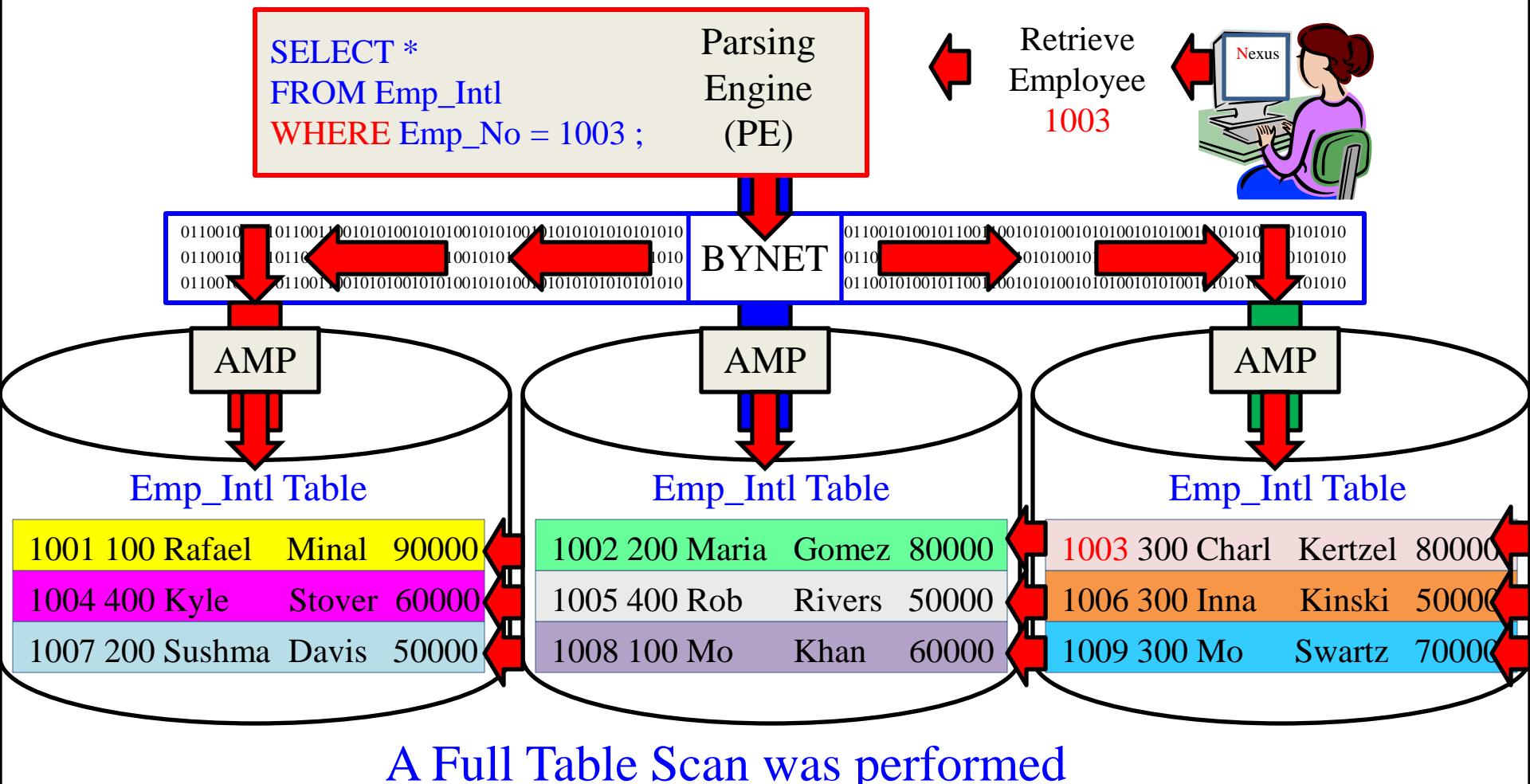


Emp_Intl Table

1003	300	Charl	Kertzel	80000
1006	300	Inna	Kinski	50000
1009	300	Mo	Swartz	70000

A Table that specifically states NO PRIMARY INDEX will receive no primary index. It will distribute the data evenly but randomly and this is often used as a staging table.

A Full Table Scan is likely on a table with NO Primary Index



Emp_No	Dept_No	First_Name	Last_Name	Salary
1003	300	Charl	Kertzel	70000

Since a NO Primary Index (NoPI) table has no primary index the system retrieves by performing a Full Table Scan, which means All-AMPs read All-Rows they own once.

Table CREATE Examples with four different Primary Indexes

1

```
CREATE TABLE Emp_Intl  
(Emp_No          INTEGER,  
 Dept_No         SMALLINT,  
 First_Name      VARCHAR(12),  
 Last_Name       CHAR(20),  
 Salary          DECIMAL(10,2))  
UNIQUE PRIMARY INDEX ( Emp_No );
```

UPI

2

```
CREATE TABLE Emp_Intl  
(Emp_No          INTEGER,  
 Dept_No         SMALLINT,  
 First_Name      VARCHAR(12),  
 Last_Name       CHAR(20),  
 Salary          DECIMAL(10,2))  
PRIMARY INDEX ( Dept_No );
```

NUPI

3

```
CREATE TABLE Emp_Intl  
(Emp_No          INTEGER,  
 Dept_No         SMALLINT,  
 First_Name      VARCHAR(12),  
 Last_Name       CHAR(20),  
 Salary          DECIMAL(10,2))  
PRIMARY INDEX ( First_Name , Last_Name );
```

Multi-Column NUPI

4

```
CREATE TABLE Emp_Intl  
(Emp_No          INTEGER,  
 Dept_No         SMALLINT,  
 First_Name      VARCHAR(12),  
 Last_Name       CHAR(20),  
 Salary          DECIMAL(10,2))  
NO Primary Index
```

No Primary Index

A table can have only one Primary Index so picking the right one is essential. Above are four different examples for your consideration.

What happens when you forget the Primary Index?

1

```
CREATE TABLE Emp_Intl  
(Emp_No      INTEGER,  
 Dept_No     SMALLINT,  
 First_Name   VARCHAR(12),  
 Last_Name    CHAR(20),  
 Salary       DECIMAL(10,2))
```

2

```
CREATE TABLE Emp_Intl  
(Emp_No      INTEGER,  
 Dept_No     SMALLINT,  
 First_Name   VARCHAR(12),  
 Last_Name    CHAR(20),  
 Salary       DECIMAL(10,2))  
NO Primary Index
```

We forgot to define the Primary Index

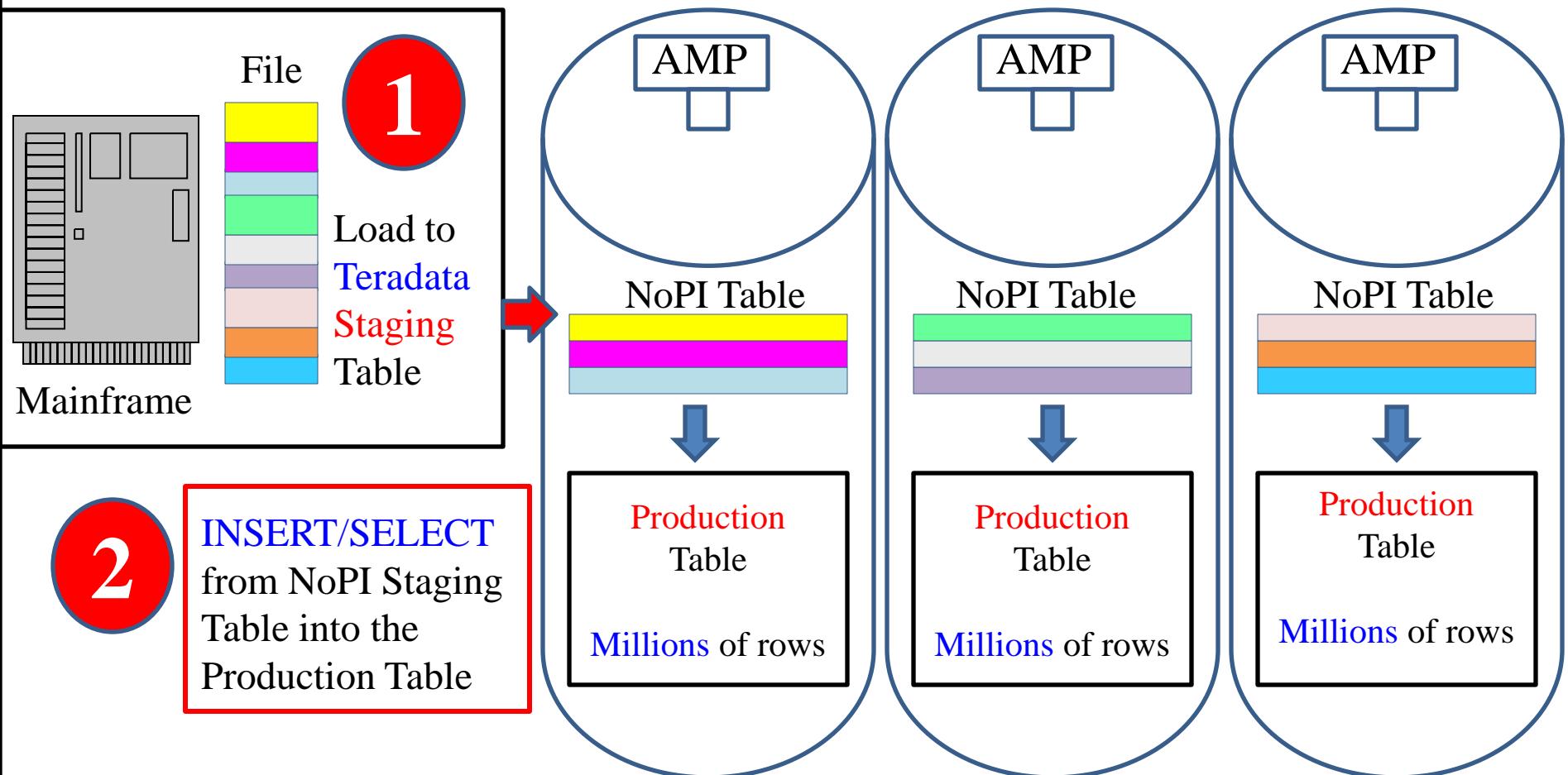
The Primary Index will default to the first column and make Emp_No a Non-Unique Primary Index (NUPI). Be careful!

We specifically stated No Primary Index

This Table will have No Primary Index. It is a NoPI table most likely used for Staging in the ETL process for tables being loaded. The data spreads perfectly.

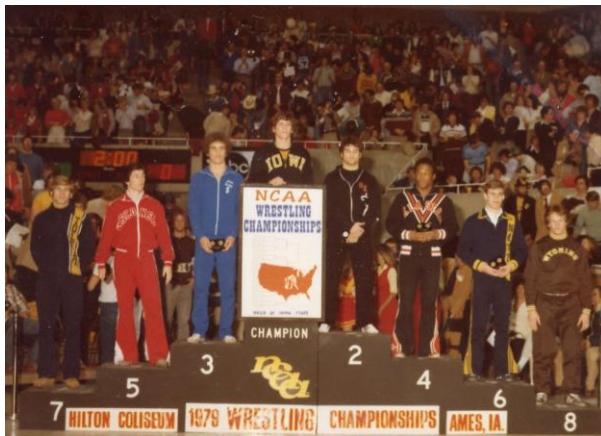
When you forget to define the Primary Index Teradata will default to the first column in the table and it will be defined as Non-Unique. Clearly define what you want!

Why create a table with No Primary Index (NoPI)?



NoPI tables are designed to be staging tables. Data from a Mainframe or server can be loaded onto Teradeca quickly with perfect distribution. Then an INSERT/SELECT can be done to move the data from the staging table (on Teradeca) to the production table (also on Teradeca). The data can be transformed in staging and there are no Load Restrictions with an INSERT/SELECT. A NoPI table usually isn't queried, but can be!

Watch the Video on the Primary Index



Tera-Tom Trivia

Tom Coffing was a two-time All-American wrestler for the University of Arizona. In 1979 Tom was “Sophomore Athlete of the Year” for the University of Arizona. That year Tom placed 3rd in the NCAA (Division 1) and is pictured above at the awards ceremony in Ames Iowa.

Click on the link below or place it in your browser and watch the video on the Primary Index.

http://www.coffingdw.com/TbasicsV12/PrimaryIndexUPI_NUPI.wmv

Chapter 3

Hashing of the

Primary Index

“The true **index** of a man's character is the health of his wife.”

- Cyril Connolly

Table of contents Chapter 3 - Hashing of the Primary Index

- [The Hashing Formula Facts](#)
- [The Hash Map Determines which AMP will own the Row](#)
- [The Hash Map Determines which AMP will own the Row Continued](#)
- [Placing rows on the AMP](#)
- [Placing rows on the AMP Continued](#)
- [A Review of the Hashing Process](#)
- [Non-Unique Primary Indexes have Skewed Data](#)
- [The Uniqueness Value](#)
- [The Row Hash and Uniqueness Value make up the Row-ID](#)
- [A Row-ID Example for a Unique Primary Index](#)
- [A Row-ID Example for a Non-Unique Primary Index \(NUPI\)](#)
- [Two Reasons why each AMP Sorts their rows by the Row-ID](#)
- [AMPs sort their rows by Row-ID to Group Like Data](#)
- [AMPs sort their rows by Row-ID to do a Binary Search](#)
- [Table CREATE Examples with four different Primary Indexes](#)
- [Null Values all Hash to the Same AMP](#)
- [A Unique Primary Index \(UPI\) Example](#)
- [A Non-Unique Primary Index \(NUPI\) Example](#)
- [A Multi-Column Primary Index Example](#)
- [A No Primary Index \(NoPI\) Example](#)
- [Watch the Video on the Hashing the Data](#)

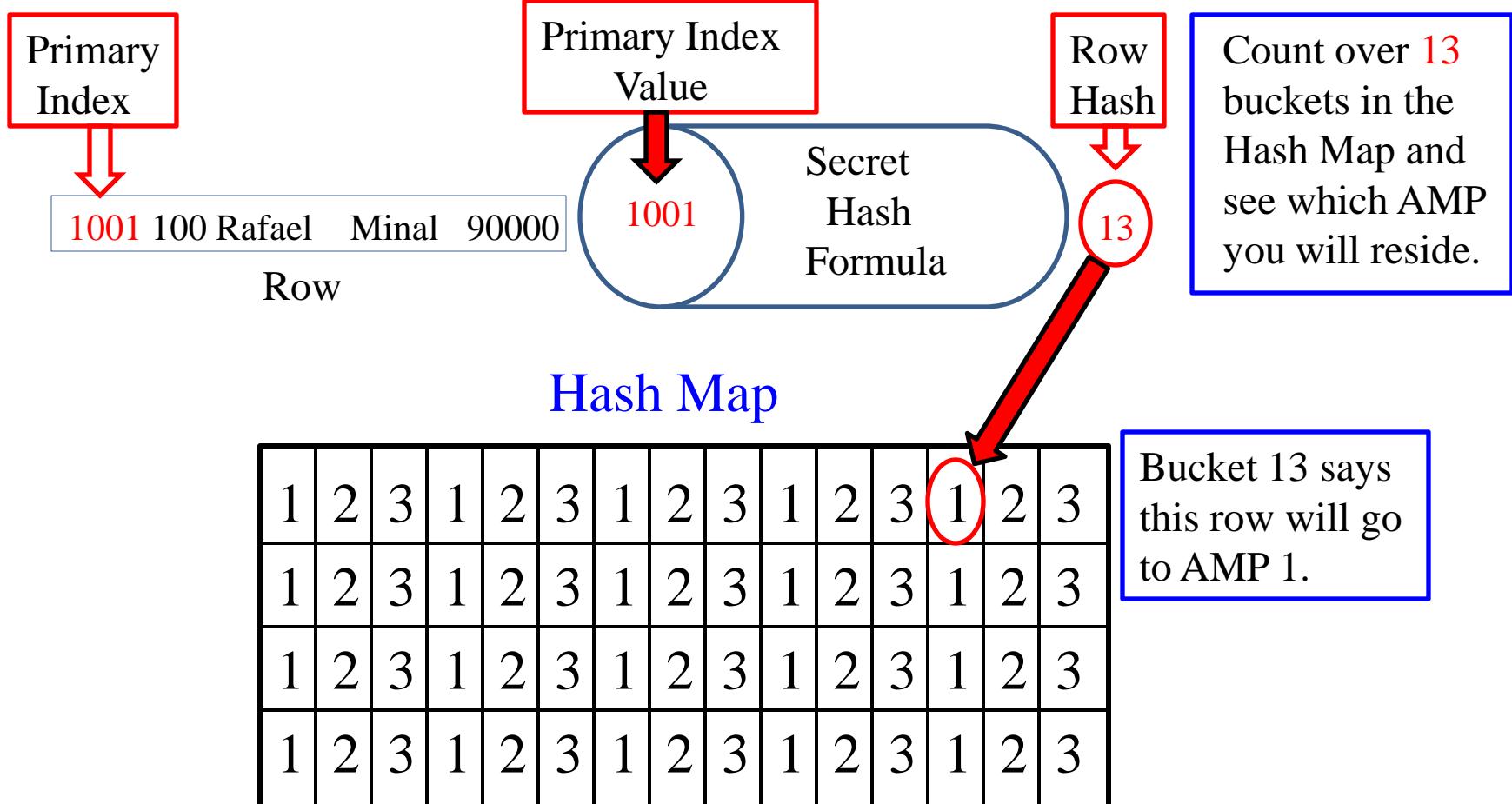
The Hashing Formula Facts



- 1 There is only **one** Hashing Formula.
- 2 A row's Primary Index **value** is hashed and the output is its **Row Hash**.
- 3 The **Row Hash** will be stored with the row (on disk) as part of the **Row_ID**.
- 4 If the Hashing Formula hashes value **1001** and gets a row hash of **13** then it will produce a **13** every time it hashes a **1001** value. Its consistent!

There is one Hashing Formula in Teradata and it is consistent. The concept is to take the value of a row's Primary Index and run it through the Hash Formula. It will produce a Row Hash number. That Row Hash will stay with the row forever and reside as the first part of the row. The Row Hash also determines which AMP owns the row.

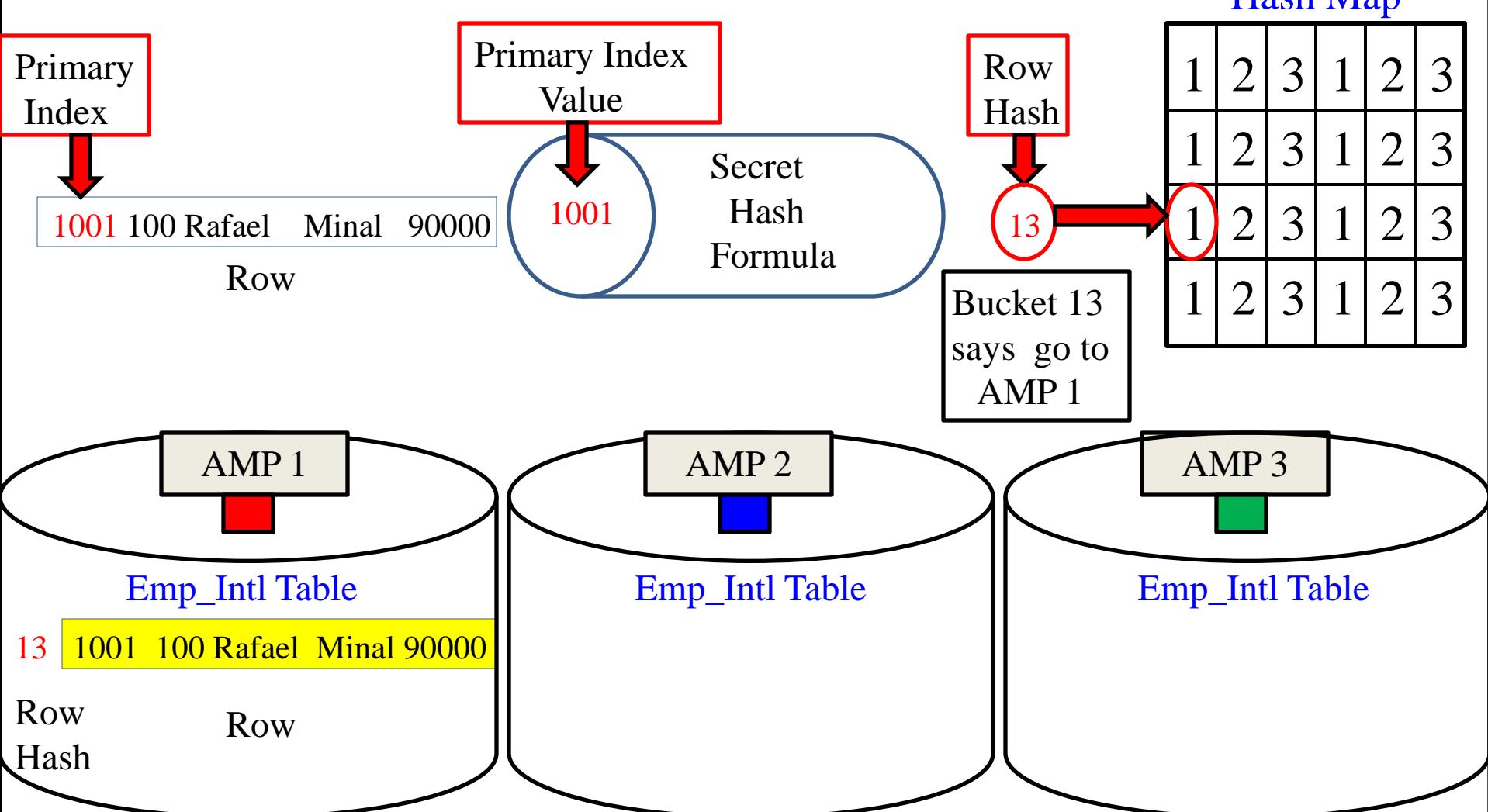
The Hash Map Determines which AMP will own the Row



This Hash Map is for a 3-AMP system

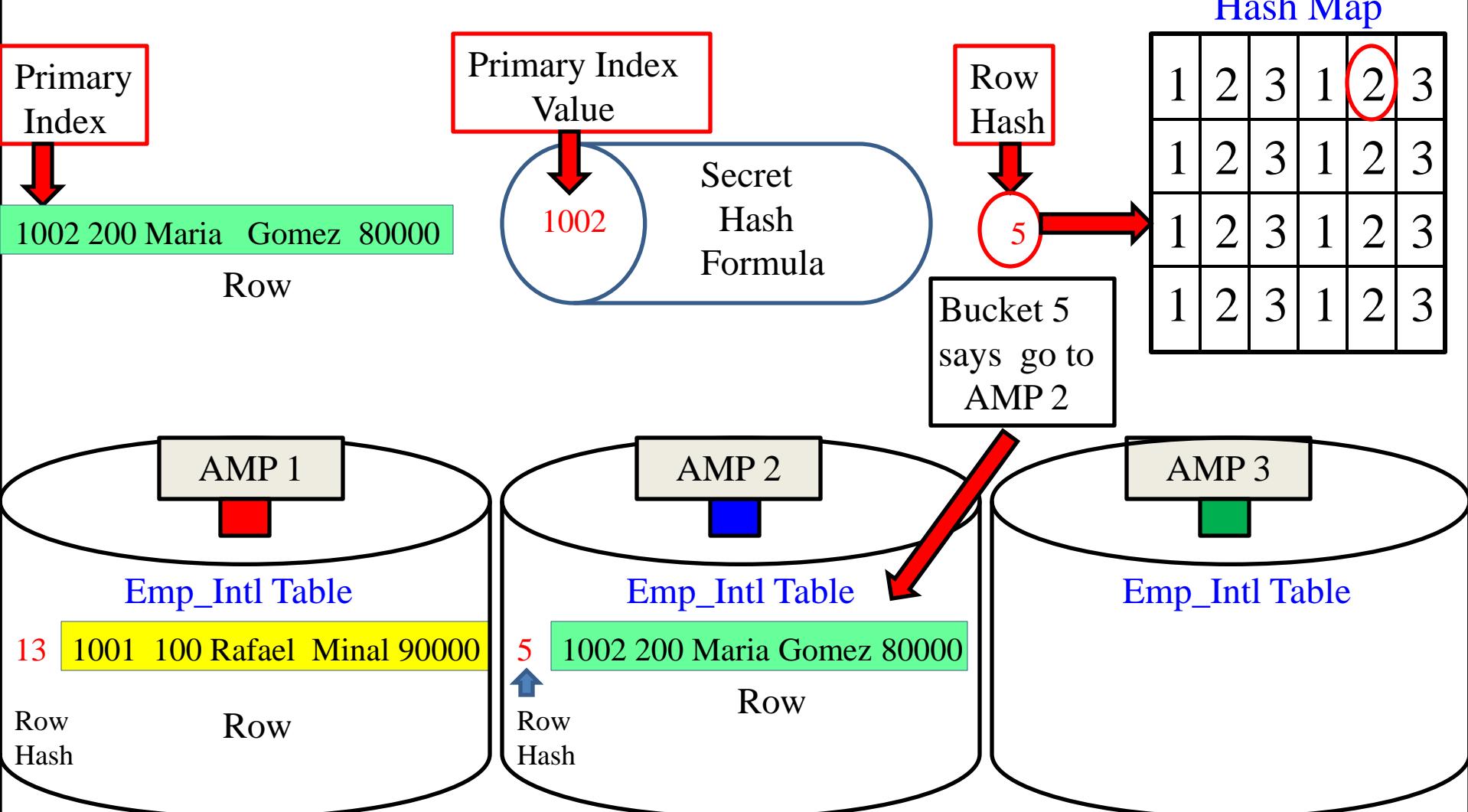
A row will be placed on an AMP after the Parsing Engine (PE) hashes the row's Primary Index value. The output of the Hashing Algorithm is a row's Row Hash. The Row hash goes to a bucket in the Hash Map and is assigned to an AMP.

The Hash Map Determines which AMP will own the Row



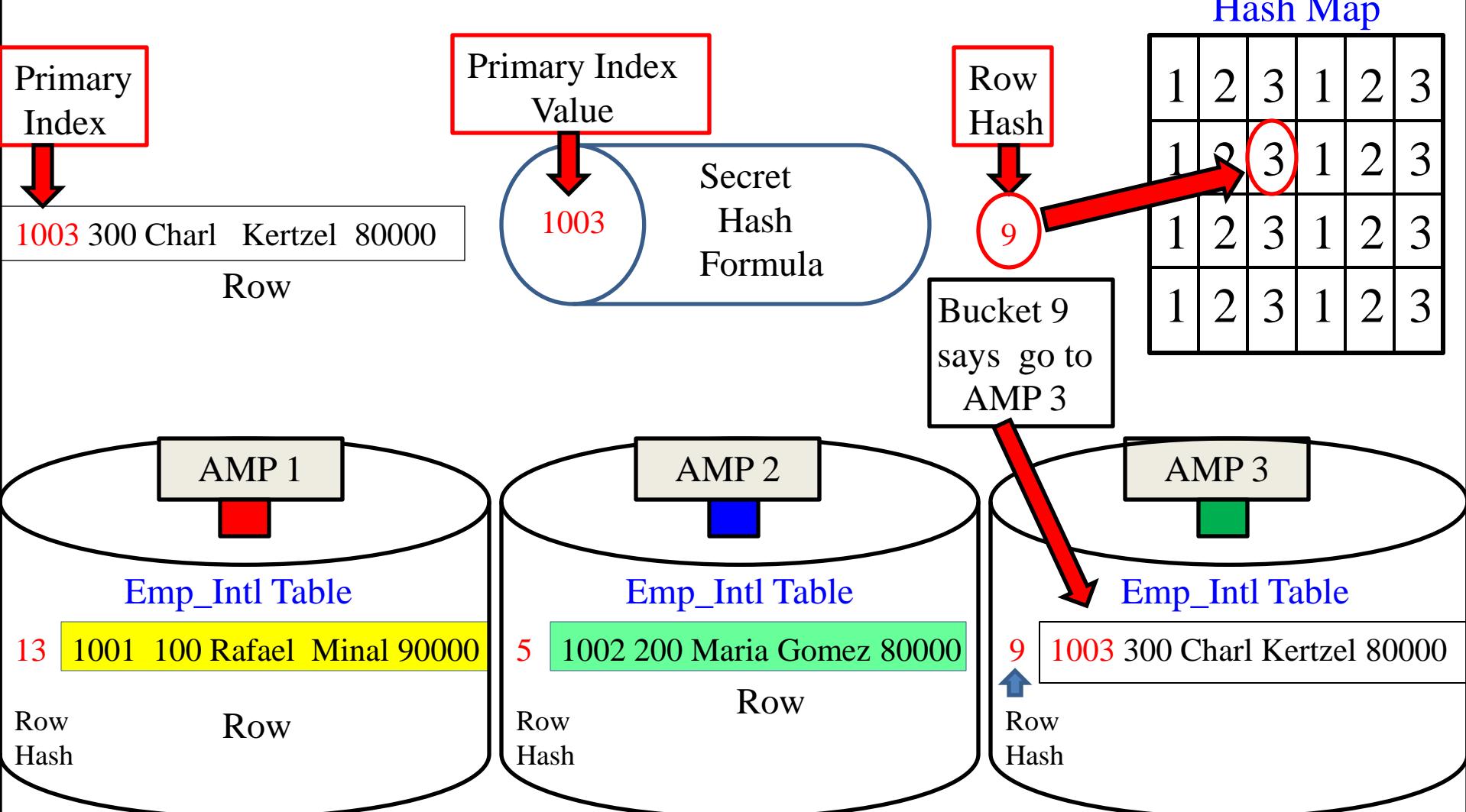
The above example hashed Emp_No 1001 (Primary Index value) and the output was a Row Hash of 13. Teradata counted over to bucket 13 in the Hash Map and it has the number one (1) inside that bucket. This means that this row will go to AMP 1.

Placing rows on the AMP



The above example hashed Emp_No 1002 (Primary Index value) and the output was a Row Hash of 5. Teradata counted over to bucket 5 in the Hash Map and it has the number two (2) inside that bucket. This means that this row will go to AMP 2.

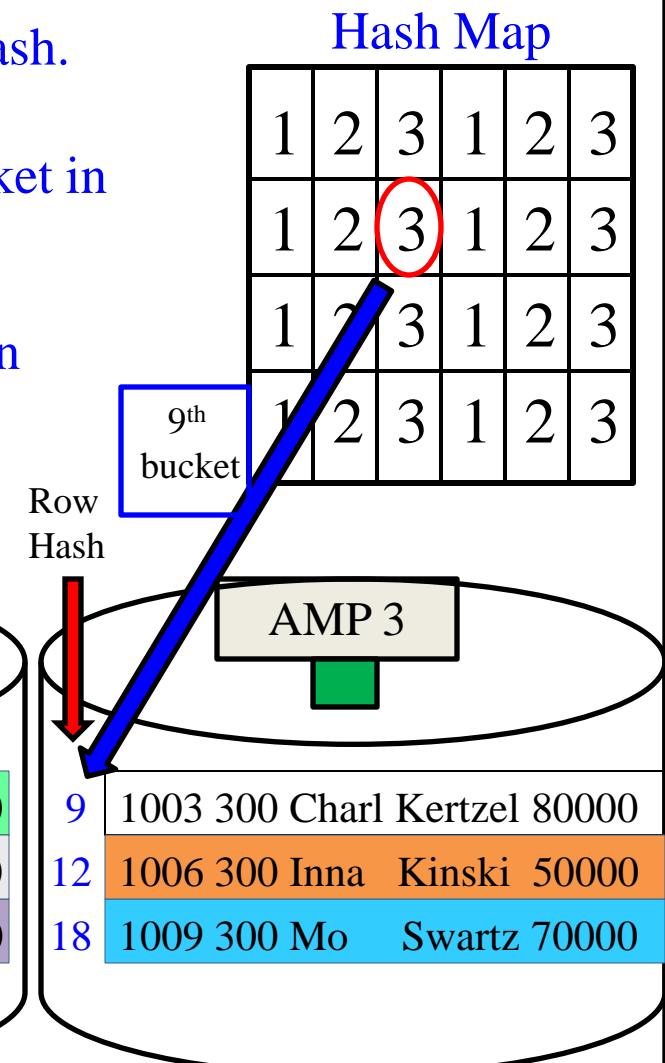
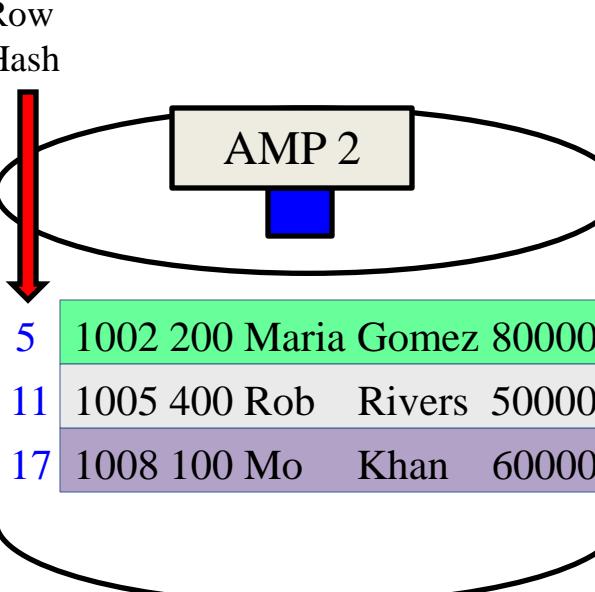
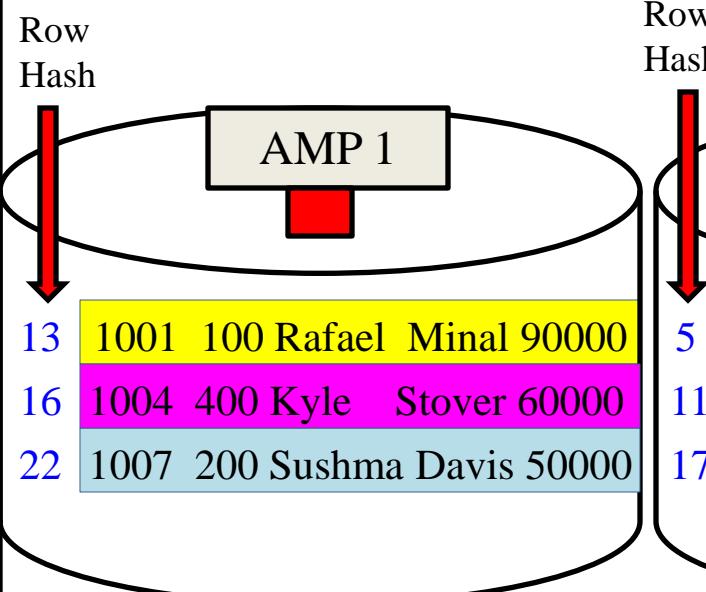
Placing rows on the AMP Continued



The above example hashed Emp_No 1003 (Primary Index value) and the output was a Row Hash of 9. Teradata counted over to bucket 9 in the Hash Map and it has the number one (3) inside that bucket. This means that this row will go to AMP 3.

A Review of the Hashing Process

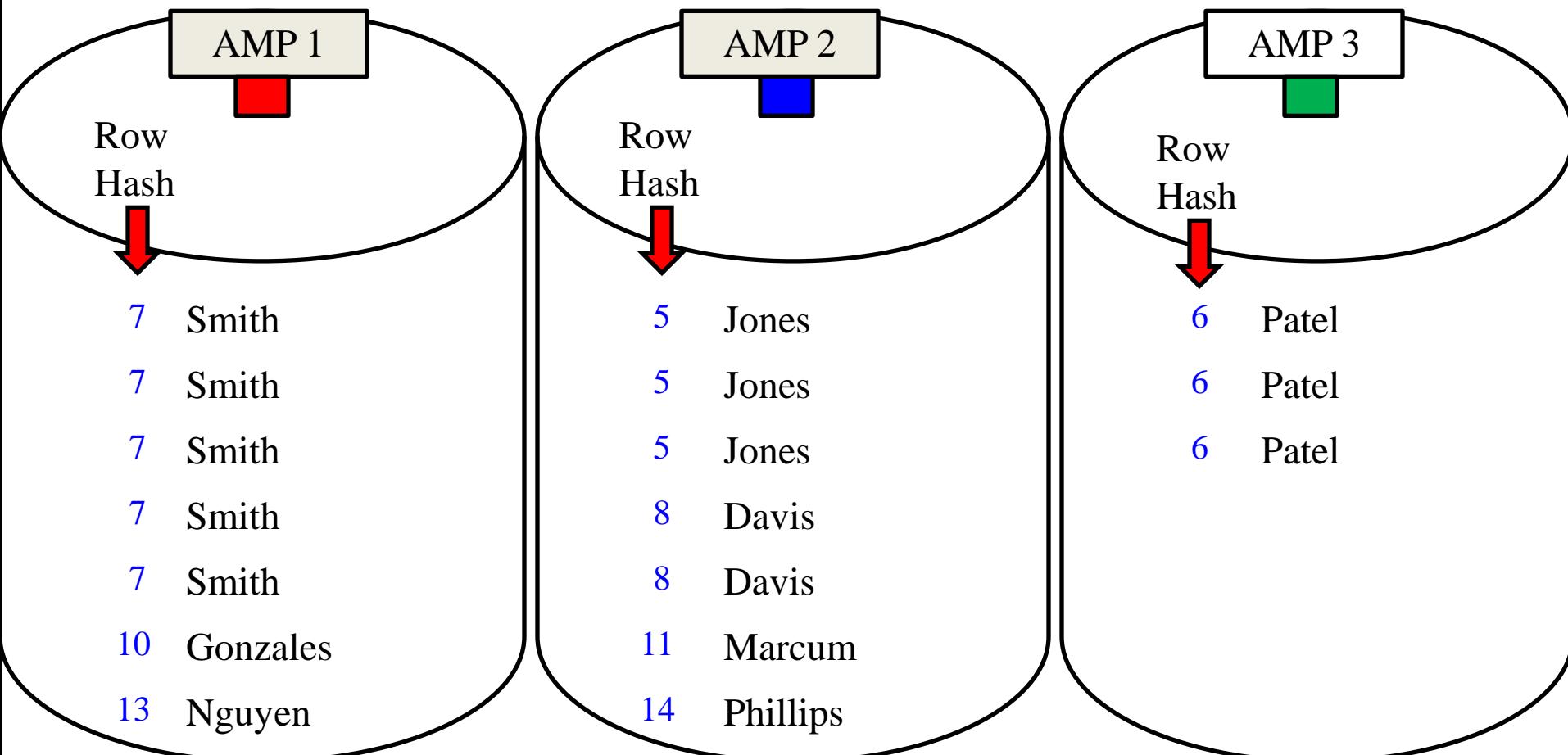
- 1 Hash the Primary Index Value for a row with the Hash Formula.
- 2 The output of the Hash Formula is a 32-bit Row Hash.
- 3 Take the Row Hash and find its corresponding bucket in the Hash Map.
- 4 Send the row and its Row Hash to the AMP listed in the Hash Map Bucket.



Take a look at the row hash for each row and notice it corresponds with the Hash Map.

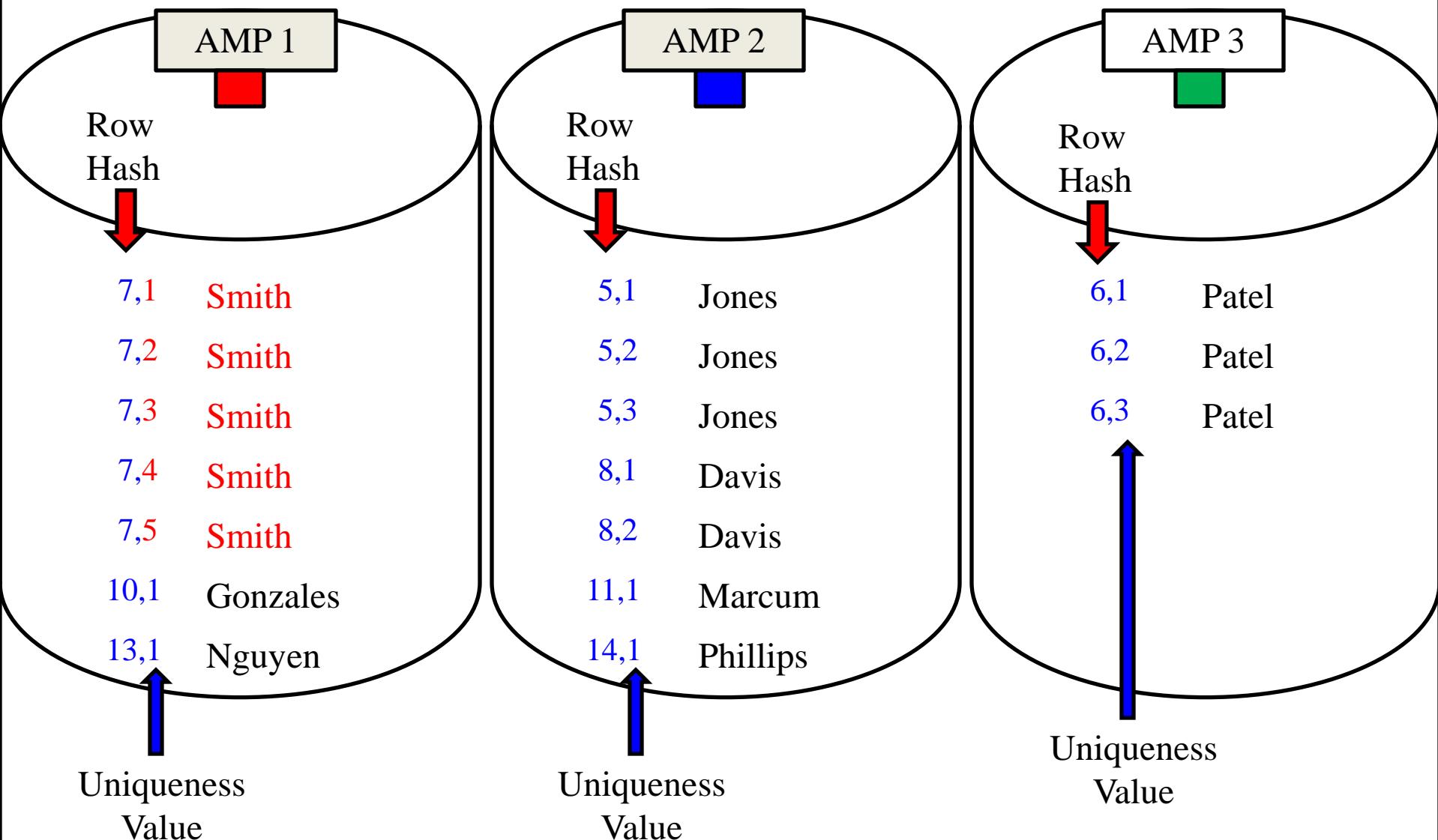
Non-Unique Primary Indexes have Skewed Data

Imagine if we made `Last_Name` the Primary Index for a table. Here is an example of how it would distribute. Notice all duplicates have the same Row Hash.



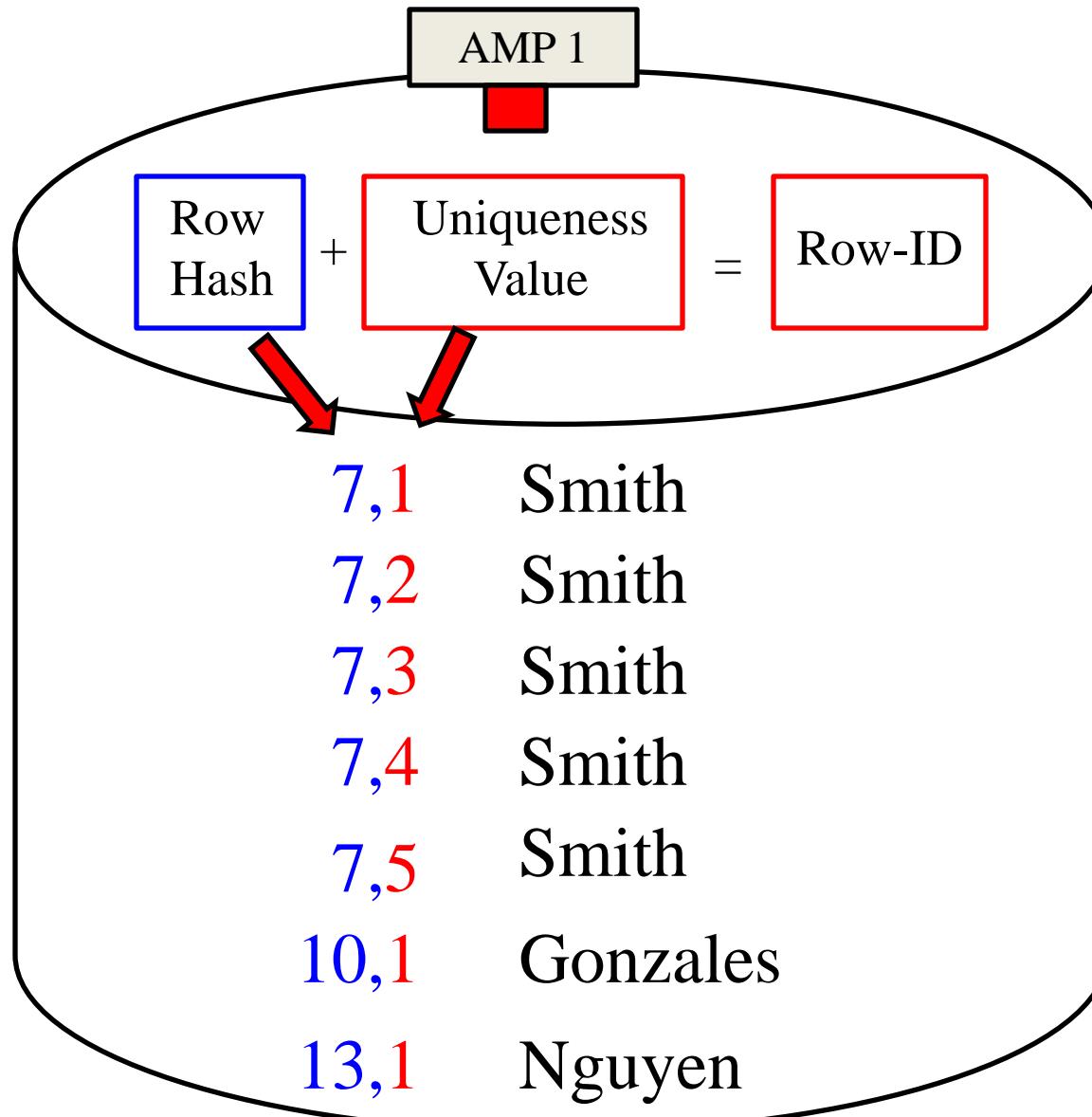
The Hash Formula is consistent so every Smith has the same Row Hash and the same goes for each Jones and each Patel. Therefore duplicate values land on the same AMP.

The Uniqueness Value



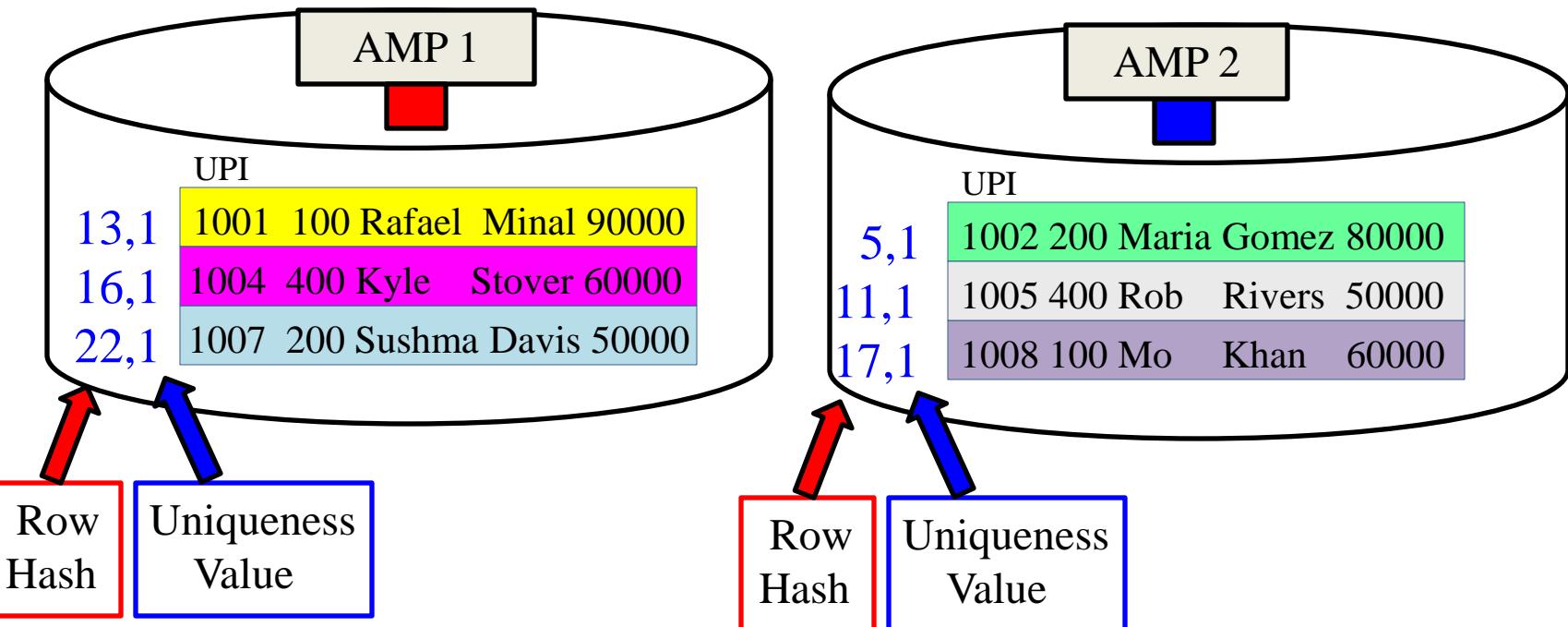
Each AMP will place a Uniqueness Value after the row hash to track duplicate values.

The Row Hash and Uniqueness Value make up the Row-ID



Row-ID equals the Row Hash of the Primary Index column and the Uniqueness Value.

A Row-ID Example for a Unique Primary Index



Notice two things for this **Unique Primary Index (UPI)** example:

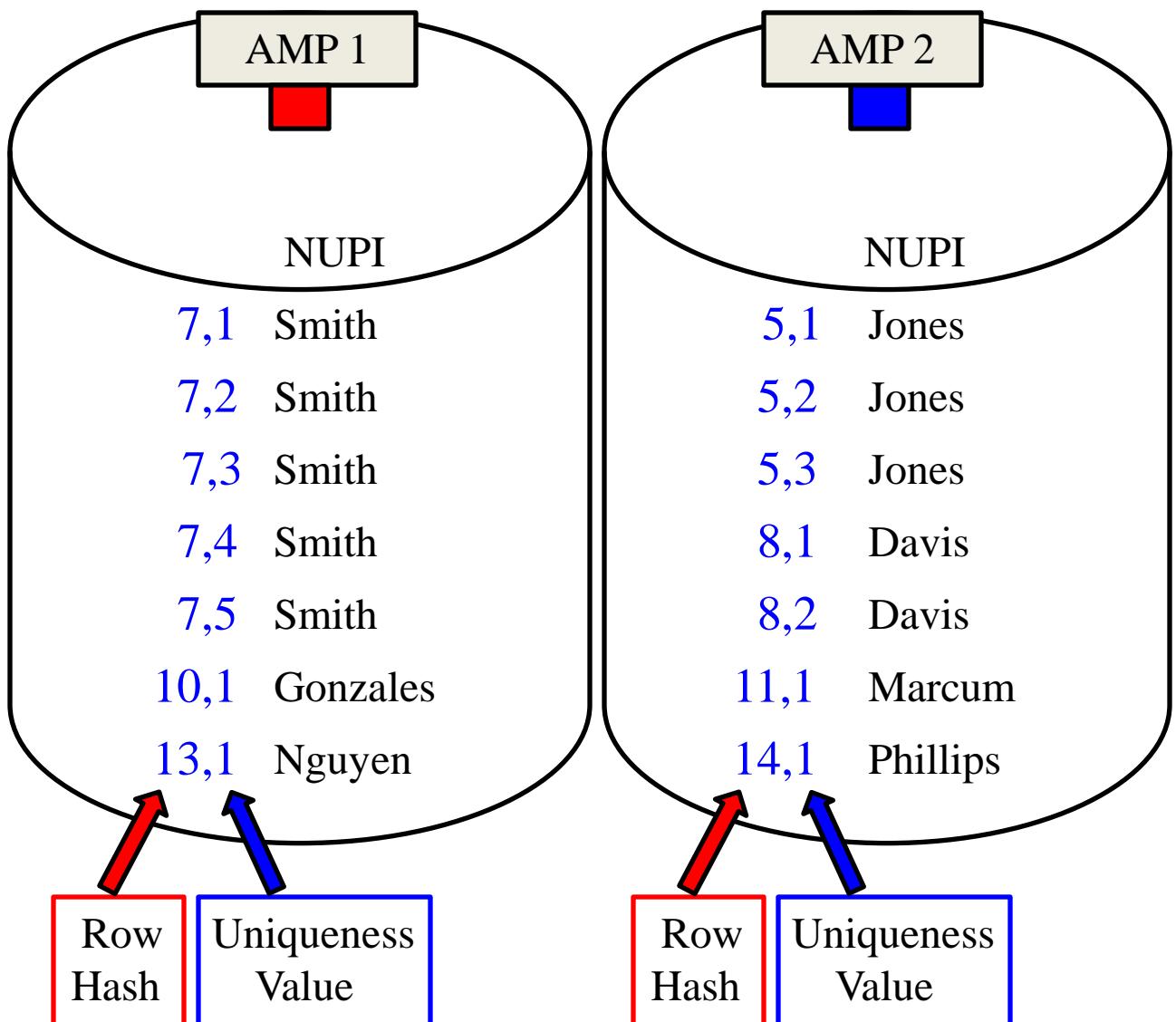
- 1) The **Uniqueness Value** on each Row-ID is **1**.
- 2) Each AMP **sorts** their rows by the **Row-ID**.

Row Hash and the Uniqueness Value make up the Row-ID. AMPS sort by the Row-ID.

A Row-ID Example for a Non-Unique Primary Index(NUPI)

Notice two things:

- 1) Uniqueness Value increases on all duplicate names.
- 2) Each AMP sorts their rows by the Row-ID.

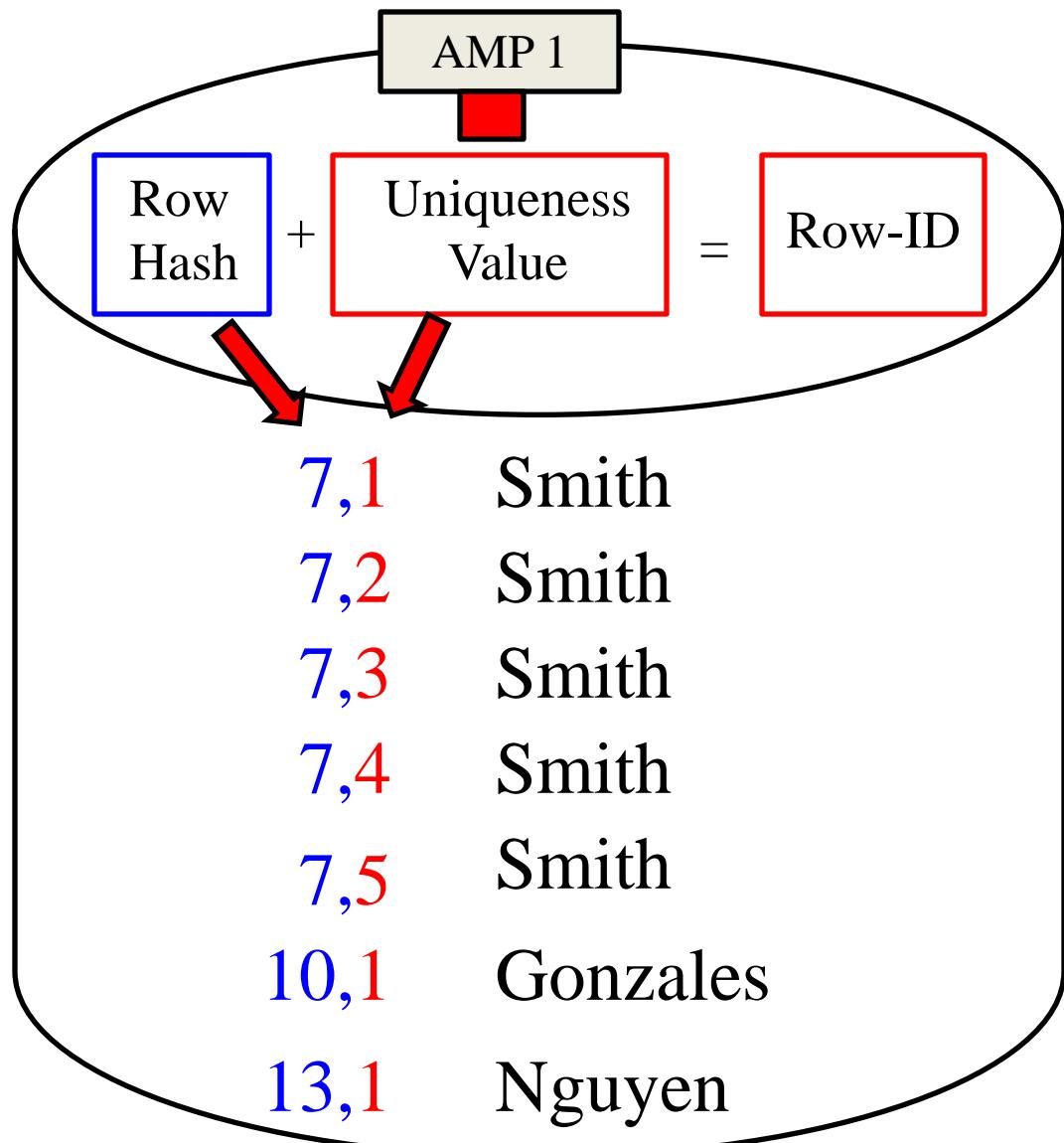


Row Hash and the Uniqueness Value make up the Row-ID. AMPS sort by the Row-ID.

Two Reasons why each AMP Sorts their rows by the Row-ID

The Two Reasons are:

- 1) So like Data such as ‘Smith’ is grouped together on the AMP’s disk.
- 2) So each AMP can perform a **Binary Search**, like a Phone Book in **Alphabetical order**.



AMPs sort rows by Row-ID so like data is grouped together and for Binary searches.

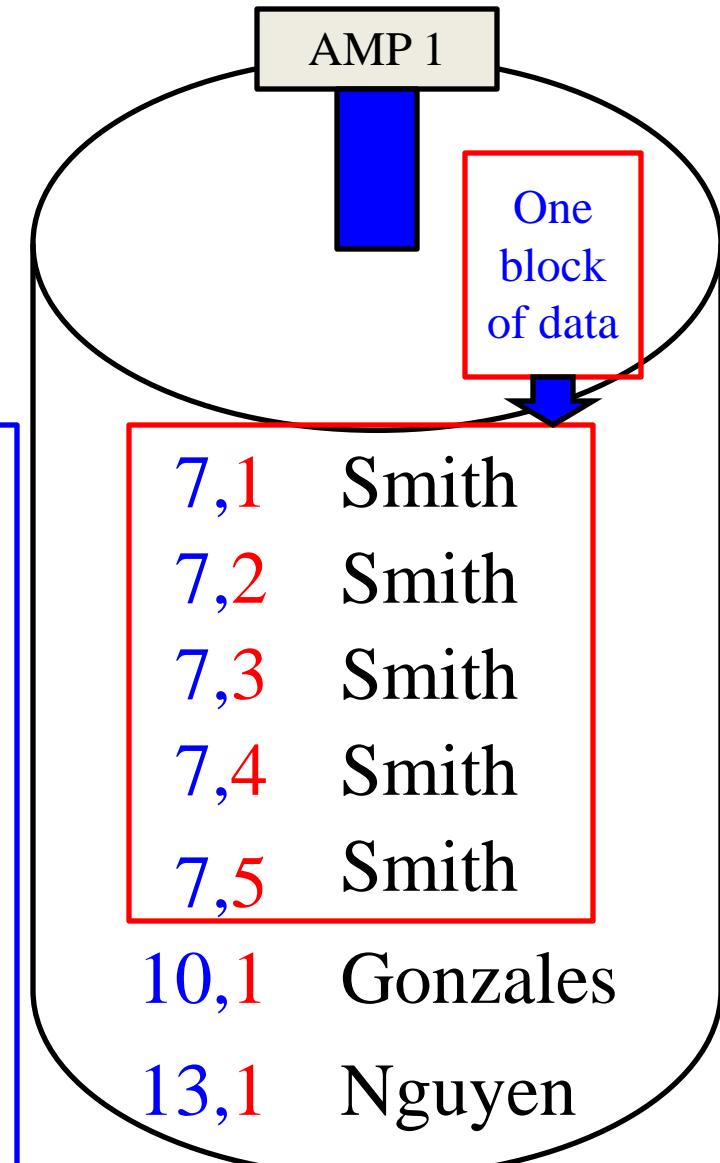
AMPs sort their rows by Row-ID to Group Like Data

Query

```
SELECT * FROM Employee_Table  
WHERE Last_Name = 'Smith';
```

Parsing Engine (PE) Plan

- 1) I see that Last_Name is the Primary Index.
- 2) So Hash 'Smith' and get the Row Hash.
- 3) The Row Hash for 'Smith' = 7.
- 4) What AMP# is in Bucket 7 of the Hash Map?
- 5) Bucket 7 says AMP 1.
- 6) Bynet - tell AMP 1 to all retrieve Row Hash 7's.
- 7) Bring back all Columns for Row Hash 7 ('Smith').



Notice that all of the Smiths are lumped together because of the sorting by Row-ID.

AMPs sort their rows by Row-ID to do a Binary Search

Query

```
SELECT * FROM Order_Table  
WHERE Order_Number = 50;
```

- 1) Hash Order_Number 50 (Row Hash = 75)
- 2) Go to bucket 75 in Hash Map
- 3) What AMP is in bucket 75? AMP 1!
- 4) Bynet – Tell AMP 1 to get Row Hash 75!
- 5) Perform a Binary Search

Start Binary Search in the Middle

Are you Row Hash 75?

No – Too Low

Move down half way more

Are you Row Hash 75?

No – Now Too High

Row-ID	Order Number	Order Date	Order Total
10,1	1	'2012-06-30'	1270.87
13,1	5	'2012-07-01'	2356.17
16,1	12	'2012-07-05'	2557.27
22,1	15	'2012-07-08'	2145.33
31,1	20	'2012-07-10'	2265.32
40,1	25	'2012-07-12'	2480.43
52,1	30	'2012-07-15'	4390.23
61,1	35	'2012-07-18'	3266.43
70,1	42	'2012-07-20'	2455.23
75,1	50	'2012-08-02'	2575.87
80,1	55	'2012-08-06'	3400.65
82,1	60	'2012-08-08'	3523.54

A Binary Search knows the Row-IDs are in numeric order . Its like you using a phone book. Go to the middle first and then go up or down in chunks to find things quickly.

Table CREATE Examples with four different Primary Indexes

1

```
CREATE TABLE Emp_Intl  
(Emp_No          INTEGER,  
 Dept_No         SMALLINT,  
 First_Name      VARCHAR(12),  
 Last_Name       CHAR(20),  
 Salary          DECIMAL(10,2))  
UNIQUE PRIMARY INDEX ( Emp_No );
```

UPI

2

```
CREATE TABLE Emp_Intl  
(Emp_No          INTEGER,  
 Dept_No         SMALLINT,  
 First_Name      VARCHAR(12),  
 Last_Name       CHAR(20),  
 Salary          DECIMAL(10,2))  
PRIMARY INDEX ( Dept_No );
```

NUPI

3

```
CREATE TABLE Emp_Intl  
(Emp_No          INTEGER,  
 Dept_No         SMALLINT,  
 First_Name      VARCHAR(12),  
 Last_Name       CHAR(20),  
 Salary          DECIMAL(10,2))  
PRIMARY INDEX ( First_Name , Last_Name );
```

Multi-Column NUPI

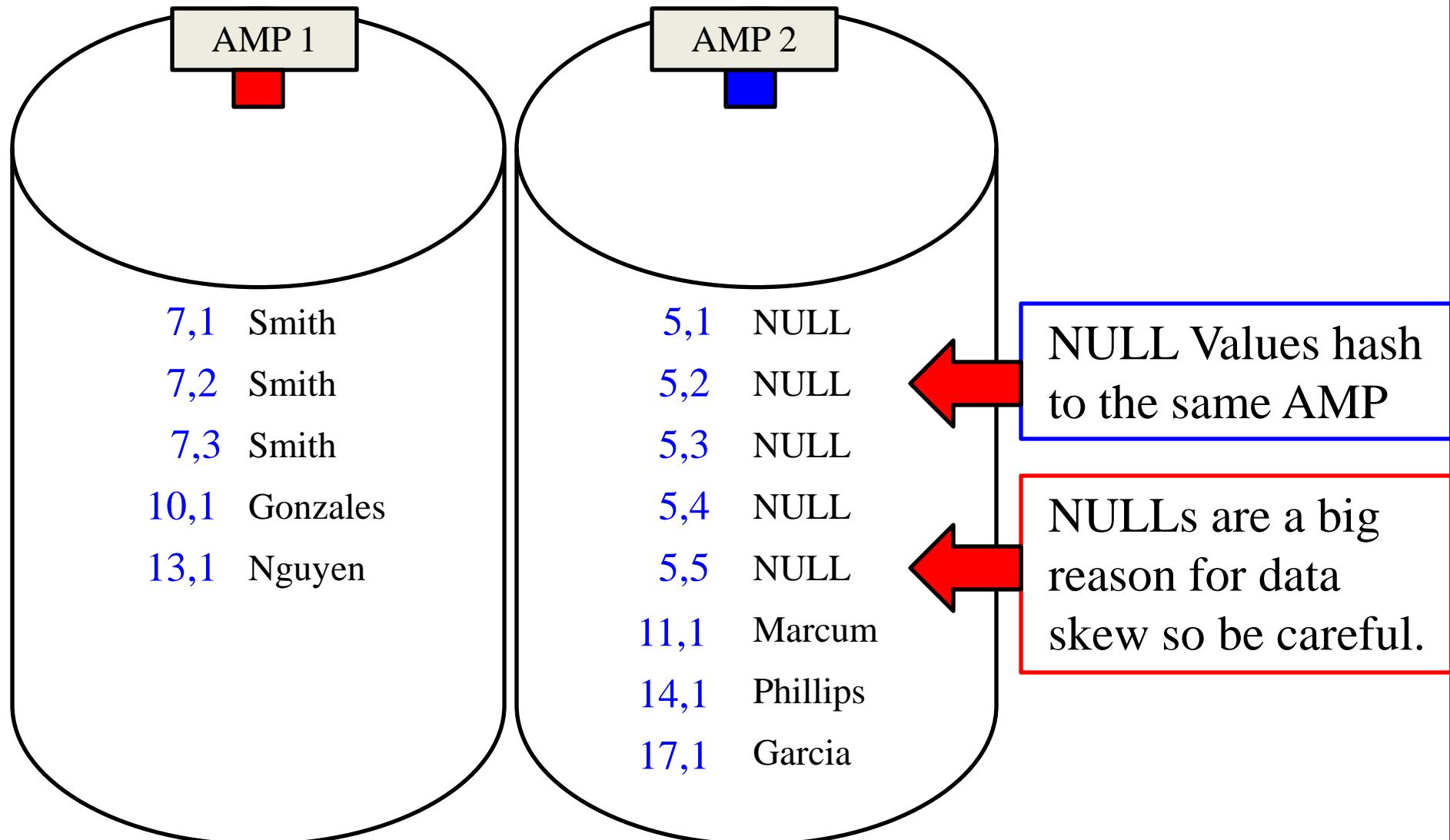
4

```
CREATE TABLE Emp_Intl  
(Emp_No          INTEGER,  
 Dept_No         SMALLINT,  
 First_Name      VARCHAR(12),  
 Last_Name       CHAR(20),  
 Salary          DECIMAL(10,2))  
NO Primary Index
```

No Primary Index

A table can have only one Primary Index so picking the right one is essential. Above are four different examples for your consideration. Now lets review with these.

Null Values all Hash to the Same AMP



If there are NULL values in the Primary Index you could find this is the reason for your skew. A Table with a Unique Primary Index can have only one Null value, but a NUPI table can have many NULL values and each NULL value hashes to the same AMP.

A Unique Primary Index (UPI) Example

1

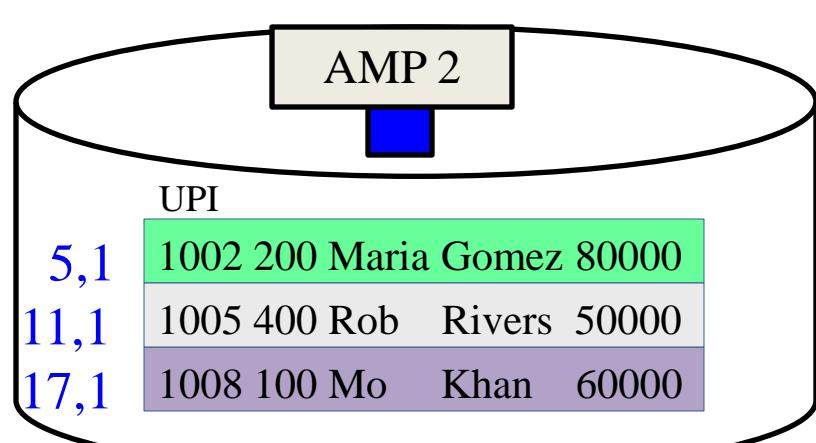
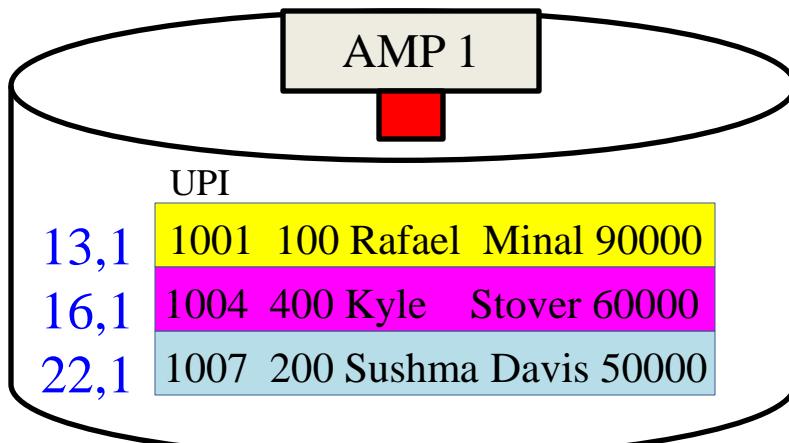
```
CREATE TABLE Emp_Intl
(Emp_No          INTEGER,
Dept_No         SMALLINT,
First_Name      VARCHAR(12),
Last_Name       CHAR(20),
Salary          DECIMAL(10,2))
UNIQUE PRIMARY INDEX ( Emp_No );
```

PE Explain Plan

Fastest Query
possible
in Teradata!

```
SELECT *
FROM Emp_Intl
WHERE Emp_No = 1001;
```

A Single-AMP Retrieve
by way of a
Unique Primary Index



A Unique Primary Index will spread the data perfectly evenly.

A Non-Unique Primary Index (NUPI) Example

2

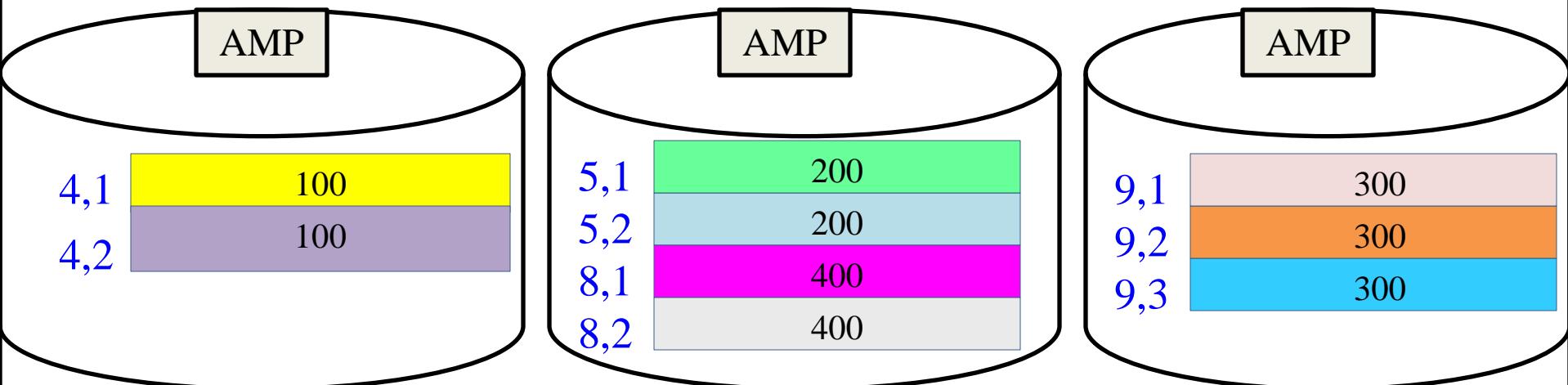
```
CREATE TABLE Emp_Intl  
(Emp_No          INTEGER,  
Dept_No         SMALLINT,  
First_Name      VARCHAR(12),  
Last_Name       CHAR(20),  
Salary          DECIMAL(10,2))  
PRIMARY INDEX ( Dept_No );
```

2nd Fastest
Query
possible
in Teradata!

```
SELECT *  
FROM Emp_Intl  
WHERE Dept_No = 300;
```

PE Explain Plan

A Single-AMP Retrieve
by way of a
Primary Index



A Non-Unique Primary Index will NOT spread the data perfectly evenly.

A Multi-Column Primary Index Example

3

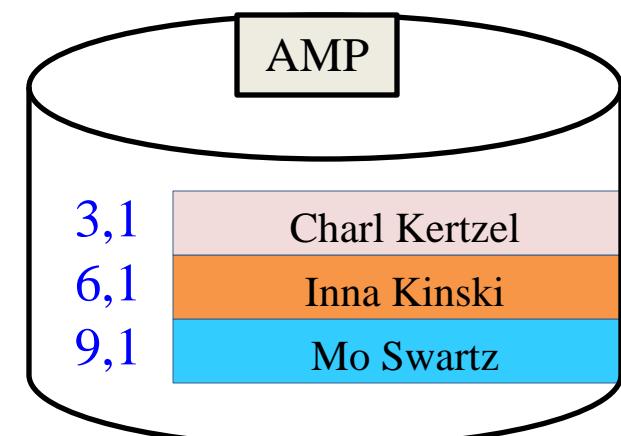
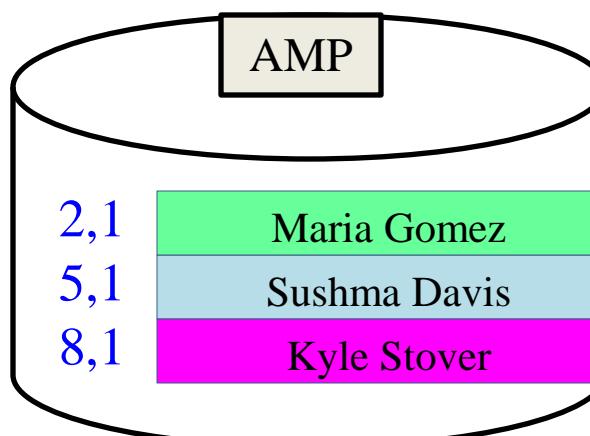
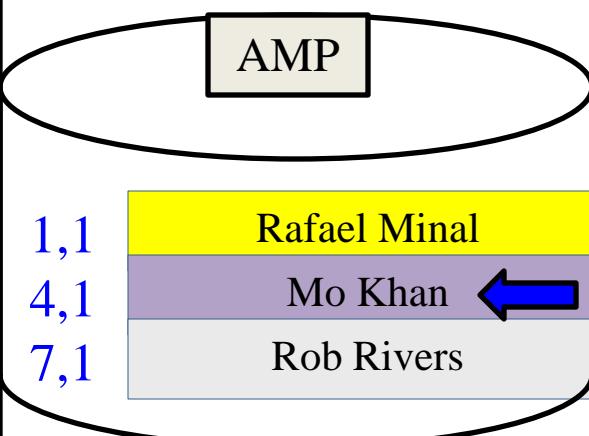
```
CREATE TABLE Emp_Intl
(Emp_No          INTEGER,
Dept_No         SMALLINT,
First_Name      VARCHAR(12),
Last_Name       CHAR(20),
Salary          DECIMAL(10,2))
PRIMARY INDEX ( First_Name , Last_Name );
```

2nd Fastest
Query
possible
in Teradata!

```
SELECT *
FROM Emp_Intl
WHERE First_Name = 'Mo'
AND Last_Name = 'Khan'
```

PE Explain Plan

A Single-AMP Retrieve
by way of a
Primary Index



A Multi-Column Primary Index is often used to fix a data skew problem.

A No Primary Index (NoPI) Example

4

```
CREATE TABLE Emp_Intl  
(Emp_No          INTEGER,  
 Dept_No         SMALLINT,  
 First_Name      VARCHAR(12),  
 Last_Name       CHAR(20),  
 Salary          DECIMAL(10,2))
```

NO Primary Index

Slowest
Query
in Teradata!

```
SELECT *  
FROM Emp_Intl  
WHERE Emp_No = 1001 ;
```

PE Explain Plan

A Full Table Scan!
All-AMP Retrieve by way
of an All-Rows scan

AMP 1

AMP 2

AMP 3

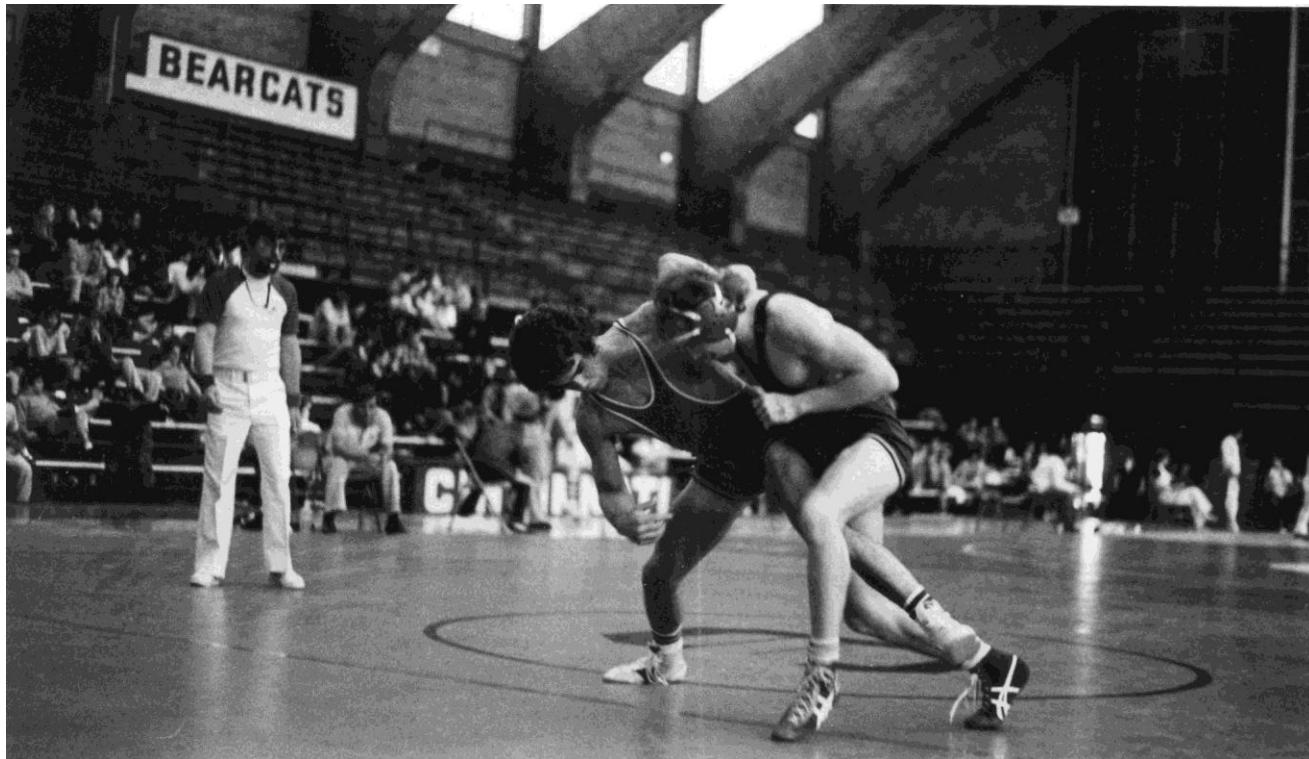
1,1	1001	100	Rafael	Minal	90000
1,2	1004	400	Kyle	Stover	60000
1,3	1007	200	Sushma	Davis	50000

2,1	1002	200	Maria	Gomez	80000
2,2	1005	400	Rob	Rivers	50000
2,3	1008	100	Mo	Khan	60000

3,1	1003	300	Charl	Kertzel	80000
3,2	1006	300	Inna	Kinski	50000
3,3	1009	300	Mo	Swartz	70000

All AMPs read all of their rows (full table scan) because there is no Primary Index.

Watch the Video on the Hashing the Data



Tera-Tom Trivia

In 1980 Tom Coffing won two Olympic Trials before the United States boycotted the Olympics.
Tom is pictured above as the wrestler on the left.

Click on the **link below** or place it in your browser and watch the video on Hashing.

<http://www.coffingdw.com/TbasicsV12/Hashing.wmv>

Chapter 4

Space

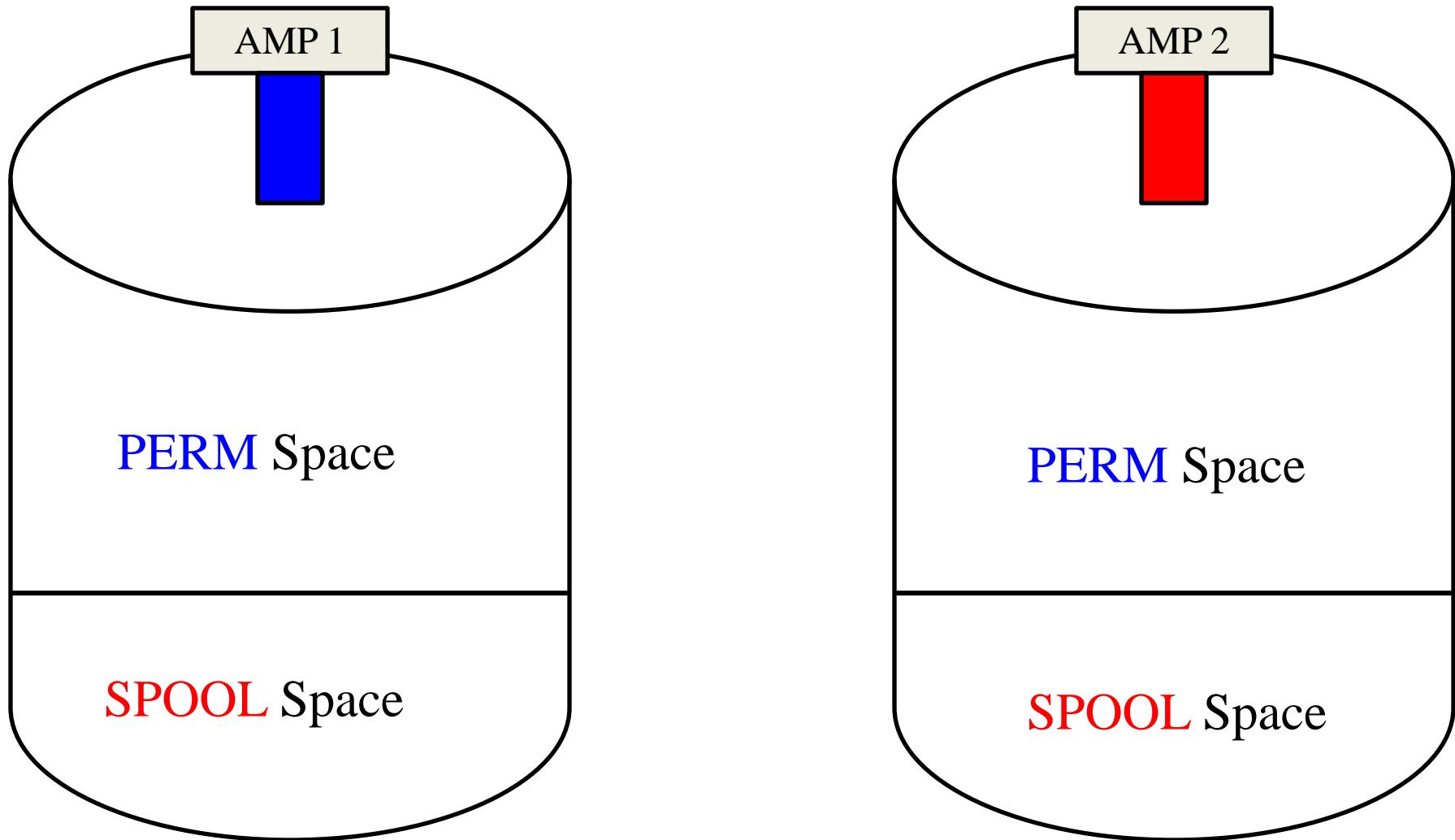
“For the wise man looks into space and he knows there are no limited dimensions.”

- Lao-tzu

Table of Contents Chapter 4 - Space

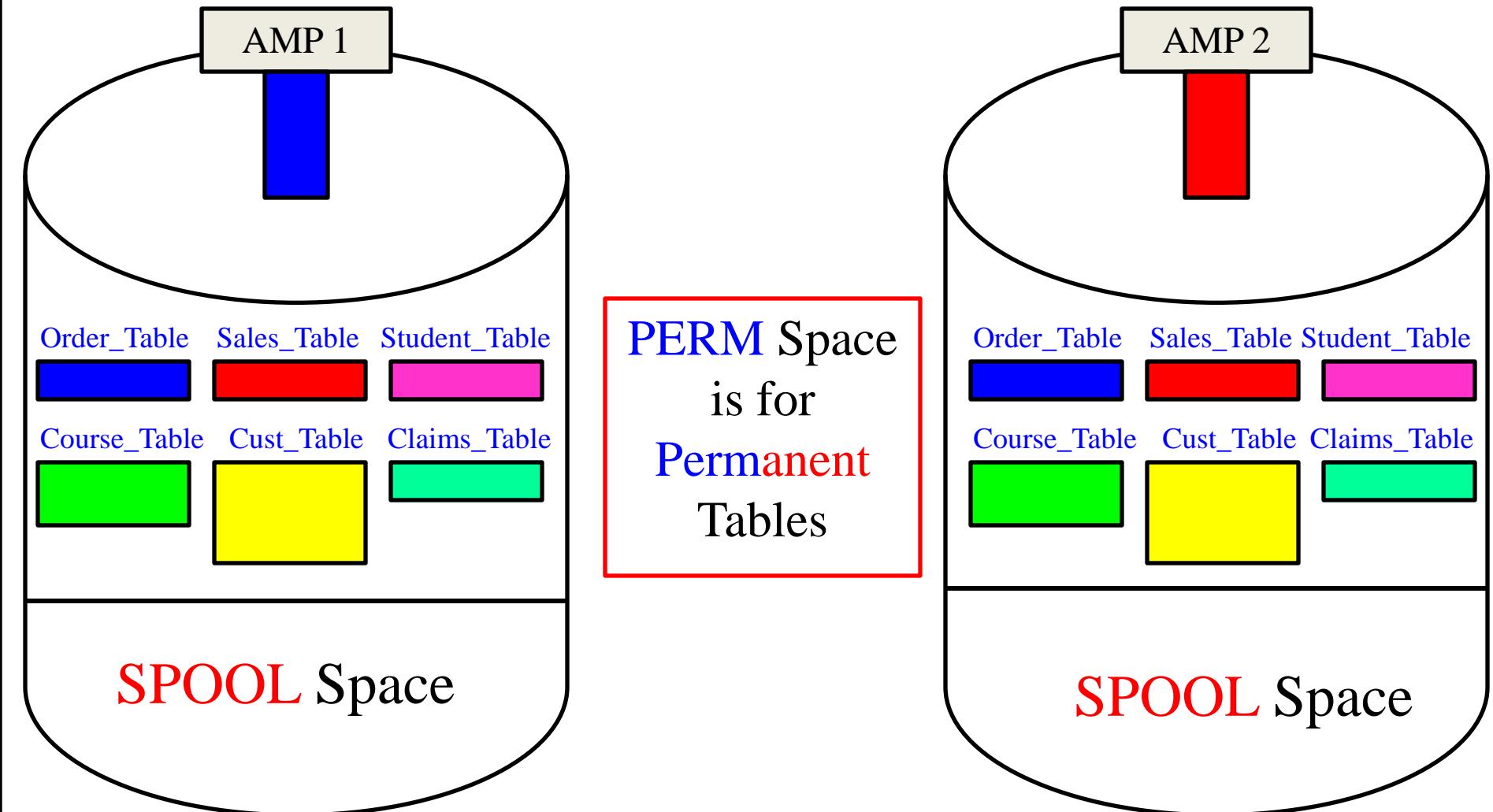
- [Perm and Spool Space](#)
- [Perm Space is for Permanent Tables](#)
- [Spool Space is work space that builds a User's Answer Sets](#)
- [Spool Space is in an AMPs memory and on its Disk](#)
- [USERs are Assigned Spool Space Limits](#)
- [What is the Purpose of Spool Limits?](#)
- [Why did my query Abort and say "Out of Spool"?](#)
- [How can Skewed Data cause me to run "Out of Spool"?](#)
- [How come my Join caused me to run "Out of Spool"?](#)
- [What does my system look like when it first arrives?](#)
- [DBC owns all the PERM Space in the system on day one](#)
- [DBC's First Assignment is Spool Space](#)
- [DBC's 2nd Assignment is to CREATE Users and Databases](#)
- [The Teradata Hierarchy Begins](#)
- [The Teradata Hierarchy Continues](#)
- [Differences between PERM and SPOOL](#)
- [Databases, Users, and Views](#)
- [What are Similarities between a DATABASE and a USER?](#)
- [What is the Difference between a DATABASE and a USER?](#)
- [Objects that take up PERM Space](#)
- [A Series of Quizzes on Adding and Subtracting Space](#)
- [Answer 1 to Quiz on Space](#)
- [Space Transfer Quiz](#)
- [Answer to Space Transfer Quiz](#)
- [Drop Space Quiz](#)
- [Answers to Drop Space Quiz](#)
- [Watch the Video on Space](#)

Perm and Spool Space



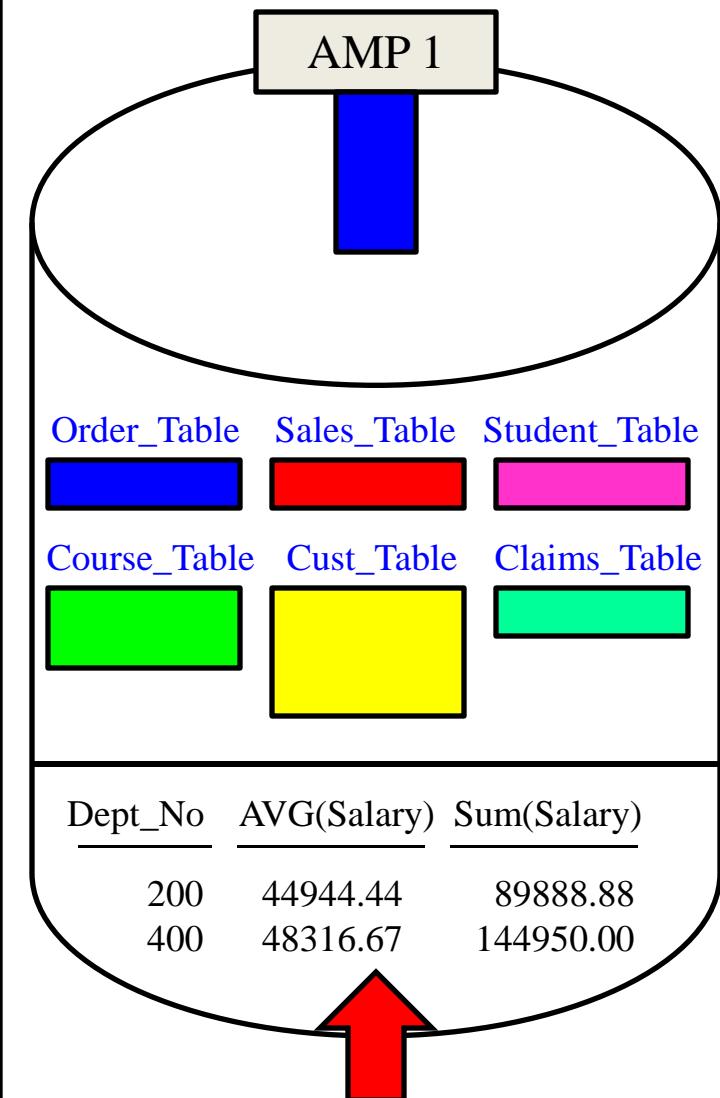
Space has only to do with space on the data warehouse disks. Each AMP controls its own disk farm (each AMP is attached to four physical disks) and about 60% of each disk will be used for tables and that is called PERM space. The other 40% (Spool) is work space for user queries and answer sets. No AMP can get into another AMP's disk.

Perm Space is for Permanent Tables

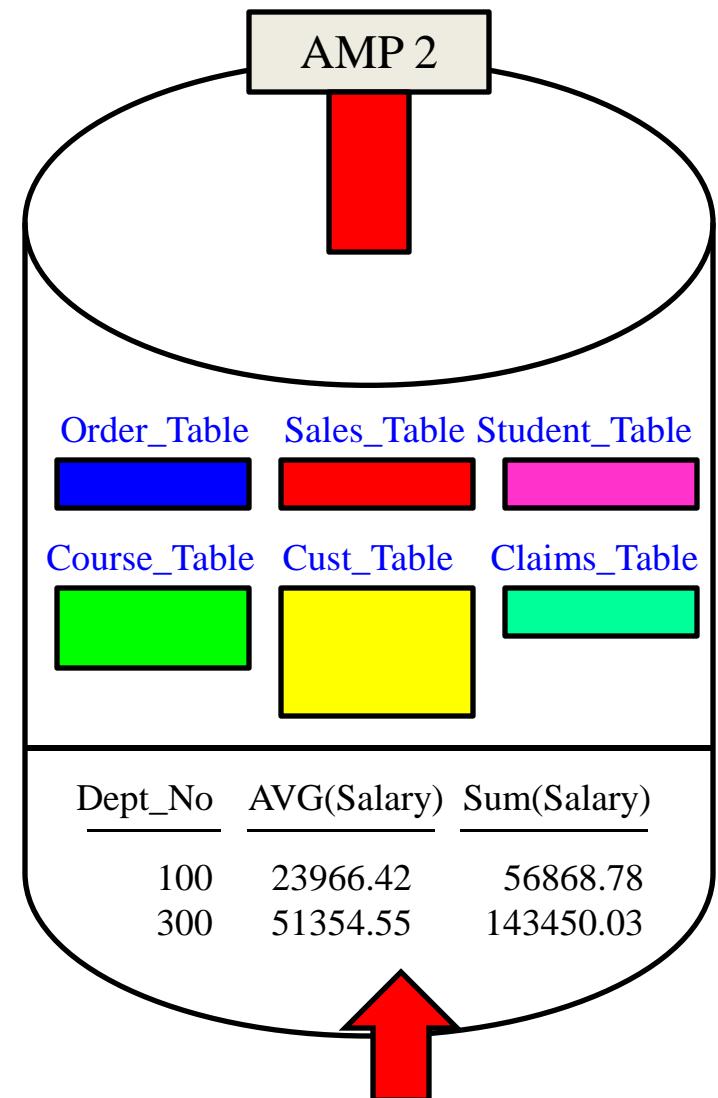


PERM Space is where an AMP keeps its tables. That is what you need to understand. You will also find out that PERM space also houses Secondary Indexes, Join Indexes and Permanent Journals. Just remember that PERM is for the tables and indexes! Notice the different sizes of tables. This is because tables are different sizes.

Spool Space is work space that builds a User's Answer Sets

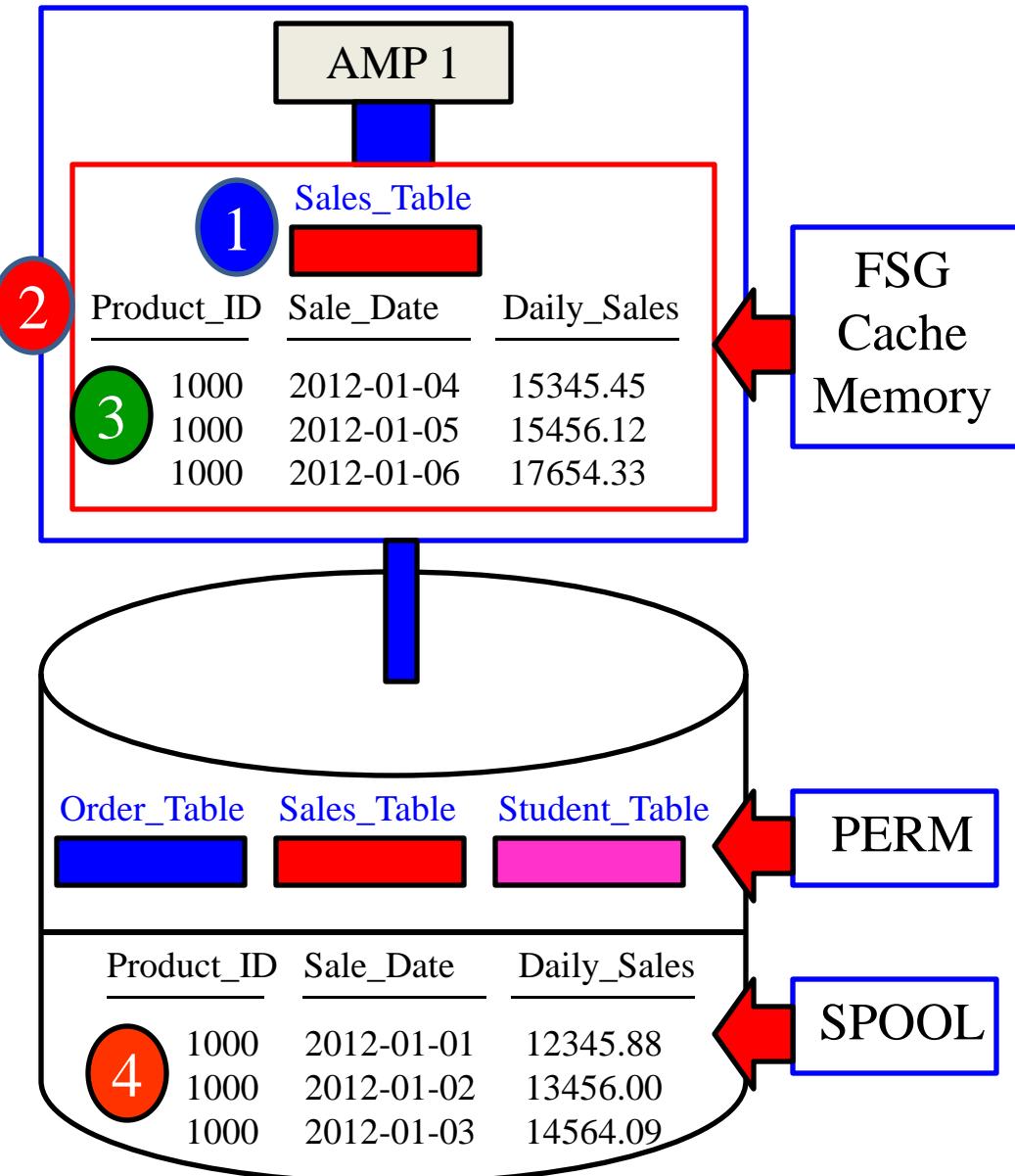


Spool is used by the AMPS as workspace to build a User's Answer Sets



Spool space is used by each AMP in order to build the answer set for the user.

Spool Space is in an AMP's Memory and on its Disk



1 Transfer the Sales_Table from the disk (Perm) to FSG Cache.

2 Get the Product_ID, Sale_Date and the Daily_Sales columns.

3 Build the Report in FSG Cache.

4 If there is no more room in FSG Cache than transfer the report to Spool on Disk.

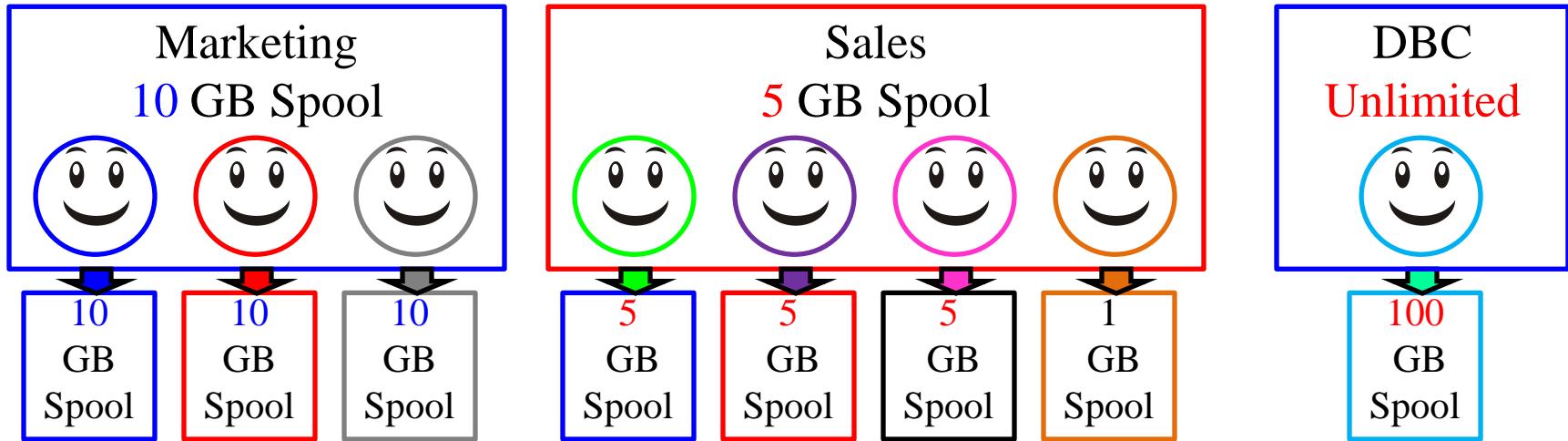
5 Keep checking if the USER has gone over their SPOOL Limit.

6 The Report is done so transfer the report to the Parsing Engine over the BYNET.

7 DELETE the Spool Files.

AMPs have memory called File System Generating Cache (FSG) used for processing.

Users are Assigned Spool Space Limits



Every User is assigned Spool Space so they can submit SQL and retrieve answer sets.

The Spool in the database Marketing is 10 GB so each user defaults to 10 GB of Spool.

Any User in Marketing can run queries, but are aborted if they go over the 10 GB limit.

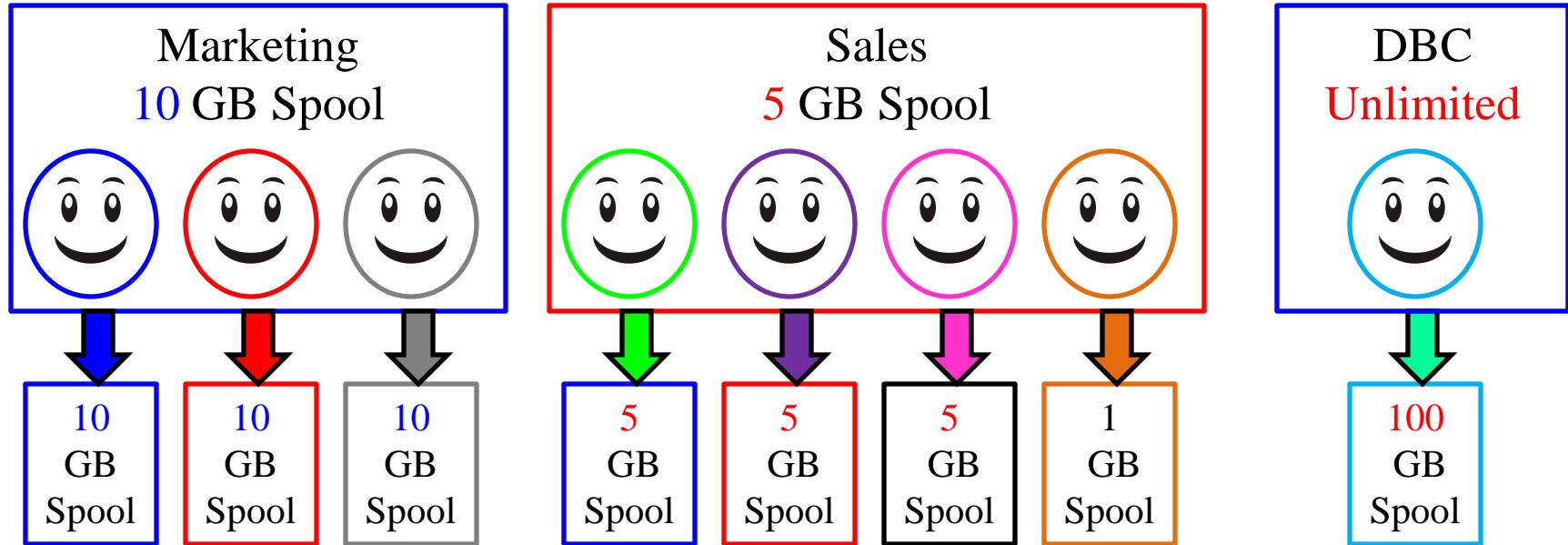
All 3 users in Marketing can query simultaneously and use 30 GB of Spool in total .

Three users in Sales defaulted to the max (5 GB) but the intern was assigned less.

The final user in DBC was given 100 GB of Spool because they are brilliant.

Spool is assigned to users and the only way you are aborted is if you go over your spool limit. Marketing isn't restricted in total to 10 GB of spool, but it is the individual max. Just like a speed limit of 60 MPH on the highway. Each car can go 60 MPH and each User in Marketing can query at a max speed limit of 10 GB of spool.

What is the Purpose of Spool Limits?

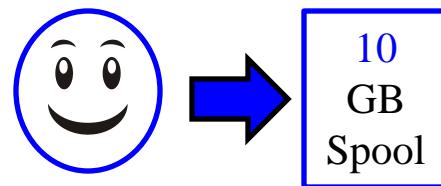


There are two reasons for Spool Limits:

1. If a user makes a mistake and runs a query that could take weeks to run it will abort as soon as the user goes over their allotted spool limit.
2. It keeps users from hogging the system.

Spool is assigned to users and the only way a user is aborted is if they go over their spool limit. **Marketing, Sales, and DBC** have **unlimited spool**, but the max for each individual user is 10 GB in Marketing, 5 GB in Sales, and our power user is at 100 GB.

Why did my query Abort and say “Out of Spool”?



How is it possible that I ran out of spool?

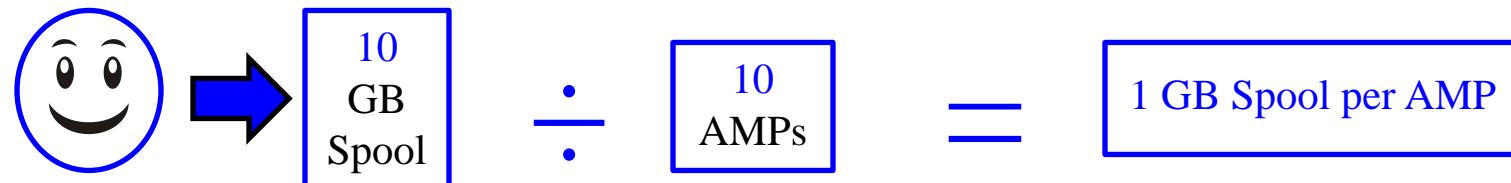
You ran out of spool because your query used over **your limit** of **10 GB** of spool.

It is also possible that you have logged onto multiple machines or ran multiple queries and the **combination** went over 10 GB of spool.

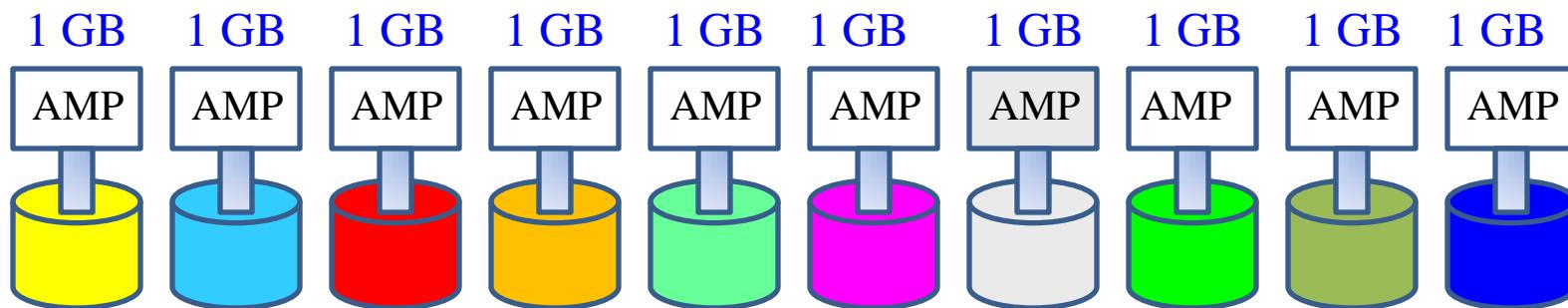
It is also very likely that the data you were working with was **NOT** evenly distributed (**skewed**) and this is a major cause of Spool errors.

Spool is assigned to users and the only way a user is aborted is if they go over their spool limit. No user has ever failed because they are in Marketing and Marketing has only 10 GB of spool. It doesn't work that way. Thousands of users in Marketing could run queries simultaneously because Marketing has unlimited amounts of spool, but each user in Marketing is limited to 10 GBs individually.

How can Skewed Data cause me to run “Out of Spool”?



Each User's Spool limit is actually done per AMP so if you are assigned **10 GBs** of spool and the system has **10 AMPs** you are really assigned **1 GB** of Spool per AMP!



If data **is skewed** and you exceed your 1 GB limit on any AMP you are “out of spool”.

Spool is assigned to every user, but since Teradata is a parallel processing system each AMP is only concerned with themselves. Each AMP processes its portion of the data in parallel. Because of this philosophy your Spool Space (10 GB) is divided among the total AMPs in the system. If you have 10 GBs of Spool and there are 10 AMPs you get 1 GB per AMP. If you go over 1 GB on any AMP you are aborted and “Out of Spool”.

How come my Join caused me to run “Out of Spool”?

1

You might not have put in a Join Condition.

```
SELECT First_Name, Last_Name, Department_Name  
FROM Employee_Table as E  
      INNER JOIN  
          Department_Table as D
```

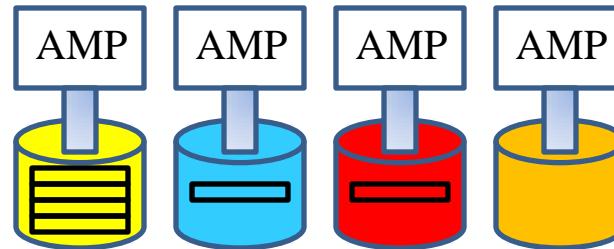
2

You might have Aliased the table and then fully qualified with the real table name.

```
SELECT First_Name, Last_Name, Department_Name  
FROM Employee_Table as E  
      INNER JOIN  
          Department_Table as D  
ON Employee_Table.Dept_No = D.Dept_No ;
```

3

There might be skewed data on one of the tables.

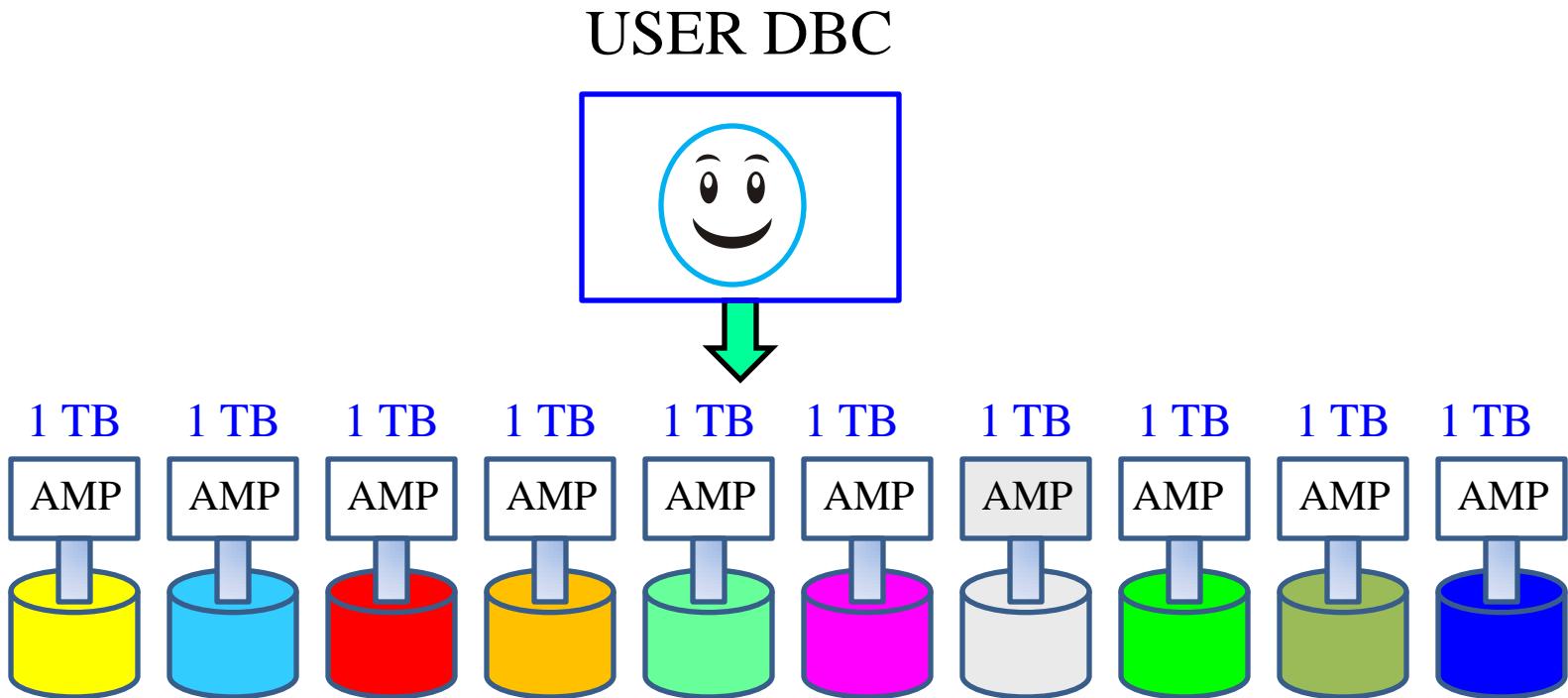


4

A Lot of NULLs on a table on an Outer Join.

```
SELECT e.*, d.* from Employee_Table as E  
      LEFT OUTER JOIN Department_Table as D  
      ON E.Dept_No= D.Dept_No ;
```

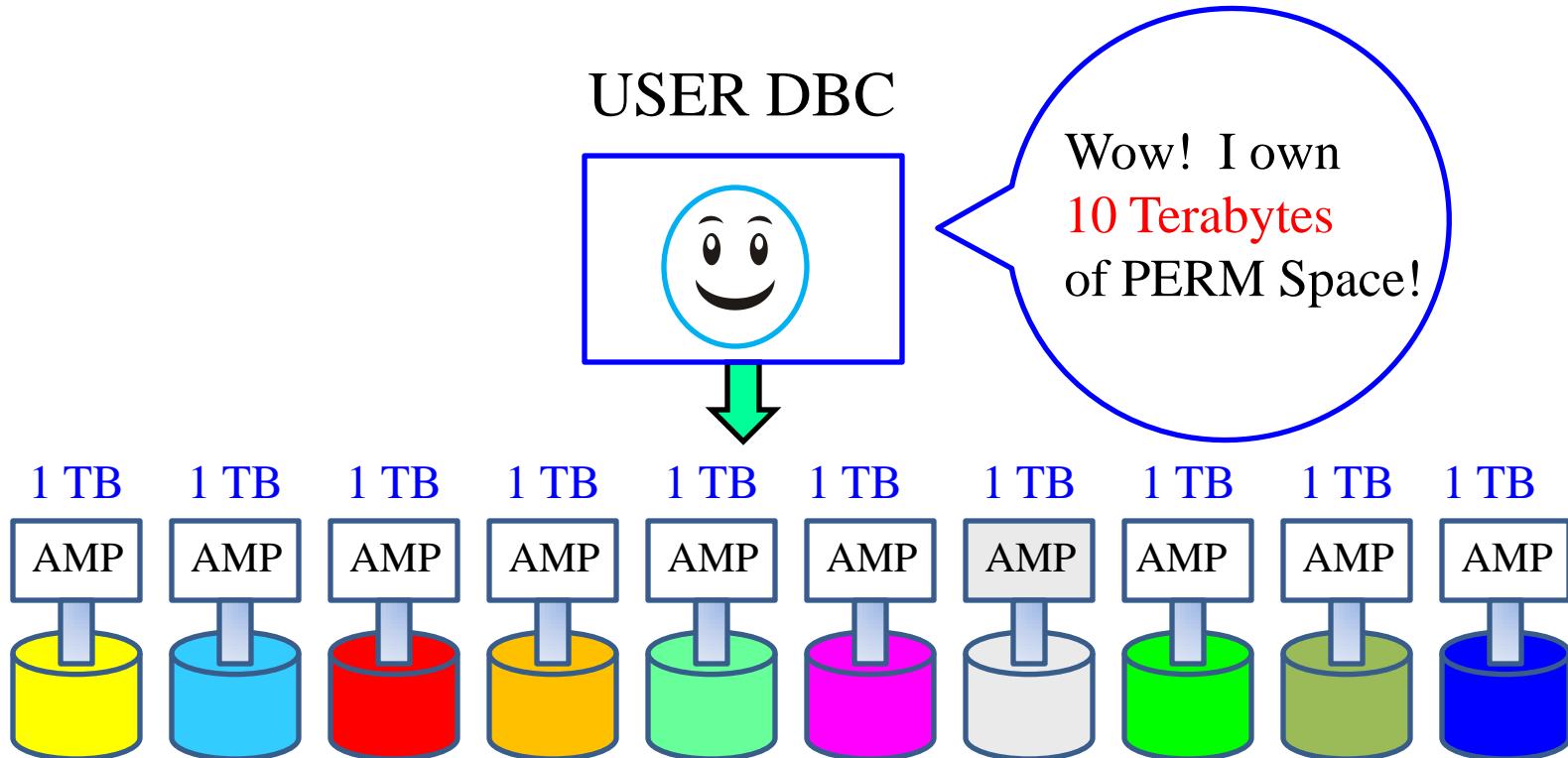
What does my system look like when it first arrives?



All Teradata systems start with one USER called **DBC**.

The first Teradata machine ever built came out in 1988 and it was called the DBC 1012. The DBC portion stood for Database Computer. The 1012 was named because 10^{12} power is equal to a Terabyte. So, the DBC 1012 was a Database Computer designed to process Terabytes of data. So, every system starts with one USER called DBC and DBC owns all the PERM Space in the system.

DBC owns all the PERM Space in the system on day one



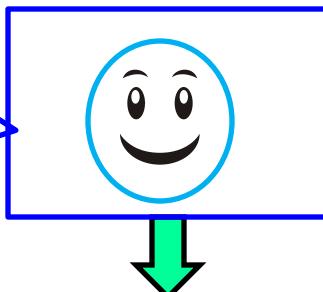
When the system starts out new and arrives at your company **DBC** is the only **USER**. DBC counts up all the **disk space** attached to each **AMP** and considers that **PERM** space owned by DBC.

DBC owns all the disk space on day one of your system's arrival. DBC will then begin to allocate space to other databases or users. Think of PERM space like money. If DBC has 10 Terabytes of space it is like having 10 dollars. If you give away 5 dollars you only have 5 dollars left. Spool space is more like a speed limit.

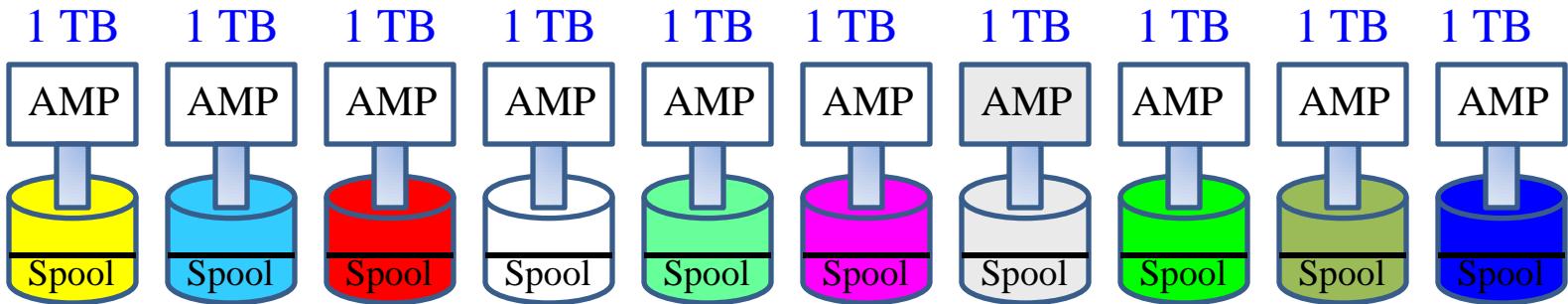
DBC's First Assignment is Spool Space

I will CREATE a Database called **Spool_Reserve** to ensure Spool Space for everyone!

USER DBC



```
CREATE DATABASE  
Spool_Reserve  
FROM DBC  
AS  
PERM = 40000000000000  
SPOOL = 40000000000000;
```



DBC will create a database called **Spool_Reserve** (any name will do), but it will reserve between 20% to 40% for Spool. What really happens is that DBC creates Spool_Reserve to claim **PERM** Space, but never places a table in the database.

When a database is given PERM Space and no object is created in that database it is used for Spool. **Spool is unused PERM!**

DBC's 2nd Assignment is to CREATE Users and Databases

I will now
CREATE USERS
and/or
DATABASES

USER DBC



```
CREATE USER Retail
  FROM DBC
  AS
    PASSWORD=abc123
    PERM=200000000000
    SPOOL=100000000000
    TEMPORARY = 100000000000
    ACCOUNT='$Med'
    DEFAULT DATABASE = DBC ;
```

```
CREATE USER Financial
  FROM DBC
  AS
    PASSWORD=abc123
    PERM=200000000000
    SPOOL=5000000000
    TEMPORARY = 100000000000
    ACCOUNT='$Med'
    DEFAULT DATABASE = DBC ;
```

DBC's 2nd assignment will be to create some USERS or DATABASES and the hierarchy begins. If a USER or DATABASE is assigned PERM space it can CREATE tables.

The Teradata Hierarchy Begins

1

USER DBC



10 TB PERM

When the system first arrives, **DBC** owns **all** PERM Space.

On day one, DBC owns **10 TB** of PERM in this **10 TB** system.

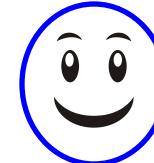
2

USER DBC



2 TB PERM

USER
Retail



2 TB PERM

DATABASE
Spool_Reserve



4 TB PERM

USER
Financial

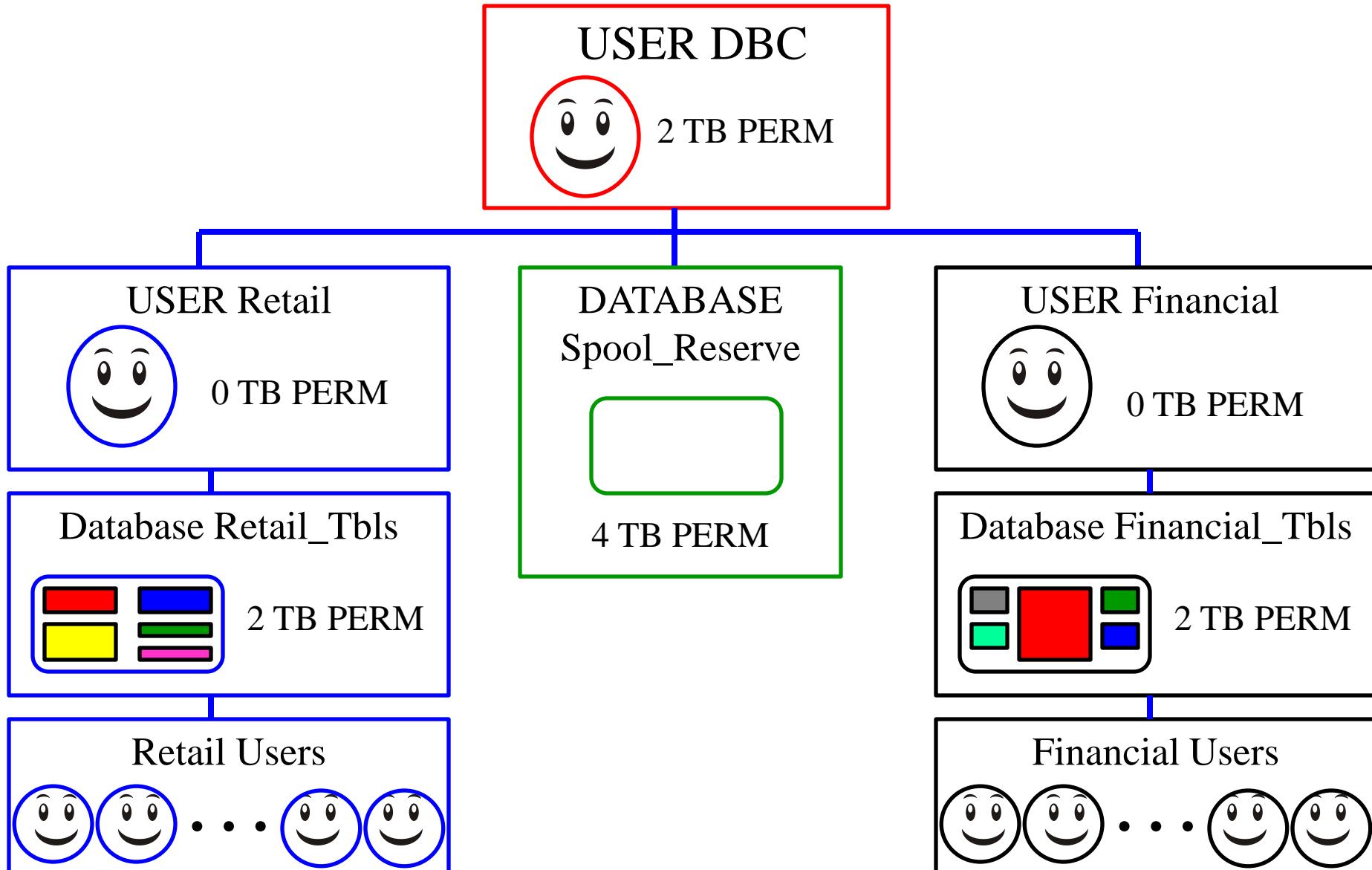


2 TB PERM

The PERM Space is dispersed among the USERS and DATABASEs with each CREATE statement.

Notice in example 1 that DBC owns 10 TB of PERM space. Notice that after DBC created Spool_Reserve (4 TB), USER Retail (2 TB) and USER Financial (2 TB) that DBC now only owns only 2 TB of PERM space.

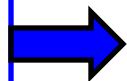
The Teradata Hierarchy Continues



USER Retail and USER Financial now create the databases and users desired.

Differences between PERM and SPOOL

Retail can
create and load
up to **2 TB** of
Tables and Data.



USER Retail

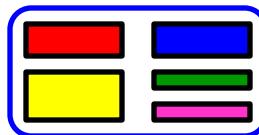
2 TB PERM

10 GB SPOOL

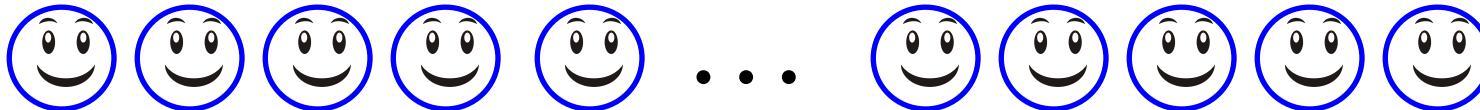


Retail's Speed Limit.
Every USER under Retail
can run queries up to **10
GB** simultaneously!

Retail has Tables under them

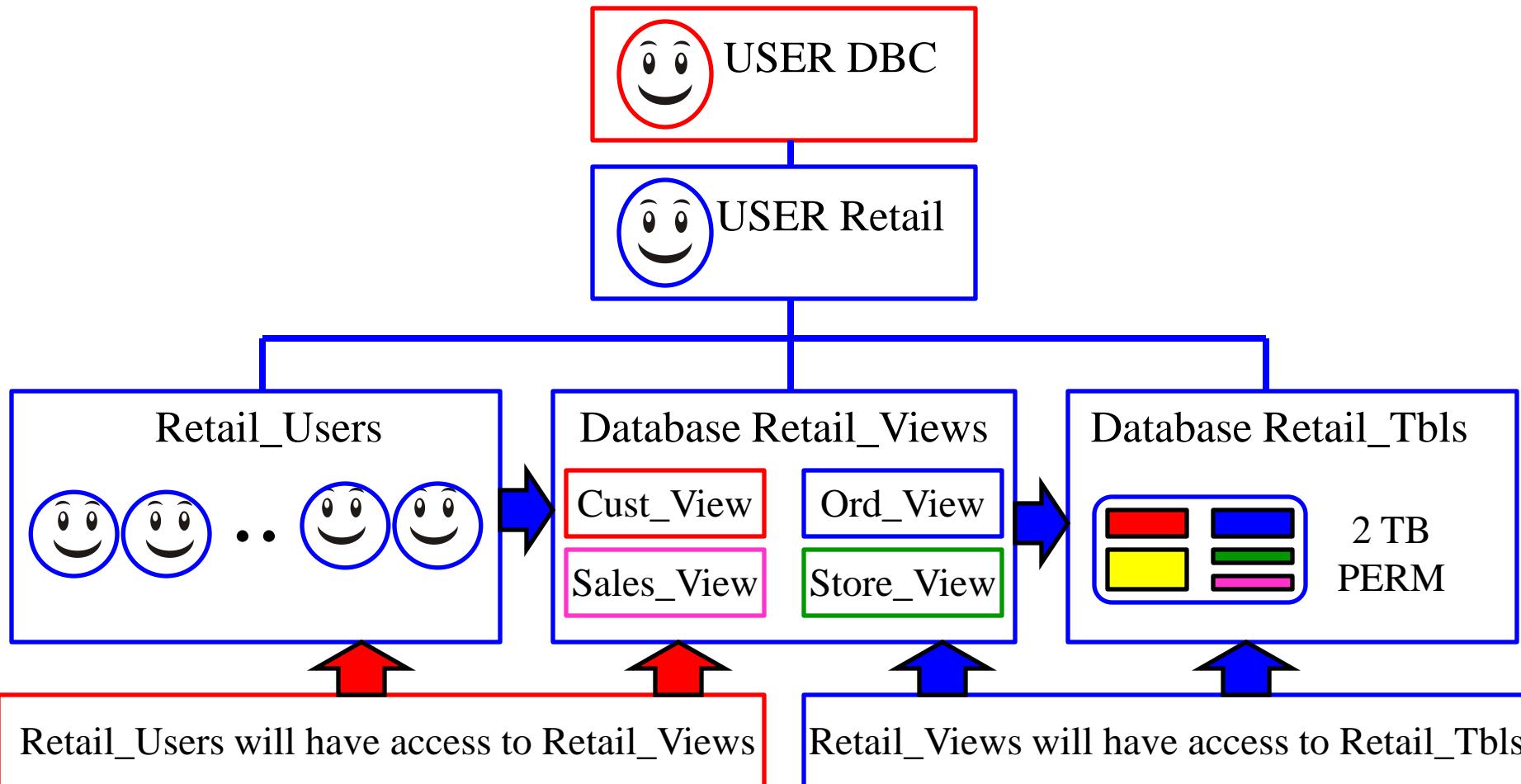


Retail has 1,000 Users under them



There are 1,000 users in Retail. Since Retail has 10 GB of spool that means that every user gets 10 GB of spool. That is the maximum limit for Retail. What it does **NOT** mean is that Retail is limited to only 10 GB of spool in total. Every user could logon and run a 9 GB query taking up Terabytes of Spool and nobody would run out of spool. Spool is system wide and calculated on an individual level only.

Databases, Users, and Views



For security purposes the Retail tables will be kept in their own database called Retail_Tbls (in this example). The general Retail User Population will NOT have access directly to these tables. A Database called Retail_VIEWS houses the views that access the tables. So, the DBA will create Access Rights that allow the views to read the tables and the Users to SELECT from the views.

What are Similarities between a DATABASE and a USER?



1 A Database or a User can be assigned PERM Space

A If the Database **Marketing** is assigned 10 GB of PERM that means it can hold up to 10 GB of Permanent Tables.

B If the User **Maria** is assigned 10 GB of PERM that means she can hold up to 10 GB of Permanent Tables.

2 A Database or a User can be assigned Spool Space

A If the Database **Marketing** is assigned 10 GB of Spool that means all users under marketing can each run 10 GB queries.

B If the User **Maria** is assigned 10 GB of Spool that means she can run up to 10 GB queries and any user created under Maria will default to 10 GB queries.

What is the Difference between a DATABASE and a USER?



Database
Marketing



USER
Maria

A USER has a login and password and therefore can
run queries

Objects that take up PERM Space

Permanent Space (Perm space) is the maximum amount of storage assigned to a user or database for holding:

- Table Rows
- Fallback Tables
- Secondary Index Subtables
- Stored Procedures
- User Defined Functions (UDFs)
- Permanent Journals

Views and Macros do NOT take up any Perm Space!

A Series of Quizzes on Adding and Subtracting Space

Marketing

10 GB Perm

10 GB Spool

Sales

5 GB Perm

5 GB Spool

- 1 Marketing has 10 GB of Perm and Spool. Sales has 5 GB Perm and Spool.
- 2 Marketing then Creates Stan and gives him 1 GB Perm and 10 GB Spool.
- 3 Sales then Creates Mary and gives her 1 GB Perm and 5 GB Spool.

Marketing

Stan

1 GB Perm

10 GB Spool

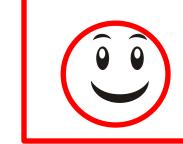


Sales

Mary

1 GB Perm

5 GB Spool



After creating users how much Perm / Spool is in Marketing and how much is in Sales?

Answer 1 to Quiz on Space

Marketing

10 GB Perm

10 GB Spool

Sales

5 GB Perm

5 GB Spool

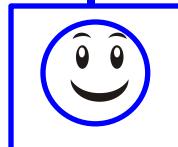
- 1 Marketing has 10 GB of Perm and Spool. Sales has 5 GB Perm and Spool.
- 2 Marketing then Creates Stan and gives him 1 GB Perm and 10 GB Spool.
- 3 Sales then Creates Mary and gives her 1 GB Perm and 5 GB Spool.

Marketing

Stan

1 GB Perm

10 GB Spool

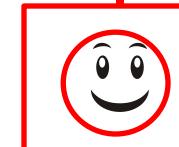


Sales

Mary

1 GB Perm

5 GB Spool



After creating users how much Perm / Spool is in Marketing and how much is in Sales?

9 GB Perm

10 GB Spool

4 GB Perm

5 GB Spool

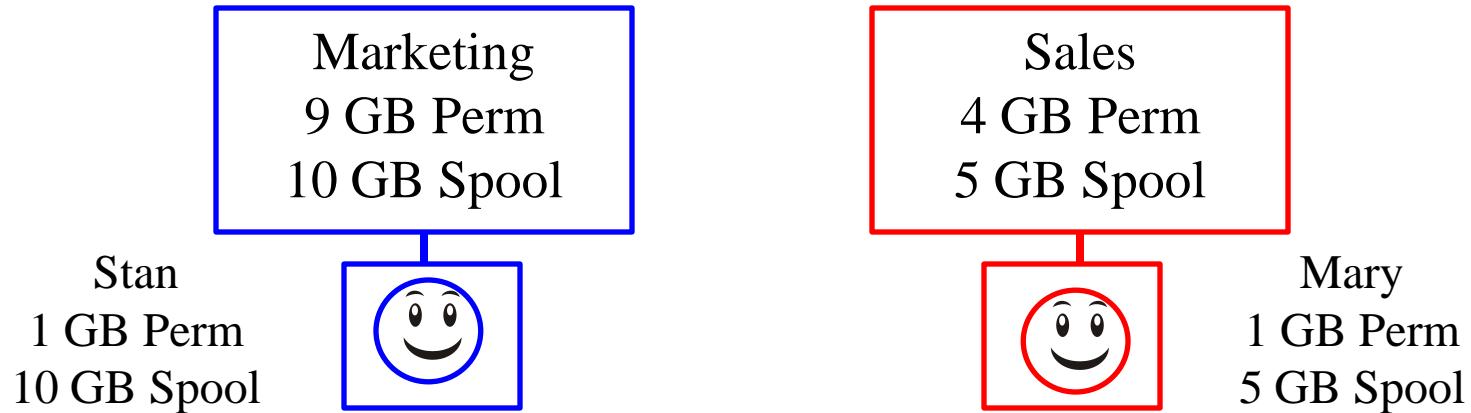
Space Transfer Quiz

1

If a USER is dropped their PERM Space goes up to the immediate parent.

2

If a USER is transferred (GIVE Statement) they take their space with them.



Stan has just been transferred to Sales.

After the transfer how much Perm / Spool is in:

Marketing

Sales

Stan

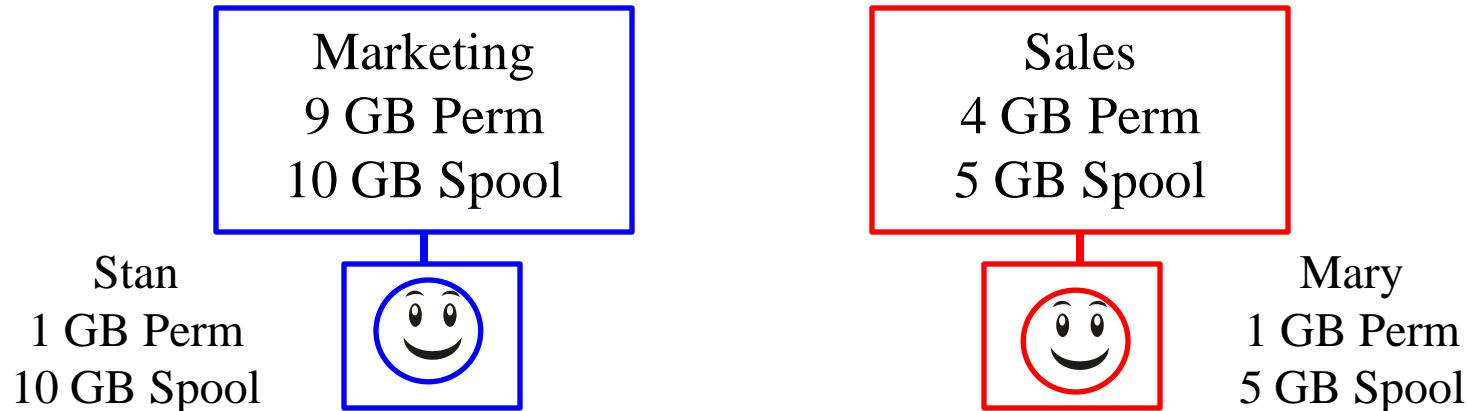
Answer to Space Transfer Quiz

1

If a USER is dropped their PERM Space goes up to their immediate parent.

2

If a USER is transferred (GIVE Statement) they take their space with them.



Stan has just been transferred to Sales.

After the transfer how much Perm / Spool is in:

Marketing	9 GB Perm	10 GB Spool
Sales	4 GB Perm	5 GB Spool
Stan	1 GB Perm	10 GB Spool

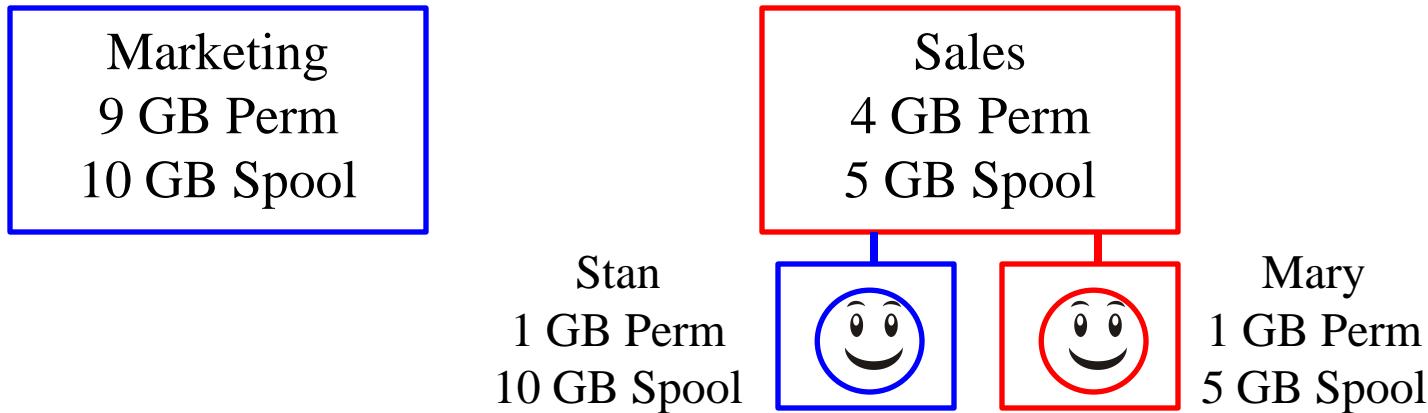
Drop Space Quiz

1

If a USER is dropped their PERM Space goes up to their immediate parent.

2

If a USER is transferred (GIVE Statement) they take their space with them.



What happens **NOW** if Stan is Dropped?

After the drop how much Perm / Spool is in:

Marketing _____

Sales _____

Stan _____

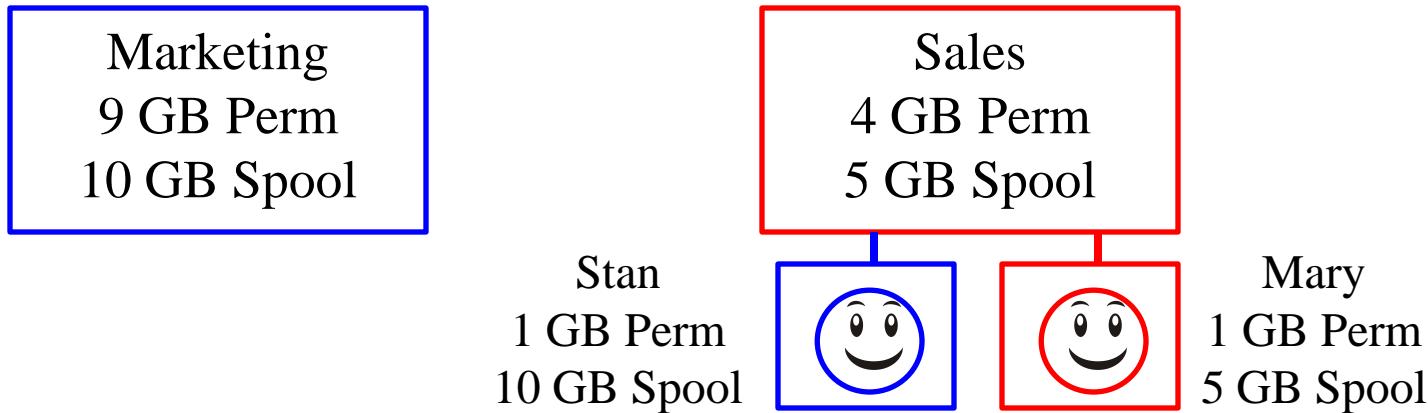
Answers to Drop Space Quiz

1

If a USER is dropped their PERM Space goes up to their immediate parent.

2

If a USER is transferred (GIVE Statement) they take their space with them.



What happens **NOW** if Stan is Dropped?

After the drop how much Perm / Spool is in:

Marketing	9 GB Perm	10 GB Spool
Sales	5 GB Perm	5 GB Spool
Stan	dropped (0)	dropped (0)

Watch the Video on Space



Tera-Toms picture was retired and placed inside the wrestling room at the University of Arizona.

The University sent Tom the picture years later and his three kids are pictured with it.

Tera-Tom Trivia

Click on the link below or place it in your browser and watch the video on space.

<http://www.coffingdw.com/TbasicsV12/Space.wmv>

Chapter 5

Partition Primary Index (PPI) Tables

“I saw an angel in the marble and carved until I set him free.”

- Michelangelo

Table of Contents Chapter 5 - Partition Primary Index (PPI) Tables

- [The Concept behind Partitioning a Table](#)
- [Review of A Unique Primary Index \(UPI\)](#)
- [Primary Index in the WHERE Clause - Single-AMP Retrieve](#)
- [Review of A Non-Unique Primary Index \(NUPI\)](#)
- [Primary Index in the WHERE Clause - Single-AMP Retrieve](#)
- [Review of a Multi-Column Primary Index](#)
- [Primary Index in the WHERE Clause - Single-AMP Retrieve](#)
- [Creating a PPI Table with Simple Partitioning](#)
- [A Visual Display of Simple Partitioning](#)
- [An SQL Example that explains Simple Partitioning](#)
- [Creating a PPI Table with RANGE_N Partitioning per Day](#)
- [A Visual of Range_N Partitioning Per Day](#)
- [An SQL Example that explains Range_N Partitioning per Day](#)
- [Creating a PPI Table with RANGE_N Partitioning per Week](#)
- [A Visual of Range_N Partitioning Per Week](#)
- [An SQL Example that explains Range_N Partitioning per Day](#)
- [Creating a PPI Table with RANGE_N Partitioning per Month](#)
- [A Visual of One Year of Data with Range_N Per Month](#)
- [An SQL Example explaining Range_N Partitioning per Month](#)
- [Creating a PPI Table with CASE_N](#)
- [A Visual of Case_N Partitioning](#)
- [An SQL Example that explains Range_N Partitioning per Day](#)
- [How many partitions do you see?](#)
- [Number of PPI Partitions Allowed](#)
- [How many partitions do you see?](#)
- [NO CASE and UNKNOWN Partitions Together](#)
- [A Visual of Case_N Partitioning](#)
- [Combining Older Data and Newer Data in PPI](#)
- [A Visual for Combining Older Data and Newer Data in PPI](#)

Continued on next page

Chapter 5 - Partition Primary Index (PPI) Tables Continued

- [The SQL on Combining Older Data and Newer Data in PPI](#)
- [Multi-Level Partitioning Combining Range_N and Case_N](#)
- [A Visual of Multi-Level Partitioning](#)
- [The SQL on a Multi-Level Partitioned Primary Index](#)
- [NON-Unique Primary Indexes \(NUPI\) in PPI](#)
- [PPI Table with a Unique Primary Index \(UPI\)](#)
- [Tricks for Non-Unique Primary Indexes \(NUPI\)](#)
- [Character Based PPI for RANGE_N](#)
- [A Visual for Character Based PPI for RANGE_N](#)
- [The SQL on Character Based PPI for RANGE_N](#)
- [Character Based PPI for CASE_N](#)
- [Dates and Character Based Multi-Level PPI](#)
- [TIMESTAMP Partitioning](#)
- [Using CURRENT_DATE to define a PPI](#)
- [ALTER to CURRENT_DATE the next year](#)
- [ALTER to CURRENT_DATE with Save](#)
- [Altering a PPI Table to Add or Drop Partitions](#)
- [Deleting a Partition](#)
- [Deleting a Partition and Saving its contents](#)
- [Using the PARTITION Keyword in your SQL](#)
- [SQL for RANGE_N](#)
- [SQL for CASE_N](#)
- [Watch the Video on Partitioning](#)

The Concept behind Partitioning a Table

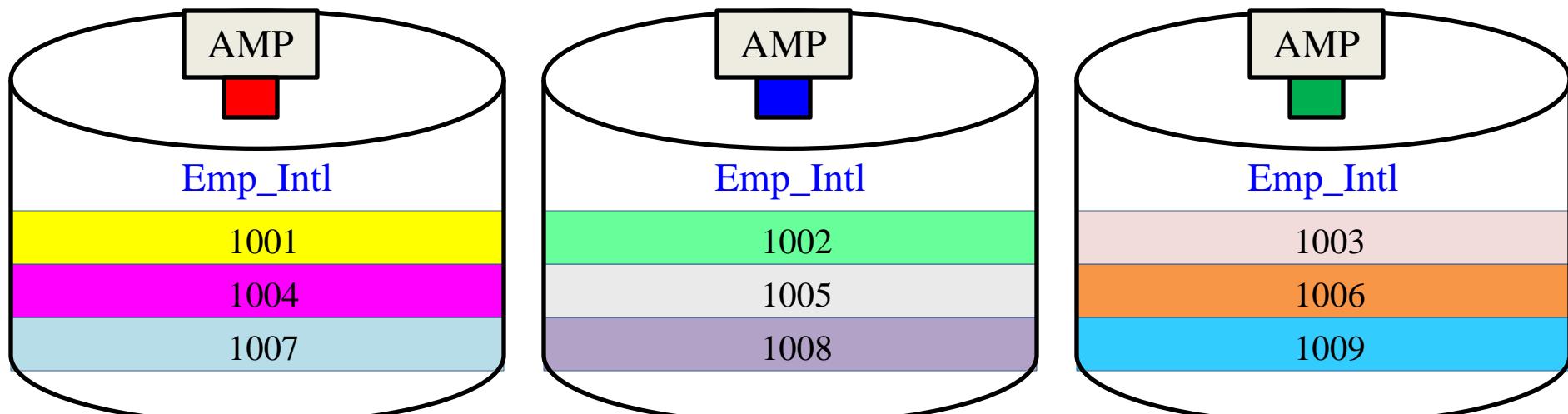
1. Each Table in Teradata has a **Primary Index**, unless it is a NoPI table.
 2. The Primary Index is the mechanism that allows Teradata to physically **distribute** the rows of a table across the AMPS.
 3. AMPS Sort their rows by the **Row-ID** so the system can perform a lightning fast Binary Search since the rows are in Row-ID Order.
 4. Partitioning merely tells the AMP to sort its tables' rows by the Partition first, but then sort the rows by Row-ID within the partition.
 5. Partitioning queries will involve all AMPS, but partitioned tables are designed to prevent FULL Table Scans.
-

The basic concepts of Partitioning are above so implant these in your mind.

Review of A Unique Primary Index (UPI)

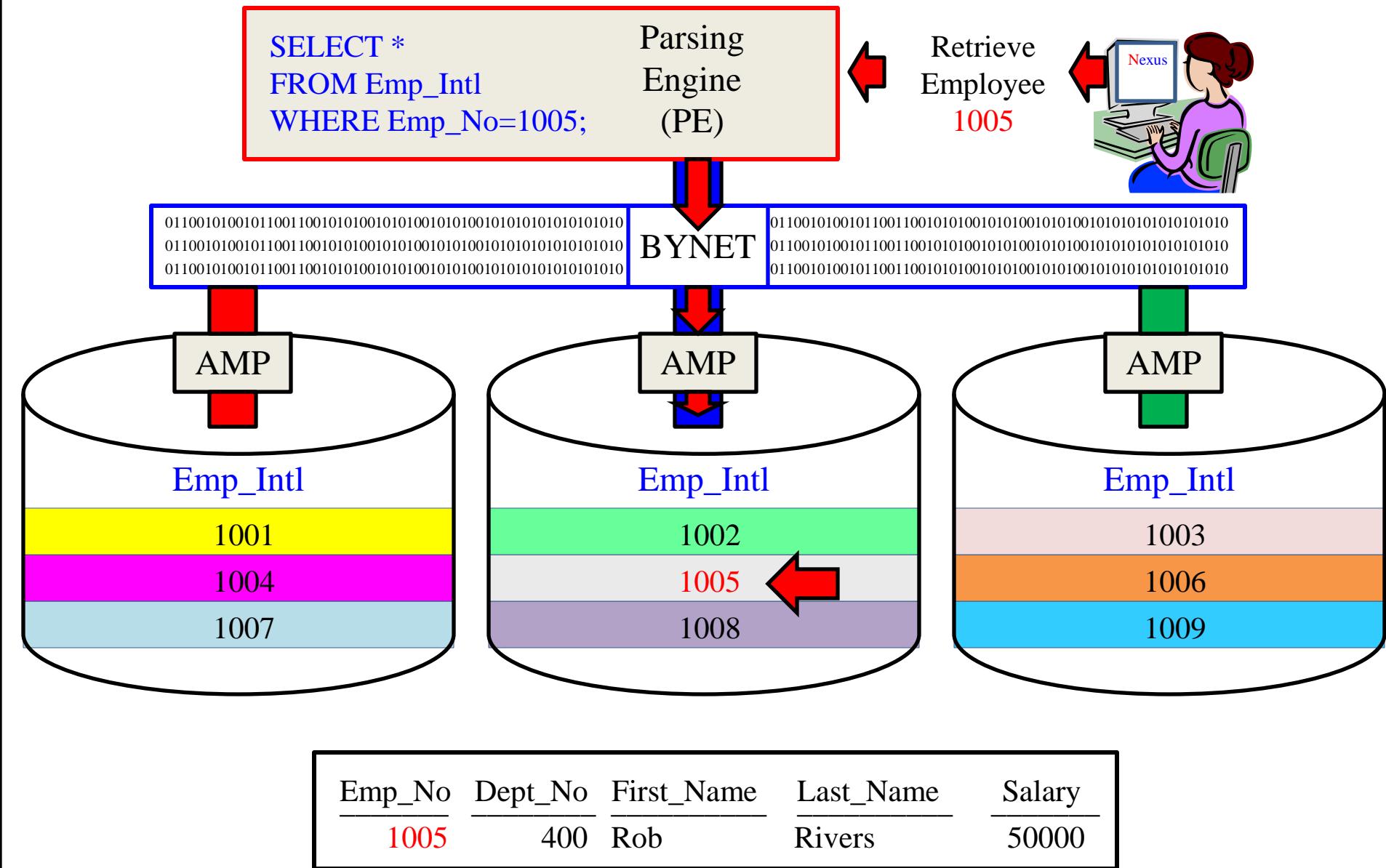
```
CREATE TABLE Emp_Intl
(Emp_No          INTEGER,
Dept_No         SMALLINT,
First_Name      VARCHAR(12),
Last_Name       CHAR(20),
Salary          DECIMAL(10,2))
UNIQUE Primary Index( Emp_No )
```

Unique Primary Index		Emp_Intl			
	Emp_No	Dept_No	First_Name	Last_Name	Salary
	1001	100	Rafael	Minal	90000.00
	1002	200	Maria	Gomez	80000.00
	1003	300	Charl	Kertzel	70000.00
	1004	400	Kyle	Stover	60000.00
	1005	400	Rob	Rivers	50000.00
	1006	300	Inna	Kinski	50000.00
	1007	200	Sushma	Davis	50000.00
	1008	100	Mo	Khan	60000.00
	1009	300	Mo	Swartz	70000.00



A Unique Primary Index(UPI) spreads the rows of a table evenly across the AMPS.

Primary Index in the WHERE Clause - Single-AMP Retrieve

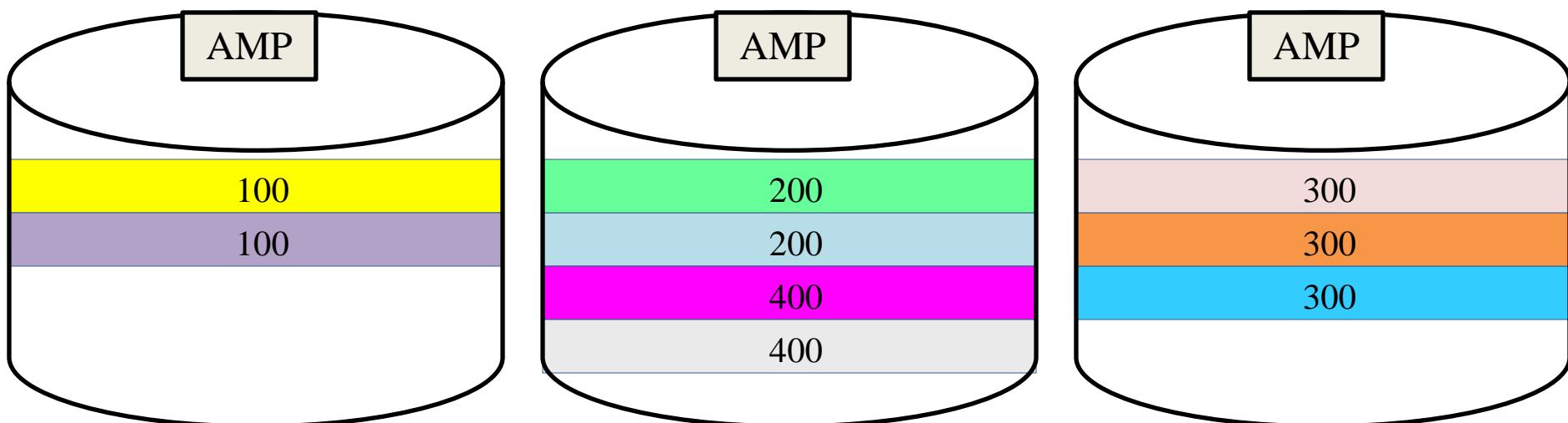


Use the Primary Index column in your SQL WHERE clause and only 1 AMP retrieves.

Review of A Non-Unique Primary Index (NUPI)

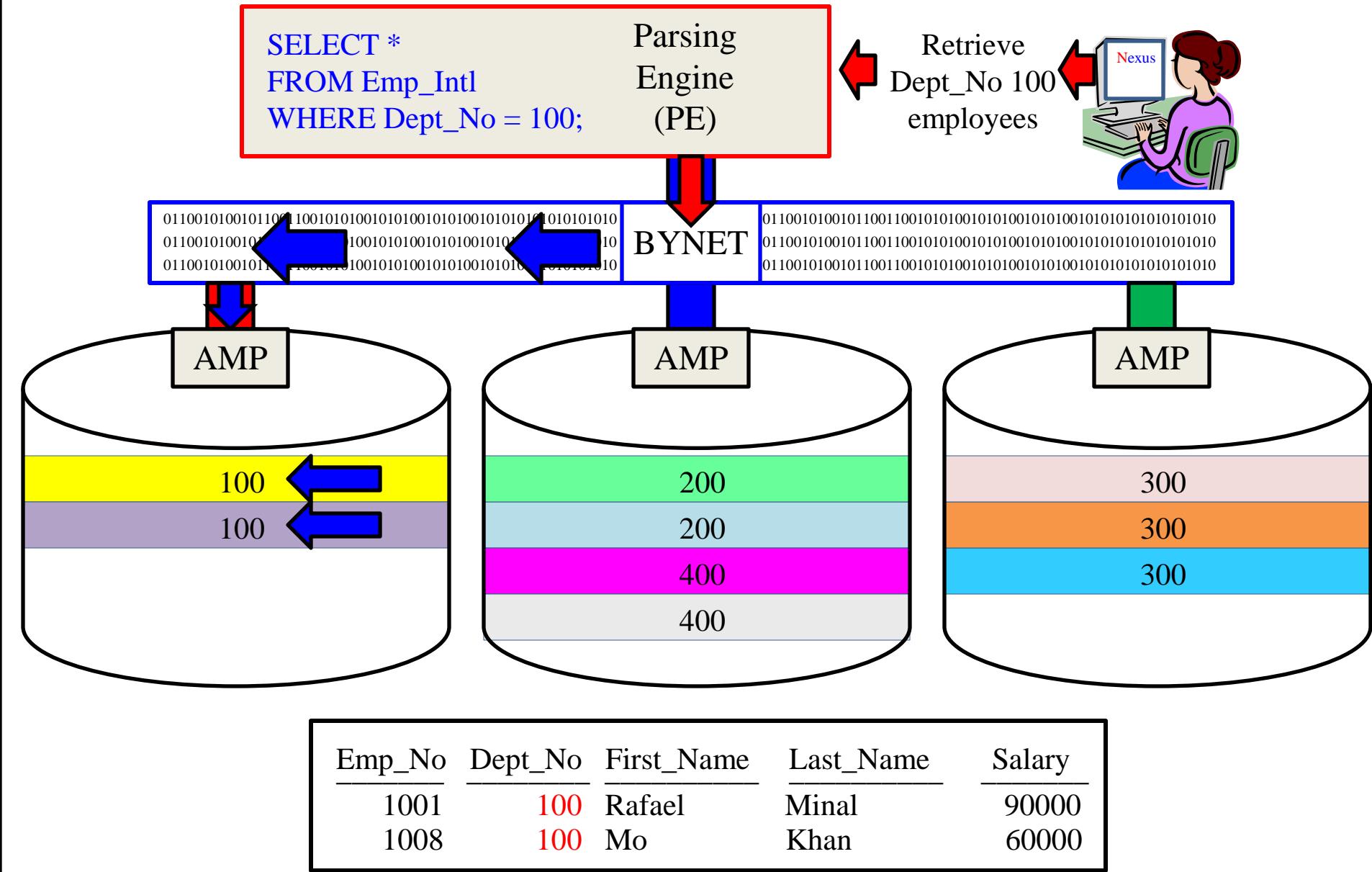
```
CREATE TABLE Emp_Intl
(Emp_No          INTEGER,
Dept_No          SMALLINT,
First_Name       VARCHAR(12),
Last_Name        CHAR(20),
Salary           DECIMAL(10,2))
Primary Index( Dept_No )
```

NUPI		Emp_Intl			
Emp_No	Dept_No	First_Name	Last_Name	Salary	
1001	100	Rafael	Minal	90000.00	
1002	200	Maria	Gomez	80000.00	
1003	300	Charl	Kertzel	70000.00	
1004	400	Kyle	Stover	60000.00	
1005	400	Rob	Rivers	50000.00	
1006	300	Inna	Kinski	50000.00	
1007	200	Sushma	Davis	50000.00	
1008	100	Mo	Khan	60000.00	
1009	300	Mo	Swartz	70000.00	



A Non-Unique Primary Index (NUPI) will have duplicates grouped together on the same AMP so data is skewed (uneven). The above skew is reasonable and OK.

Primary Index in the WHERE Clause - Single-AMP Retrieve

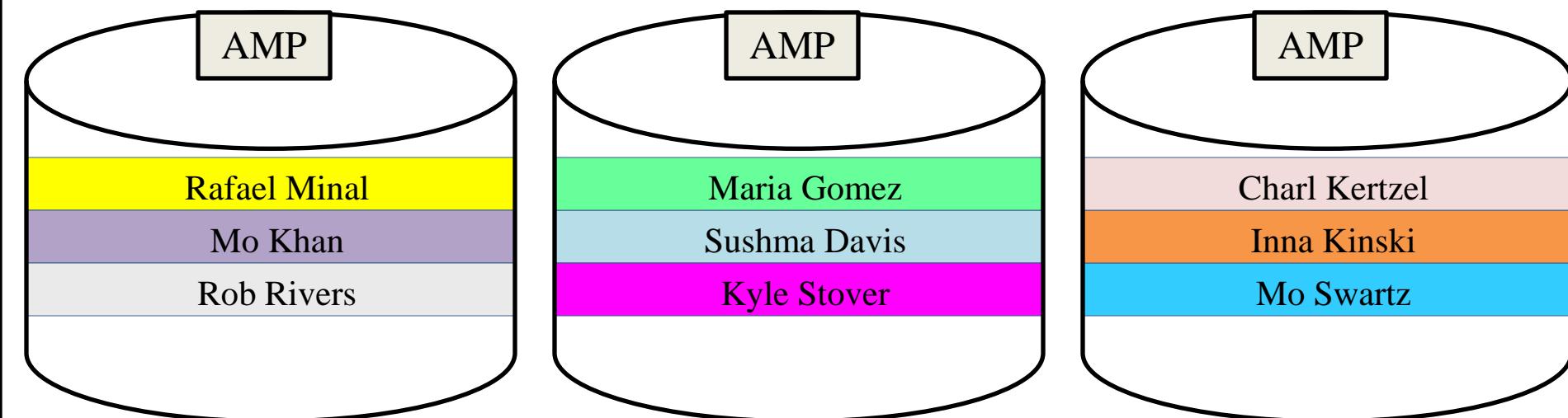


Use the Primary Index column in your SQL WHERE clause and only 1 AMP retrieves.

Review of a Multi-Column Primary Index

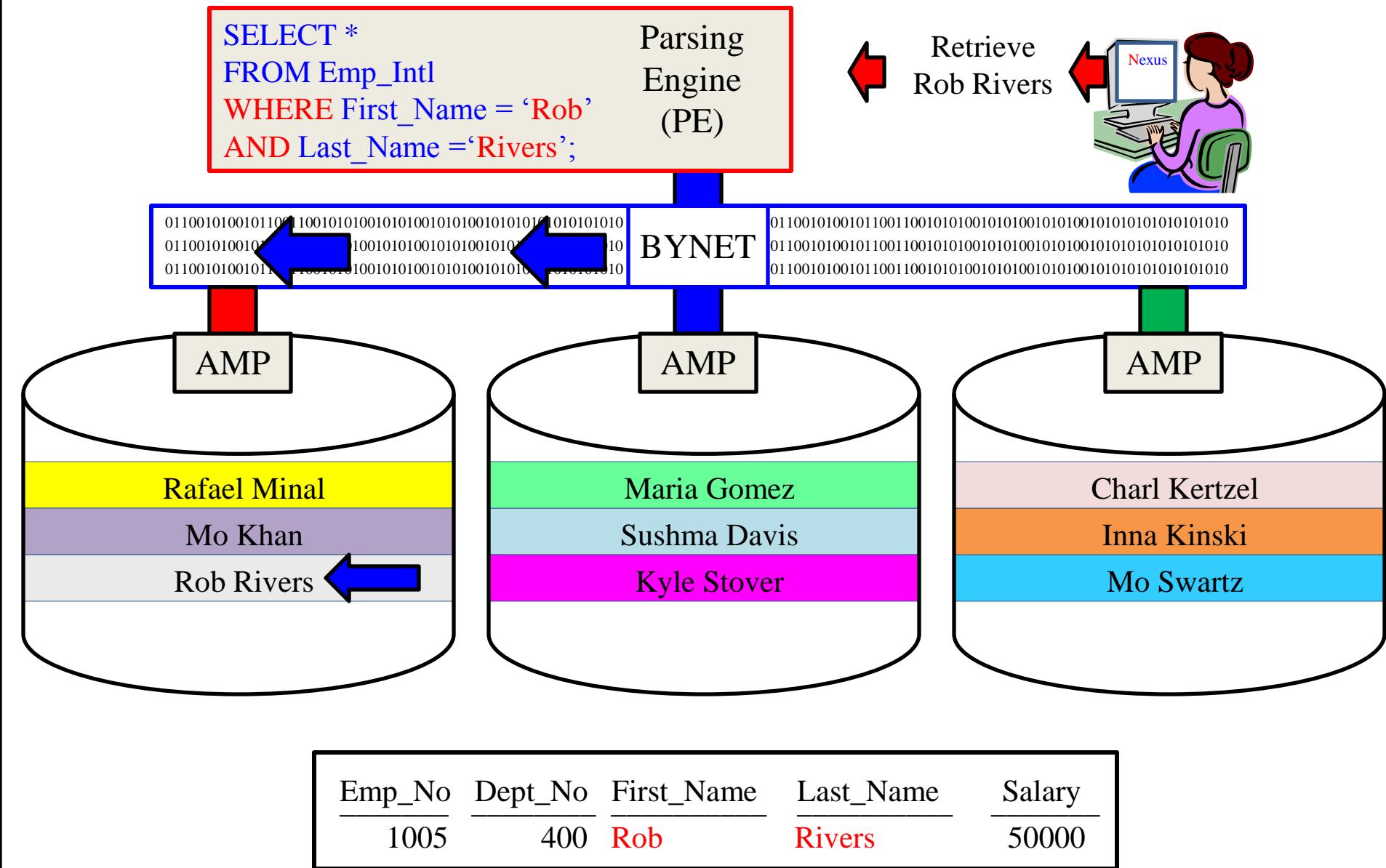
```
CREATE TABLE Emp_Intl
(Emp_No          INTEGER,
Dept_No         SMALLINT,
First_Name      VARCHAR(12),
Last_Name       CHAR(20),
Salary          DECIMAL(10,2))
Primary Index( First_Name, Last_Name)
```

Multi-Column Primary Index					
Emp_No	Dept_No	First_Name	Last_Name	Salary	
1001	100	Rafael	Minal	90000.00	
1002	200	Maria	Gomez	80000.00	
1003	300	Charl	Kertzel	70000.00	
1004	400	Kyle	Stover	60000.00	
1005	400	Rob	Rivers	50000.00	
1006	300	Inna	Kinski	50000.00	
1007	200	Sushma	Davis	50000.00	
1008	100	Mo	Khan	60000.00	
1009	300	Mo	Swartz	70000.00	



A table can have only one Primary Index, but you can combine up to 64 columns together max to form one Multi-Column Primary Index.

Primary Index in the WHERE Clause - Single-AMP Retrieve



Use the Primary Index column in your SQL WHERE clause and only 1 AMP retrieves.

Creating a PPI Table with Simple Partitioning

```
CREATE Table Employee_Table_PPI  
(  
    Employee_No      INTEGER  
    ,Dept_No         INTEGER  
    ,Last_Name       CHAR(20)  
    ,First_Name      VARCHAR(20)  
    ,Salary          DECIMAL(10,2)  
) Primary Index (Employee_No)  
PARTITION BY (Dept_No) ;
```

There are three different types of Partitioning:

- 1) Simple
- 2) RANGE_N
- 3) CASE_N

Sort the rows on each AMP by Dept_No!



A PPI Table is a table with a Partition on it. A PPI table has the AMPs sort (order) the rows on the table by the Partition. This allows for people to avoid Full Table Scans. This is an example of Simple PPI. Each AMP will sort the rows they own by Dept_No.

A Visual Display of Simple Partitioning

Partition
by Dept_No



Sort the rows on
each AMP by
Dept_No!



Employee_Table_PPI				
Emp_No	Dept_No	First_Name	Last_Name	Salary
1001	100	Rafael	Minal	90000.00
1002	200	Maria	Gomez	80000.00
1003	300	Charl	Kertzel	70000.00
1004	400	Kyle	Stover	60000.00
1005	400	Rob	Rivers	50000.00
1006	300	Inna	Kinski	50000.00
1007	200	Sushma	Davis	50000.00
1008	100	Mo	Khan	60000.00
1009	300	Mo	Swartz	70000.00

AMP

AMP

Partition 1	1001	100	Rafael	Minal	90000
Partition 2	1007	200	Sushma	Davis	50000
Partition 3	1006	300	Inna	Kinski	50000
3	1009	300	Mo	Swartz	70000
Partition 4	1004	400	Kyle	Stover	60000

Partition 1	1008	100	Mo	Khan	60000
Partition 2	1002	200	Maria	Gomez	80000
Partition 3	1003	300	Charl	Kertzel	80000
Partition 4	1005	400	Rob	Rivers	50000

An SQL Example that explains Simple Partitioning

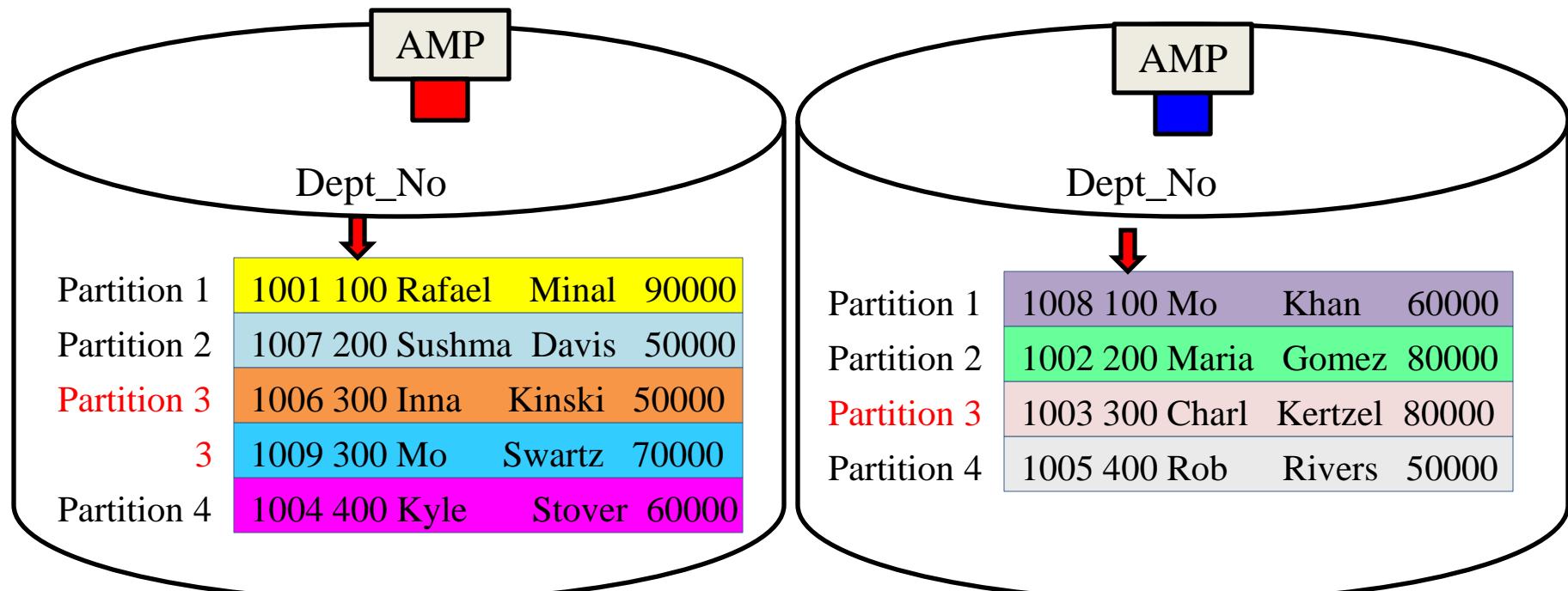
SQL Query

```
SELECT *  
FROM Employee_Table_PPI  
WHERE Dept_No = 300 ;
```

PE's Explain Plan

We do an All-AMPs retrieve by way of a Single Partition.

The table below was Partitioned by **Dept_No** using Simple Partitioning. Each AMP merely reads from **partition 3** which holds all **Dept_No 300** employees.



Creating a PPI Table with RANGE_N Partitioning per Day

```
CREATE TABLE Order_Table_PPI
(Order_Number      INTEGER  Not Null
,Customer_Number  INTEGER
,Order_Date        DATE
,Order_Total       Decimal (10,2)
) PRIMARY INDEX(Order_Number)
PARTITION BY RANGE_N (Order_Date
                      BETWEEN date '2012-01-01'
                      AND   date '2012-12-31'
                      EACH INTERVAL '1' DAY);
```

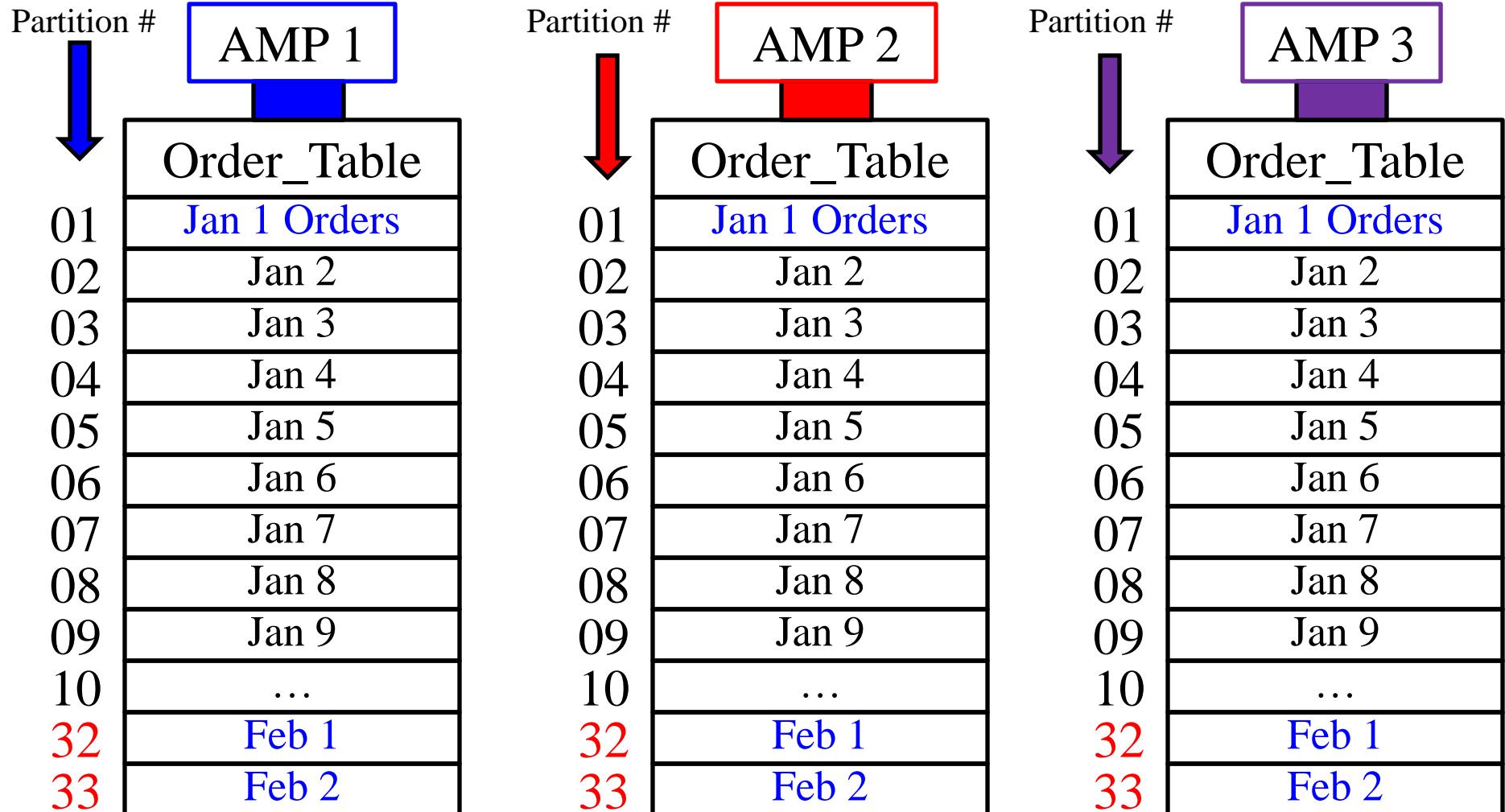
There are three different types of Partitioning:

- 1) Simple
- 2) RANGE_N
- 3) CASE_N

Sort the rows on each AMP by Order_Date and create a different partition each Day!

The above syntax represents a RANGE_N partition.

A Visual of Range_N Partitioning Per Day



There would be 365 partitions in total for an entire year of Partitioning per Day. Each AMP holds the orders assigned to them in daily partitions on the day of the order date.

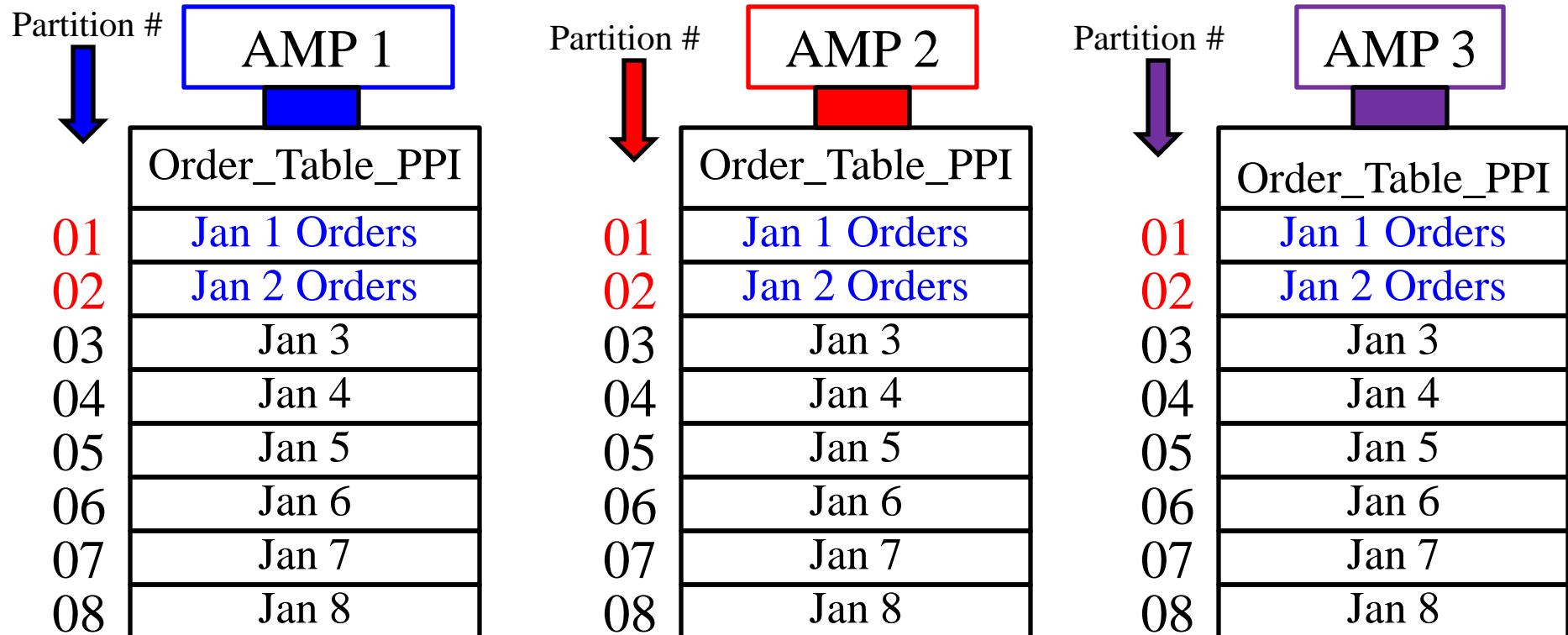
An SQL Example that explains Range_N Partitioning per Day

SQL Query

```
SELECT *  
FROM Order_Table_PPI  
WHERE Order_Date BETWEEN  
'2012-01-01' and '2012-01-02';
```

PE's Explain Plan

We do an All-AMPs
retrieve by way of
2 Partitions.



Each AMP holds the orders assigned to them in daily partitions on the day of the order .

Creating a PPI Table with RANGE_N Partitioning per Week

```
CREATE TABLE Order_Table_PPI
(Order_Number      INTEGER  Not Null
,Customer_Number  INTEGER
,Order_Date       DATE
,Order_Total      Decimal (10,2)
) PRIMARY INDEX(Order_Number)
PARTITION BY RANGE_N (Order_Date
                      BETWEEN date '2012-01-01'
                      AND   date '2012-12-31'
                      EACH INTERVAL '7' DAY);
```

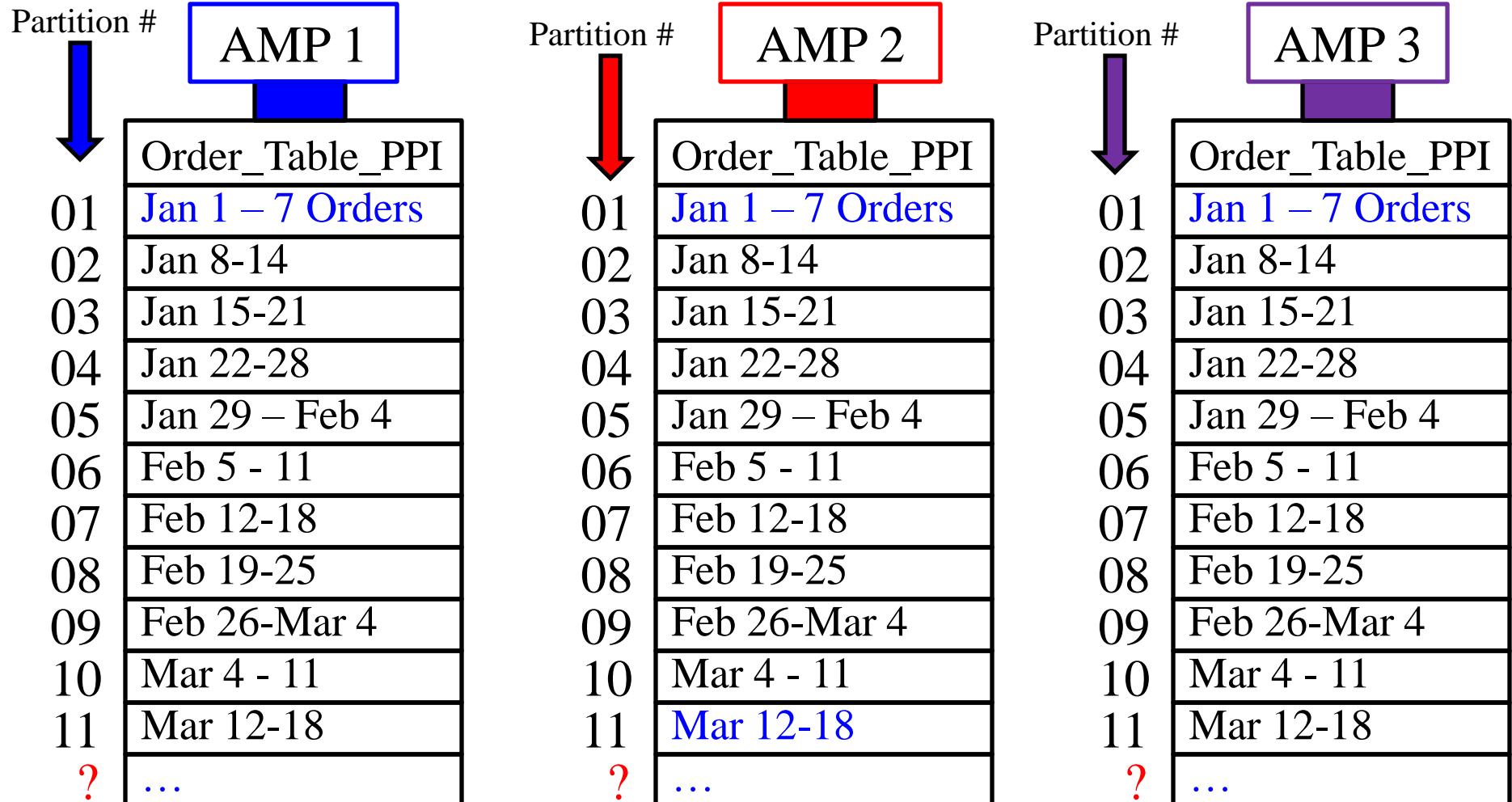
There are three different types of Partitioning:

- 1) Simple
- 2) RANGE_N
- 3) CASE_N

Sort the rows on each AMP by Order_Date and create a different partition every 7 days

You can order everything by day. You can also order everything by week, my month, and by year. You have many options to choose from.

A Visual of Range_N Partitioning Per Week



There would be 52 partitions in total for an entire year of Partitioning per Week. Each AMP holds the orders assigned to them in weekly partitions on the week of the order.

SQL Example that explains Range_N Partitioning per Week

SQL Query

```
SELECT *
FROM Order_Table_PPI
WHERE Order_Date BETWEEN
'2012-01-01' and '2012-01-07';
```

PE's Explain Plan

We do an All-AMPs retrieve by way of a single Partition.

Partition #

AMP 1

Partition #

AMP 2

Partition #

AMP 3

Order_Table_PPI	
01	Jan 1 – 7 Orders
02	Jan 8-14
03	Jan 15-21
04	Jan 22-28
05	Jan 29 – Feb 4
06	Feb 5 - 11
07	Feb 12-18

Order_Table_PPI	
01	Jan 1 – 7 Orders
02	Jan 8-14
03	Jan 15-21
04	Jan 22-28
05	Jan 29 – Feb 4
06	Feb 5 - 11
07	Feb 12-18

Order_Table_PPI	
01	Jan 1 – 7 Orders
02	Jan 8-14
03	Jan 15-21
04	Jan 22-28
05	Jan 29 – Feb 4
06	Feb 5 - 11
07	Feb 12-18

Each AMP holds the orders assigned to them in weekly partitions based on the week of the order.

Creating a PPI Table with RANGE_N Partitioning per Month

```
CREATE TABLE Order_Table_PPI
(Order_Number      INTEGER  Not Null
,Customer_Number  INTEGER
,Order_Date        DATE
,Order_Total       Decimal (10,2)
) PRIMARY INDEX(Order_Number)
PARTITION BY RANGE_N (Order_Date
                      BETWEEN date '2012-01-01'
                      AND   date '2012-12-31'
                      EACH INTERVAL '1' Month);
```

There are three different types of Partitioning:

- 1) Simple
- 2) RANGE_N
- 3) CASE_N

Sort the rows on each AMP by Order_Date and create a different partition every Month

You can order everything by day. You can also order everything by week, by month, and by year. You have many options to choose from.

A Visual of One Year of Data with Range_N Per Month



Order_Table_PPI	
01	JAN
02	FEB
03	MAR
04	APR
05	MAY
06	JUN
07	JUL
08	AUG
09	SEP
10	OCT
11	NOV
12	DEC

Order_Table_PPI	
01	JAN
02	FEB
03	MAR
04	APR
05	MAY
06	JUN
07	JUL
08	AUG
09	SEP
10	OCT
11	NOV
12	DEC

Order_Table_PPI	
01	JAN
02	FEB
03	MAR
04	APR
05	MAY
06	JUN
07	JUL
08	AUG
09	SEP
10	OCT
11	NOV
12	DEC

Order_Table_PPI	
01	JAN
02	FEB
03	MAR
04	APR
05	MAY
06	JUN
07	JUL
08	AUG
09	SEP
10	OCT
11	NOV
12	DEC

Each AMP sorts their rows by Month (of Order_Date).

An SQL Example explaining Range_N Partitioning per Month

SQL Query

```
SELECT *
FROM Order_Table
WHERE Order_Date BETWEEN
‘2012-01-01’ and ‘2012-03-31’;
```

PE’s Explain Plan

We do an All-AMPs retrieve by way of
3 Partition3.

AMP 1

AMP 2

AMP 3

AMP 4

Order_Table_PPI	
01	JAN
02	FEB
03	MAR
04	APR
05	MAY
06	JUN
07	JUL

Order_Table_PPI	
	JAN
	FEB
	MAR
	APR
	MAY
	JUN
	JUL

Order_Table_PPI	
	JAN
	FEB
	MAR
	APR
	MAY
	JUN
	JUL

Order_Table_PPI	
	JAN
	FEB
	MAR
	APR
	MAY
	JUN
	JUL

The above query wants to see orders in the first quarter so each AMP reads 3 partitions.

Creating a PPI Table with CASE_N

```
CREATE TABLE Order_Table_CPPI  
(Order_Number      INTEGER  Not Null  
)
```

PRIMARY INDEX(Order_Number)

PARTITION BY CASE_N

```
(Order_Total < 1000,  
 Order_Total < 5000,  
 Order_Total < 10000,  
 Order_Total < 20000,  
 NO Case, UNKNOWN);
```

There are three different types of Partitioning:

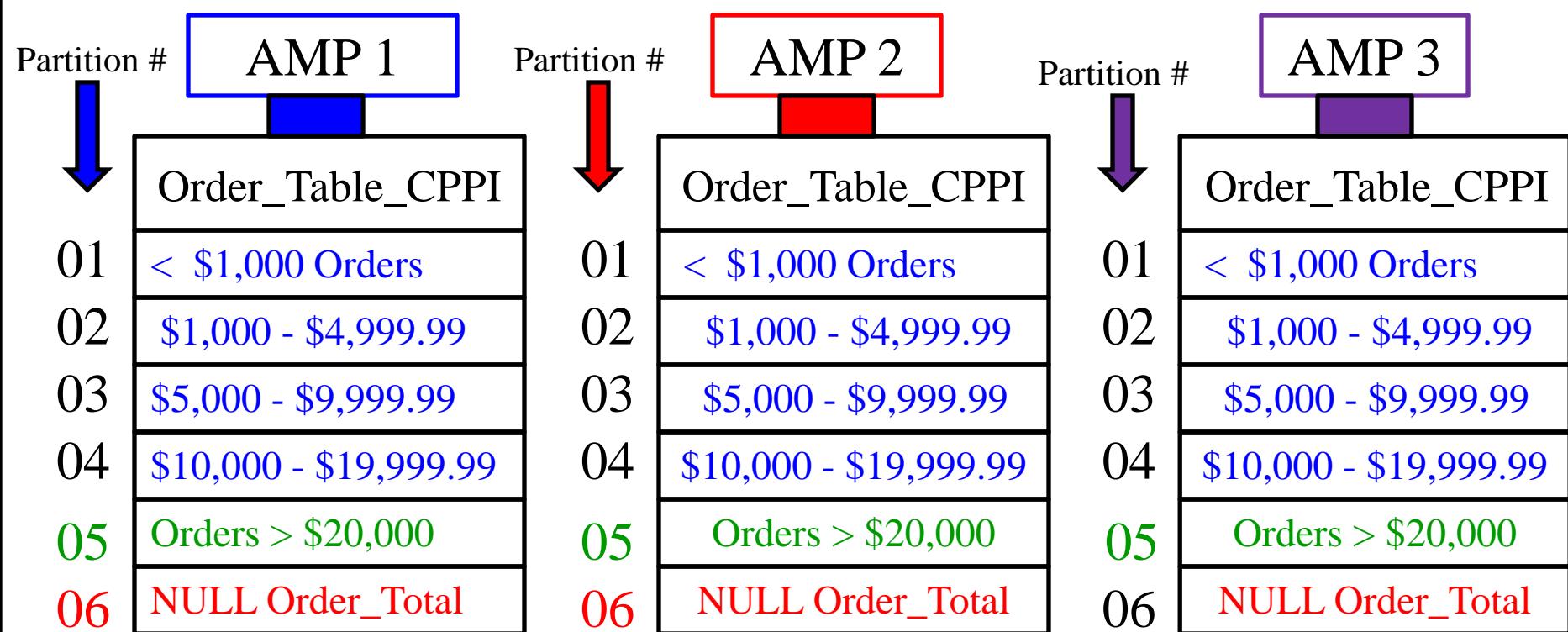
- 1) Simple
- 2) RANGE_N
- 3) CASE_N

Sort the rows on each AMP by a CASE Statement. There are six partitions in total.

The above syntax represents the CASE_N partitioning. If an Order_Total is < 1000 it will go into Partition 1. If an Order_Total is between 1000 and 4999.99 it will go in Partition 2. The NO Case partition is for anything falling through and the UNKNOWN partition is for NULL values in the Total.

A Visual of Case_N Partitioning

```
PARTITION BY CASE_N  
(Order_Total < 1000,  
 Order_Total < 5000,  
 Order_Total < 10000,  
 Order_Total < 20000,  
 NO Case, UNKNOWN);
```



Three are six partitions for this table. Orders go into partitions based on Order_Total.

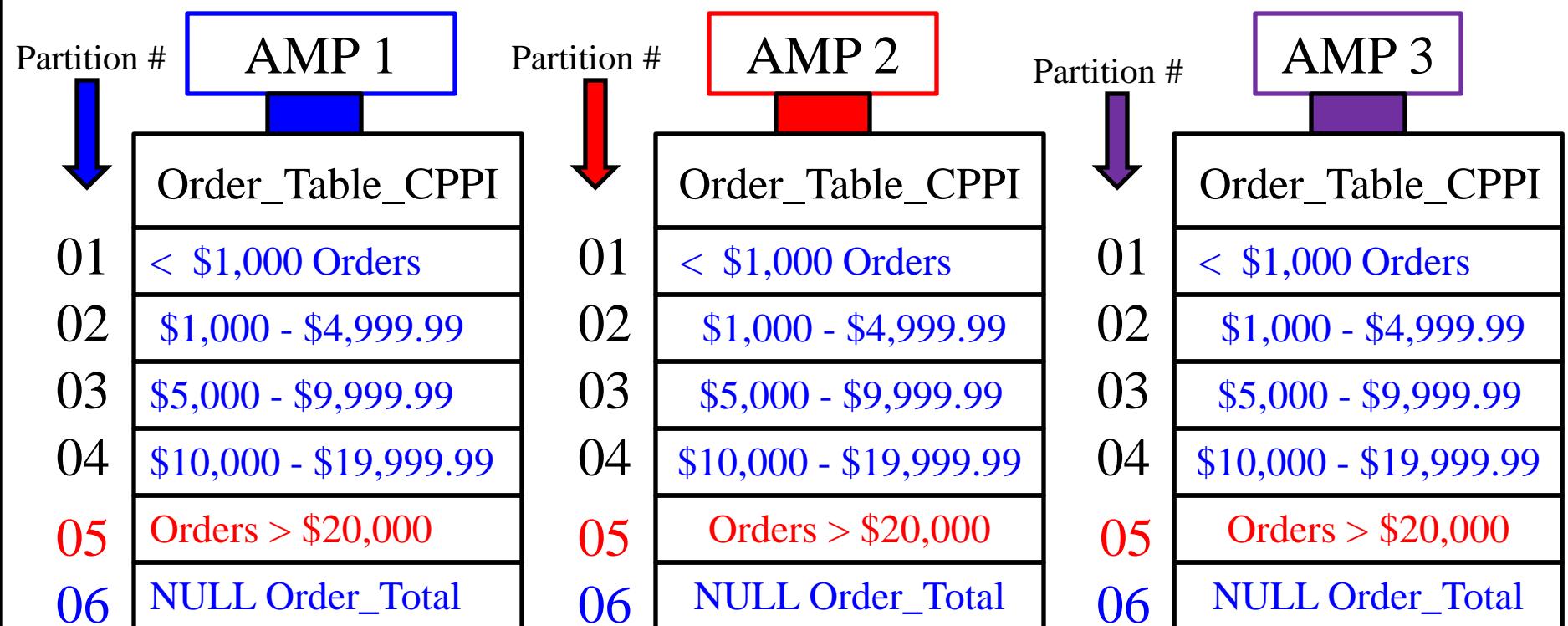
An SQL Example that explains CASE_N Partitioning

SQL Query

```
SELECT *
FROM Order_Table_CPPI
WHERE Order_Total > 20000.00
```

PE's Explain Plan

We do an All-AMPs retrieve by way of a Single Partition.



All AMPs retrieve but each only reads partition 5 which is a mere sliver of the disk.

How many partitions do you see?

```
CREATE TABLE Order_Table_PPI
(Order_Number      INTEGER  Not Null
,Customer_Number  INTEGER
,Order_Date        DATE
,Order_Total       Decimal (10,2)
) PRIMARY INDEX(Order_Number)
PARTITION BY RANGE_N (Order_Date
                      BETWEEN date '2012-01-01'
                        AND date '2012-12-31'
                      EACH INTERVAL '1' DAY);
```

How many partitions will Order_Table_PPI have at the year's end? It will have 365, unless it is leap year and then it will have 366.

Number of PPI Partitions Allowed

```
CREATE TABLE Order_Table_PPI
(Order_Number      INTEGER  Not Null
,Customer_Number  INTEGER
,Order_Date       DATE
,Order_Total      Decimal (10,2)
) PRIMARY INDEX(Order_Number)
PARTITION BY RANGE_N (Order_Date
                      BETWEEN date '2012-01-01'
                      AND   date '2012-12-31'
                      EACH INTERVAL '1' DAY);
```

Each AMP can have 65,535 partitions per table, but after Teradata V14 each AMP can have 9.223 Quintillion partitions per table.

Teradata Tables can have 65,535 partitions max, but after V14 it is 9.223 Quintillion.

How many partitions do you see?

```
CREATE TABLE Order_Table_CPPI
(Order_Number      INTEGER  Not Null
,Customer_Number  INTEGER
,Order_Date       DATE
,Order_Total      Decimal (10,2))
PRIMARY INDEX(Order_Number)
PARTITION BY CASE_N
(Order_Total < 1000,
Order_Total < 5000,
Order_Total < 10000,
Order_Total < 20000,
NO Case, UNKNOWN);
```

How many partitions do you see in the above example?

There are six partitions in the above example. Partition 1 is for Order_Totals < 1000 and Partition 2 is for Order_Totals < 5000, etc. The **NO Case** partition is for anything falling through the CASE statement, like an ELSE so it is for any Order_Total 20,000 or greater. The **UNKNOWN** is for Order_Totals that are Null.

NO CASE and UNKNOWN Partitions Together

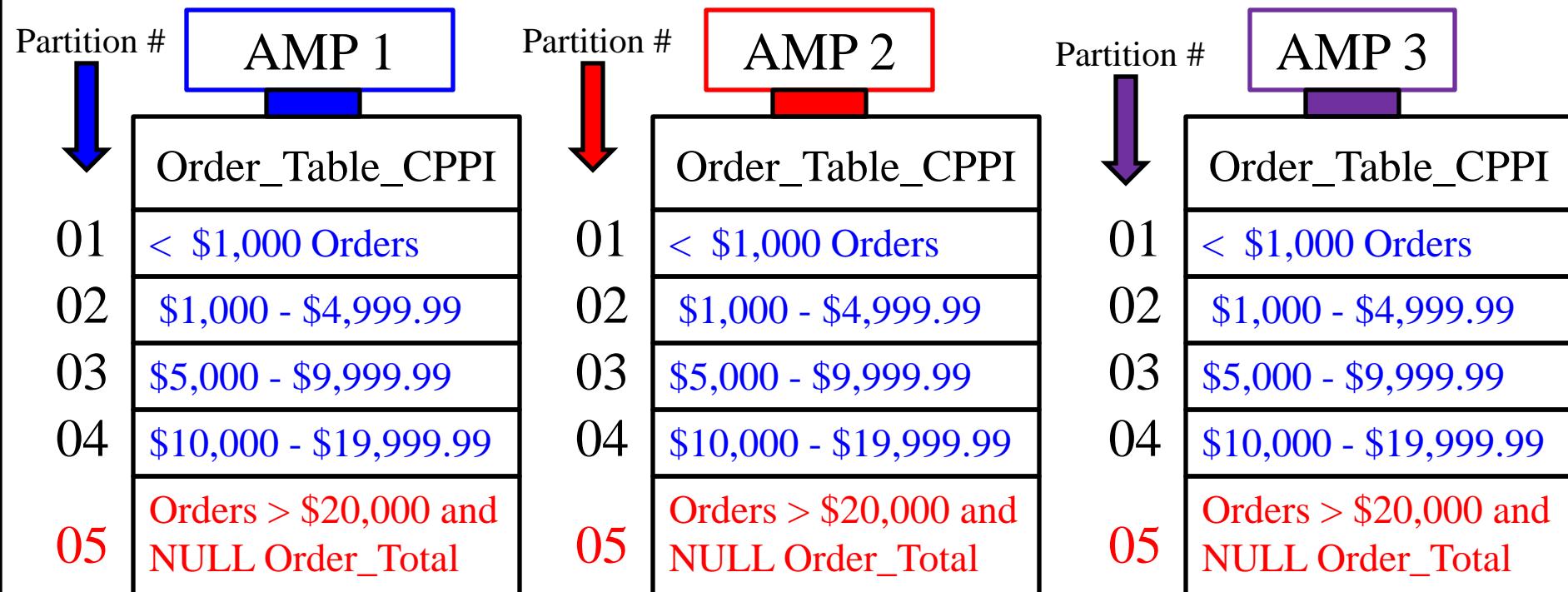
```
CREATE TABLE Order_Table_CPPI
(Order_Number      INTEGER  Not Null
,Customer_Number  INTEGER
,Order_Date       DATE
,Order_Total      Decimal (10,2)
)
PRIMARY INDEX(Order_Number)
PARTITION BY CASE_N
(Order_Total < 1000,
Order_Total < 5000,
Order_Total < 10000,
Order_Total < 20000,
NO Case OR UNKNOWN);
```

How many partitions do you see in the above example?

There are five partitions in the above example. Partition 1 is for Order-Totals < 1000 and Partition 2 is for Order_Totals < 5000, etc. The NO Case partition and the UNKNOWN partition are combined. So now any Order_Total that is 20,000 or greater or any Order_Total that is NULL will be in the 5th partition together.

A Visual of Case_N Partitioning

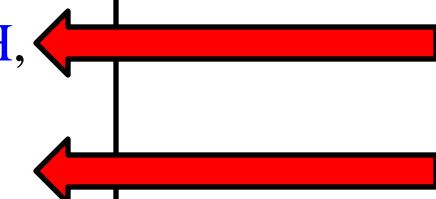
PARTITION BY CASE_N
(Order_Total < 1000,
Order_Total < 5000,
Order_Total < 10000,
Order_Total < 20000,
NO CASE OR UNKNOWN);



Three are five partitions for this table because we combined the NO CASE and UNKNOWN partitions together. Orders are placed in partitions based on Order_Total.

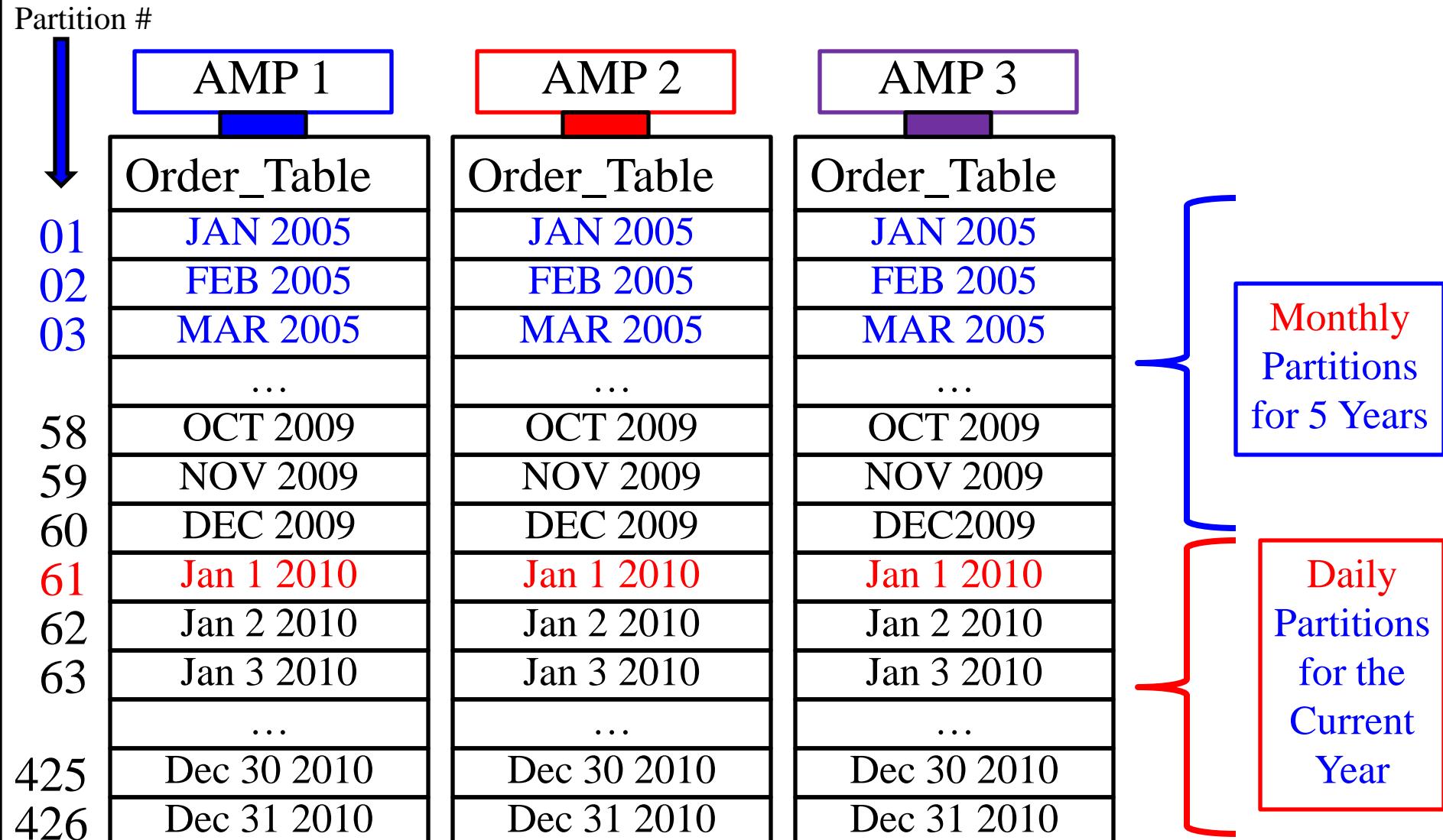
Combining Older Data and Newer Data in PPI

```
CREATE TABLE Order_Table_MultiYear  
(Order_Number      INTEGER    Not Null  
,Customer_Number   INTEGER  
,Order_Date        DATE  
,Order_Total       Decimal (10,2)  
)  
PRIMARY INDEX(Order_Number)  
PARTITION BY RANGE_N  
(Order_Date BETWEEN  
  Date '2005-01-01' AND date '2009-12-31'  
    EACH INTERVAL '1' MONTH,  
  Date '2010-01-01' AND date '2010-12-31'  
    EACH INTERVAL '1' DAY );
```



The above example shows Older Data Partitioned by Month and Newer Data by Day.

A Visual for Combining Older Data and Newer Data in PPI



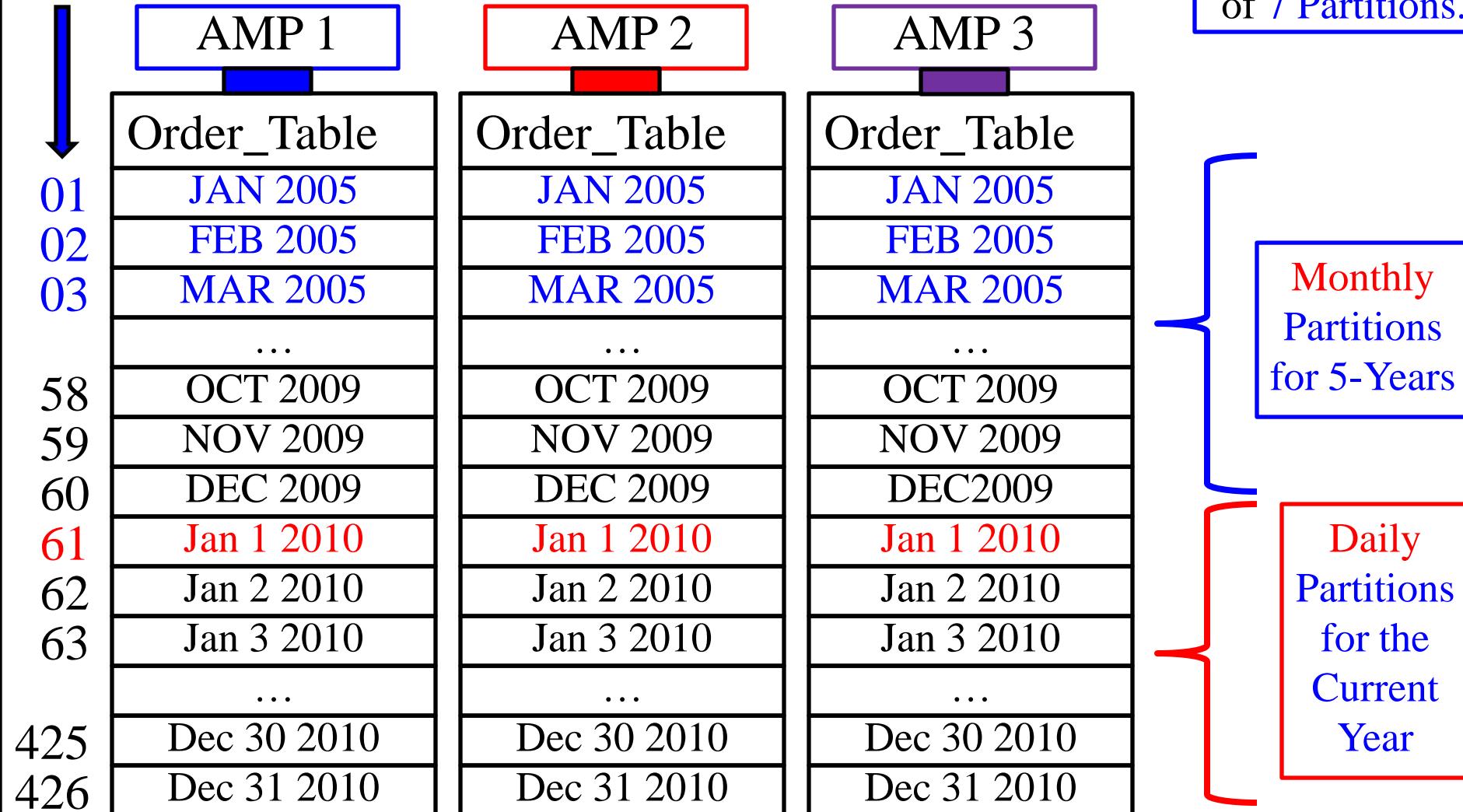
This example combines 5 years of monthly partitions with 1 year of daily partitions.

The SQL on Combining Older Data and Newer Data in PPI

```
SELECT * FROM Order_Table  
WHERE Order_Date BETWEEN '2010-01-01' and '2010-01-07';
```

We do an All-AMPs retrieve by way of 7 Partitions.

Partition #



	AMP 1	AMP 2	AMP 3
Order_Table	Order_Table	Order_Table	Order_Table
01	JAN 2005	JAN 2005	JAN 2005
02	FEB 2005	FEB 2005	FEB 2005
03	MAR 2005	MAR 2005	MAR 2005
...
58	OCT 2009	OCT 2009	OCT 2009
59	NOV 2009	NOV 2009	NOV 2009
60	DEC 2009	DEC 2009	DEC 2009
61	Jan 1 2010	Jan 1 2010	Jan 1 2010
62	Jan 2 2010	Jan 2 2010	Jan 2 2010
63	Jan 3 2010	Jan 3 2010	Jan 3 2010
...
425	Dec 30 2010	Dec 30 2010	Dec 30 2010
426	Dec 31 2010	Dec 31 2010	Dec 31 2010

Multi-Level Partitioning Combining Range_N and Case_N

```
CREATE TABLE Order_TableMLPPI  
(Order_Number      INTEGER      Not Null  

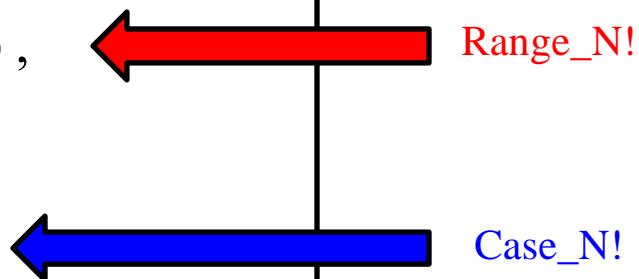
```

```
PARTITION BY (RANGE_N (Order_Date BETWEEN  
Date '2012-01-01' AND date '2012-12-31'  
EACH INTERVAL '1' DAY),
```

```
CASE_N (Order_Total < 1000,  
        Order_Total < 5000,  
        Order_Total < 10000,  
        Order_Total < 20000,  
        No Case));
```

Up to 15 Levels of
Multi-Level Partitioning
and the levels can be

- 1) RANGE_N
- 2) CASE_N



This type of partitioning is called a MULTI-LEVEL PPI. It is a Partition within a Partition. The top partition is the only one you can ALTER so keep that in mind. With Multi-Level partitioning you can combine up to 15 Case_N or Range_N partitions within partitions. NO Simple Partitioning can be used in Multi-Level Partitioning.

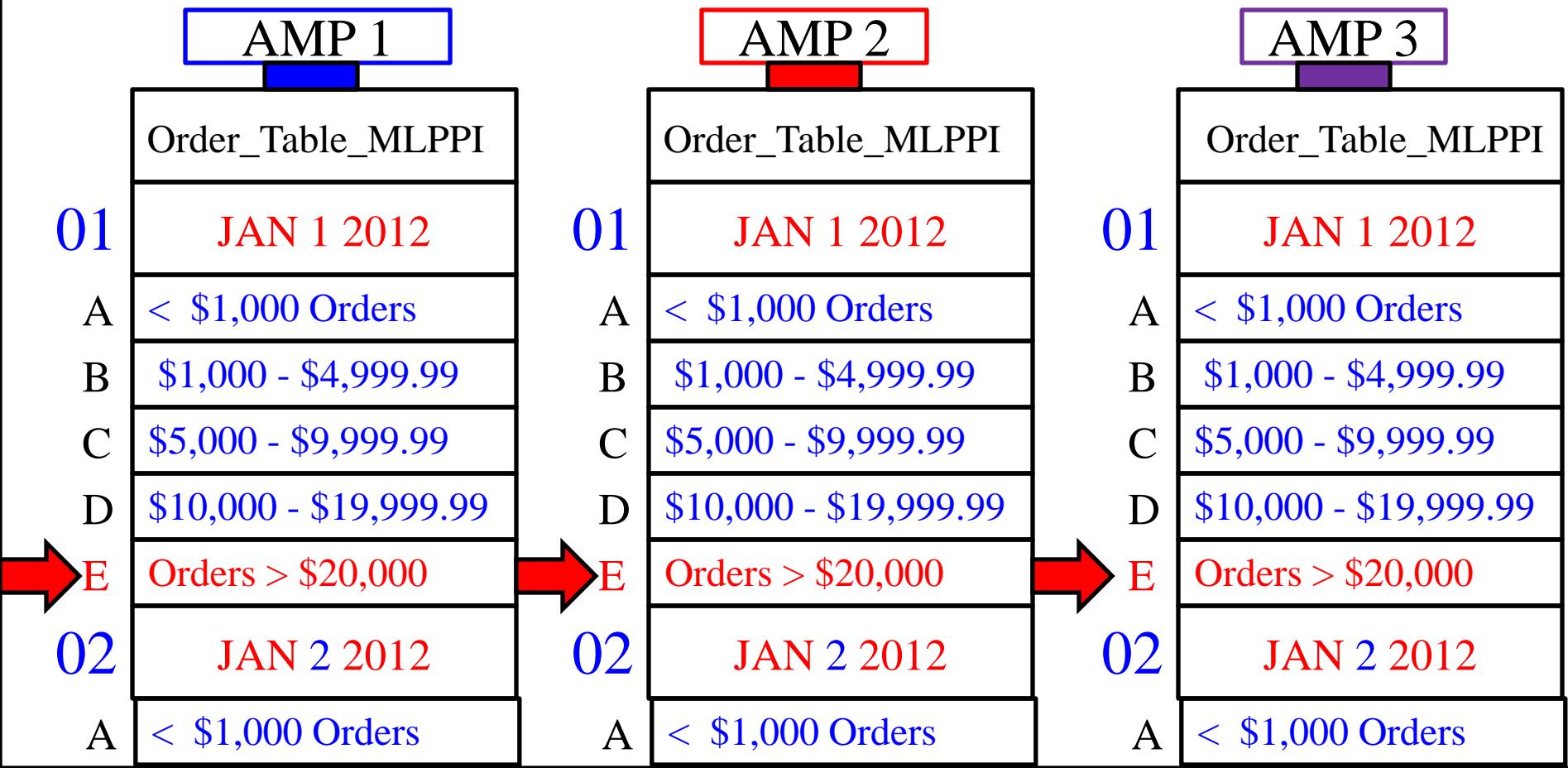
A Visual of Multi-Level Partitioning

Partition #	AMP 1	Partition #	AMP 2	Partition #	AMP 3
01	Order_Table_MLPPI	01	Order_Table_MLPPI	01	Order_Table_MLPPI
JAN 1 2012		JAN 1 2012		JAN 1 2012	
A < \$1,000 Orders		A < \$1,000 Orders		A < \$1,000 Orders	
B \$1,000 - \$4,999.99		B \$1,000 - \$4,999.99		B \$1,000 - \$4,999.99	
C \$5,000 - \$9,999.99		C \$5,000 - \$9,999.99		C \$5,000 - \$9,999.99	
D \$10,000 - \$19,999.99		D \$10,000 - \$19,999.99		D \$10,000 - \$19,999.99	
E Orders > \$20,000		E Orders > \$20,000		E Orders > \$20,000	
02	JAN 2 2012	02	JAN 2 2012	02	JAN 2 2012
A < \$1,000 Orders		A < \$1,000 Orders		A < \$1,000 Orders	
B \$1,000 - \$4,999.99		B \$1,000 - \$4,999.99		B \$1,000 - \$4,999.99	
C \$5,000 - \$9,999.99		C \$5,000 - \$9,999.99		C \$5,000 - \$9,999.99	
D \$10,000 - \$19,999.99		D \$10,000 - \$19,999.99		D \$10,000 - \$19,999.99	
E Orders > \$20,000		E Orders > \$20,000		E Orders > \$20,000	
03	03	03

The SQL on a Multi-Level Partitioned Primary Index

```
SELECT * FROM Order_Table_MLPPI  
WHERE Order_Date = '2012-01-01' and Order_Total > 20000 ;
```

We do an All-AMPs retrieve by way of a Single Partition.



NON-Unique Primary Indexes (NUPI) in PPI

```
CREATE Set TABLE Order_Table_CPPI
(Order_Number      INTEGER  Not Null
,Customer_Number   INTEGER
,Order_Date        DATE
,Order_Total       Decimal (10,2)
) PRIMARY INDEX(Order_Number)
PARTITION BY CASE_N
(Order_Total < 1000,
 Order_Total < 5000,
 Order_Total < 10000,
 Order_Total < 20000,
 NO Case OR UNKNOWN);
```

RULE 1:

A Primary Index must be Non-Unique on all Partitioned tables unless the Primary Index column is also used to define the Partition.

Why were we forced to make the Primary Index of Order_Number a Non-Unique Primary Index (NUPI)? This is because the PRIMARY INDEX (Order_Number) was not part of the Partition. That is why most PPI tables are defined as NUPI.

PPI Table with a Unique Primary Index (UPI)

```
CREATE Set TABLE Order_Table_CPPI
(Order_Number      INTEGER  Not Null
,Customer_Number  INTEGER
,Order_Date       DATE
,Order_Total      Decimal (10,2)
) UNIQUE PRIMARY INDEX(Order_Number, Order_Total)
PARTITION BY CASE_N
( Order_Total < 1000,
  Order_Total < 5000,
  Order_Total < 10000,
  Order_Total < 20000,
NO Case OR UNKNOWN);
```

We were able to make the Primary Index of (Order_Number, Order_Total) a Unique Primary Index (UPI) because Order_Total was part of the Primary Index and part of the Partition.

Tricks for Non-Unique Primary Indexes (NUPI)

```
CREATE Set TABLE Order_Table_Tricks  
  (Order_Number      INTEGER  Not Null  
   ,Customer_Number  INTEGER  
   ,Order_Date       DATE  
   ,Order_Total      Decimal (10,2)  
 ) PRIMARY INDEX(Order_Number)  
 PARTITION BY CASE_N  
   (Order_Total < 1000,  
    Order_Total < 5000,  
    Order_Total < 10000,  
    Order_Total < 20000,  
    NO Case OR UNKNOWN);
```

NUPI

```
CREATE UNIQUE INDEX(Order_Number)  
on Order_Table_Tricks ;
```

USI

```
CREATE INDEX(Order_Number)  
on Order_Table_Tricks ;
```

NUSI

You can create a secondary index (USI or NUSI) on the Primary Index to either ensure uniqueness or for faster retrieval. Use either one depending on your needs.

Character Based PPI for RANGE_N

```
CREATE TABLE Employee_Char
(Employee_No          INTEGER
,Dept_No             INTEGER
,First_Name          CHAR(20)
,Last_Name           VARCHAR(20)
,Salary              Decimal(10,2)
) PRIMARY INDEX(Employee_No)
PARTITION BY RANGE_N
```

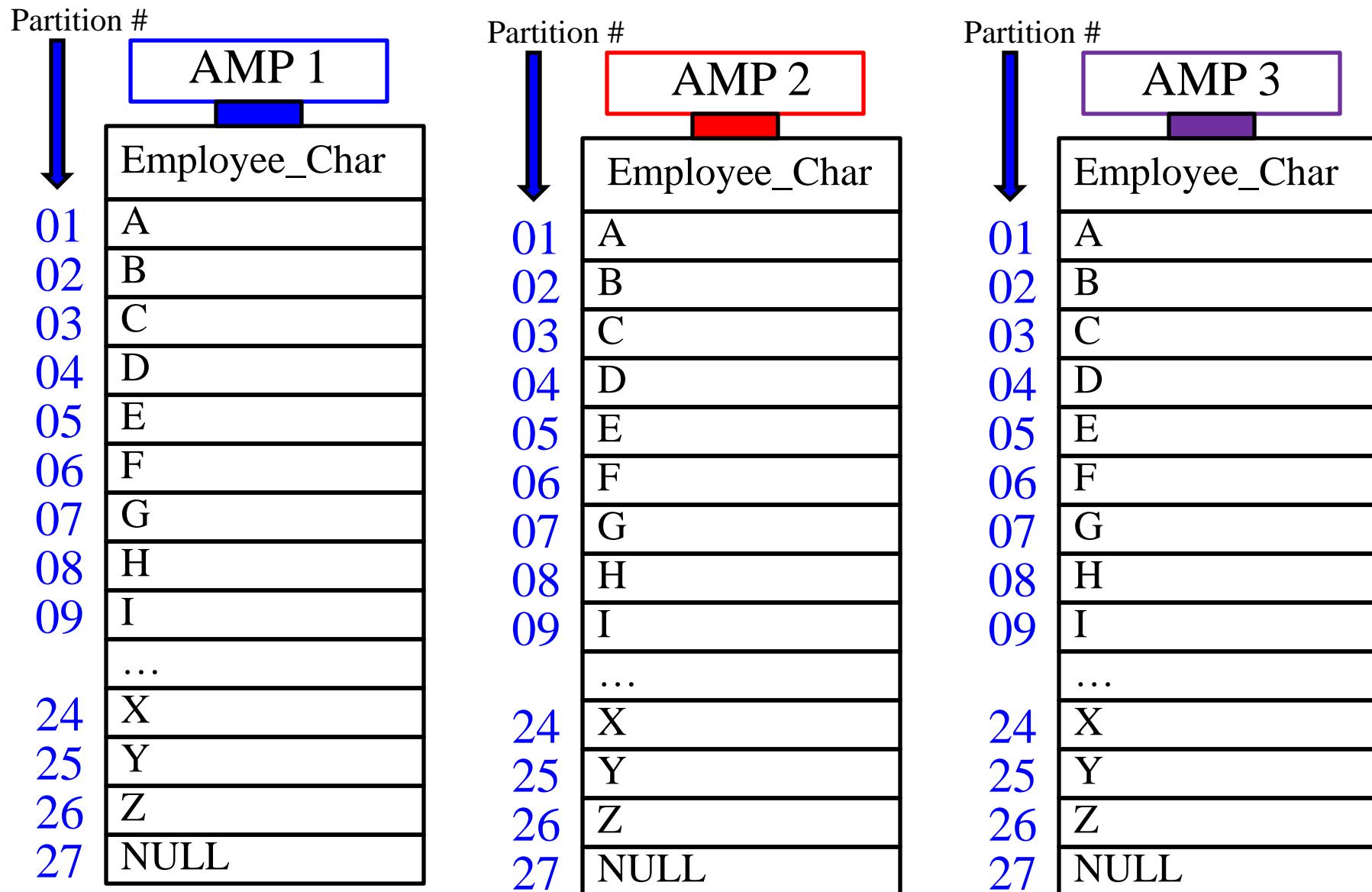
(Last_Name Between

```
(‘A’, ‘B’, ‘C’, ‘D’, ‘E’, ‘F’, ‘G’, ‘H’, ‘I’,
‘J’, ‘K’, ‘L’, ‘M’, ‘N’, ‘O’, ‘P’, ‘Q’, ‘R’,
‘S’, ‘T’, ‘U’, ‘V’, ‘W’, ‘X’, ‘Y’, ‘Z’
AND ‘ZZ’, UNKNOWN));
```

```
SELECT *
FROM Employee_Table_Char
WHERE Last_Name LIKE ‘C%’;
```

Character Based PPI (Teradata Version 13.10 and beyond) paves the way for Range Queries that are used in conjunction with the LIKE command.

A Visual for Character-Based PPI for RANGE_N

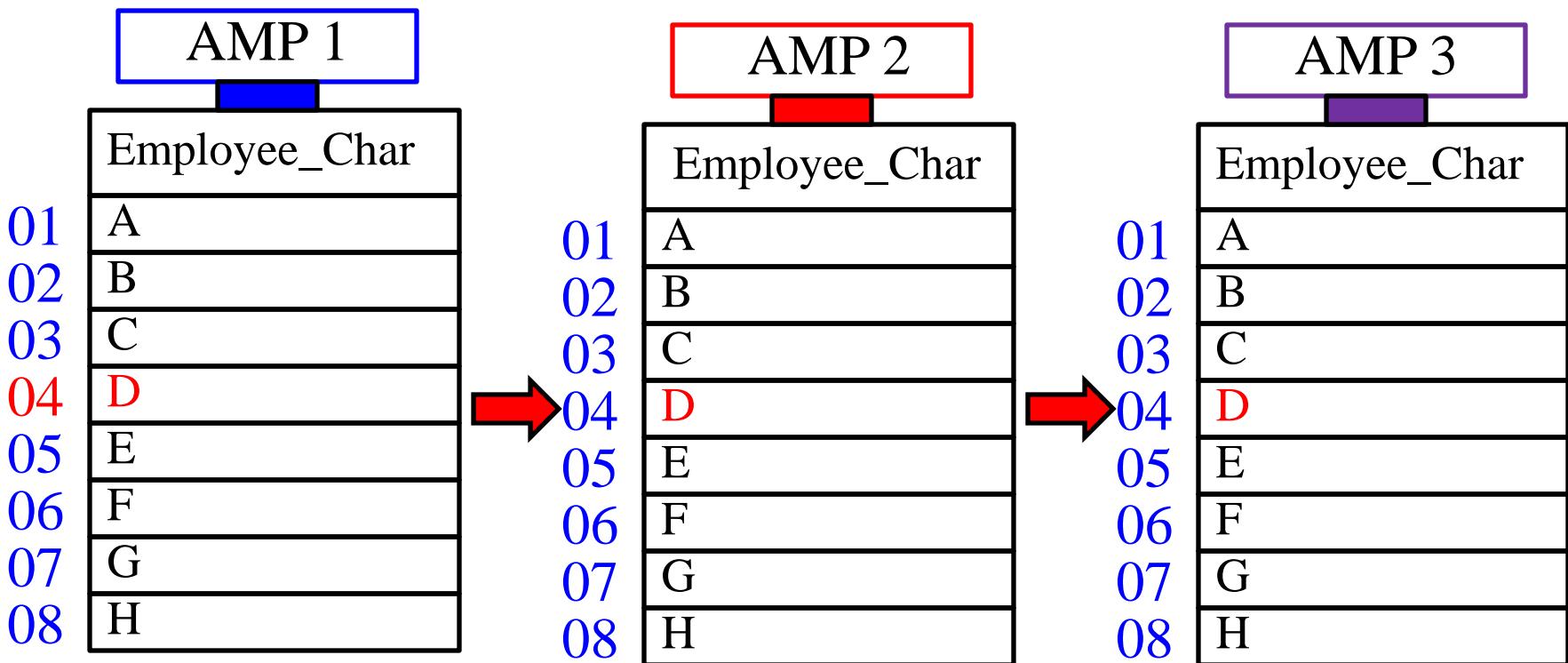


Each letter in the alphabet gets their own partition to easily track Last_Name.

The SQL on Character-Based PPI for RANGE_N

```
SELECT * FROM Employee_Char  
WHERE Last_Name LIKE 'DAV%'
```

We do an All-AMPs retrieve by way of a Single Partition.



Only one partition is read on each AMP and that is the partition for the names that start with the letter 'D'. The query above finds all last names that start with the letters DAV.

Character-Based PPI for CASE_N

```
CREATE TABLE Product_Table_Char
(Product_ID          INTEGER
,Product_Name        (CHAR(20)) NOT CASESPECIFIC
,Product_Cost        Decimal(10,2)
,Product_Description VARCHAR(100)
) PRIMARY INDEX(Product_ID)
PARTITION BY CASE_N
(Product_Name < 'Apples'
Product_Name < 'Bananas'
Product_Name < 'Cantaloupe'
Product_Name < 'Grapes'
Product_Name < 'Lettuce'
Product_Name < 'Mangos'
Product_Name >= 'Mangos' and <= 'Tomatoes') ;
```

```
SELECT * FROM Product_Table_Char
WHERE Product_Name BETWEEN 'Apples' and 'Grapes' ;
```

Dates and Character-Based Multi-Level PPI

```
CREATE TABLE Claims_Table_PPI
(Claim_ID      INTEGER NOT NULL
,Claim_Date    DATE    NOT NULL
,State_ID      CHAR(2) NOT NULL
,CL_Details    VARCHAR(1000) NOT NULL
) PRIMARY INDEX(Claim_ID)
PARTITION BY (RANGE_N
(Claim_Date BETWEEN DATE '2012-01-01'
                AND     DATE '2012-12-31'
                EACH INTERVAL '1' DAY)
,RANGE_N (State_ID Between
('AL', 'AK', 'AZ', 'AR', 'CA', 'CO', 'CT', 'DE', 'FL',
 'GA', 'HI', 'ID', 'IL', 'IN', 'IA', 'KS', 'KY', 'LA',
 'ME', 'MD', 'MA', 'MI', 'MN', 'MS', 'MT', 'NE',
 'NV', 'NH', 'NJ', 'NM', 'NY', 'NC', 'ND', 'OH', 'OK',
 'OR', 'PA', 'RI', 'SC', 'SD', 'TN', 'TX', 'UT', 'VT',
 'VA', 'WA', 'WV', 'WI', 'WY'));
```

This Multi-Level PPI table uses both Range_N on both Dates and Character based PPI.

TIMESTAMP Partitioning

```
CREATE TABLE Order_Table_TS  
  (Order_Number      INTEGER NOT NULL  
   ,Customer_Number  INTEGER NOT NULL  
   ,Order_Date_Time  TIMESTAMP(0) WITH TIME ZONE  
   ,Order_Total       Decimal(10,2)  
  ) PRIMARY INDEX(Order_Number)  
    PARTITION BY (RANGE_N  
    (CAST (Order_Date_Time as DATE AT 0)  
     BETWEEN DATE '2012-01-01' AND DATE '2012-12-31'  
     EACH INTERVAL '1' Month ) ;
```

```
SELECT * FROM Order_Table_TS  
WHERE Order_Date_Time BETWEEN  
TIMESTAMP '2012-01-13 00:00:00-08:00' AND  
TIMESTAMP '2012-01-13 23:59:59-08:00' ;
```

Find all Orders
placed on the day
January 13, 2012

TIMESTAMP partitioning is available with Teradata V13.10. Notice the DATE at 0 in the CREATE statement. This means the Timestamp is Deterministic, which means the system will know your Time Zone and retrieve the rows from your point of time.

Using CURRENT_DATE to define a PPI

```
CREATE TABLE Order_Table_PPI2
(Order_Number      INTEGER Not Null
,Customer_Number  INTEGER
,Order_Date       DATE FORMAT 'YYYY-MM-DD'
,Order_Total      Decimal (10,2)
) PRIMARY INDEX(Order_Number)
PARTITION BY RANGE_N (Order_Date
                      BETWEEN date '2009-01-01' AND date '2013-12-31'
                      EACH INTERVAL '1' Month);
```

The year is 2012 when both these tables were created. Each statement CREATES a PPI Table with 3 previous years, 1 current year, and 1 future year for a total of 5 years of data.

```
CREATE TABLE Order_Table_PPI3
(Order_Number      INTEGER Not Null
,Customer_Number  INTEGER
,Order_Date       DATE FORMAT 'YYYY-MM-DD'
,Order_Total      Decimal (10,2)
) PRIMARY INDEX(Order_Number)
PARTITION BY RANGE_N (Order_Date BETWEEN
                      CAST((EXTRACT(YEAR FROM CURRENT_DATE) -3 -1900) * 10000 +0101) AS DATE)
                      AND
                      CAST((EXTRACT(YEAR FROM CURRENT_DATE) +1 - 1900) * 10000 + 1231) AS DATE)
                      EACH INTERVAL '1' Month);
```

This PPI example uses the CURRENT_DATE formula available in Teradata Version 13.10. Why do it this way? Turn the page.

Both CREATE statements above demand 5 years of partitioned data from 2009-2013.

ALTER to CURRENT_DATE the next year

```
CREATE TABLE Order_Table_PPI3  
(Order_Number      INTEGER  Not Null  
,Customer_Number   INTEGER  
,Order_Date        DATE FORMAT 'YYYY-MM-DD'  
,Order_Total       Decimal (10,2)  
) PRIMARY INDEX(Order_Number)  
PARTITION BY RANGE_N (Order_Date BETWEEN  
CAST((EXTRACT(YEAR FROM CURRENT_DATE) -3 -1900) * 10000 +0101) AS DATE)  
AND  
CAST((EXTRACT(YEAR FROM CURRENT_DATE) +1 - 1900) * 10000 + 1231) AS DATE)  
EACH INTERVAL '1' Month);
```

This PPI example uses the CURRENT_DATE formula available in Teradata Version 13.10. Why do It this way?

When 2013 comes along you can easily ALTER the table using TO CURRENT.

ALTER TABLE Order_Table_PPI3 TO CURRENT WITH DELETE ;

In Teradata V13.10 the system allowed for CURRENT_DATE to be utilized in the PPI CREATE statement. This allowed for easy altering of the data and deleting of partitions. The ALTER Table command can alter to reflect 5 years, now going from 1010-2014.

ALTER to CURRENT_DATE with Save

```
CREATE TABLE Order_Table_PPI3  
(Order_Number      INTEGER  Not Null  
,Customer_Number   INTEGER  
,Order_Date        DATE FORMAT 'YYYY-MM-DD'  
,Order_Total       Decimal (10,2)  
) PRIMARY INDEX(Order_Number)  
PARTITION BY RANGE_N (Order_Date BETWEEN  
CAST(((EXTRACT(YEAR FROM CURRENT_DATE) -3 -1900) * 10000 +0101) AS DATE)  
AND  
CAST(((EXTRACT(YEAR FROM CURRENT_DATE) +1 - 1900) * 10000 + 1231) AS DATE)  
EACH INTERVAL '1' Month);
```

This PPI example uses the CURRENT_DATE formula available in Teradata Version 13.10. Why do it this way?

When 2013 comes along you can easily ALTER the table using TO CURRENT.

```
ALTER TABLE Order_Table_PPI3 TO CURRENT WITH  
INSERT INTO Order_Table_History ;
```

Notice we are ALTERING the table to CURRENT, but instead of deleting the data we are moving the old data into a History file called Order_Table_History.

Altering a PPI Table to Add or Drop Partitions

Creating a PPI Table

```
CREATE TABLE Order_TablePPI  
(Order_Number      INTEGER Not Null  
PRIMARY INDEX(Order_Number)
```

```
PARTITION BY RANGE_N  
      date '2010-01-01'  
    AND date '2010-12-31'  
EACH INTERVAL '1' DAY);
```

ALTERING a PPI Table

```
ALTER TABLE Order_TablePPI  
MODIFY PRIMARY INDEX  
DROP RANGE BETWEEN  
      DATE '2010-01-01'  
    AND DATE '2010-12-31'  
      EACH INTERVAL '1' DAY  
ADD RANGE BETWEEN  
      DATE '2011-01-01'  
    AND DATE '2011-12-31'  
      EACH INTERVAL '1' DAY  
WITH DELETE ;
```

What will happen to the 2010 data after the ALTER?

Deleting a Partition

Creating a PPI Table

```
CREATE TABLE Order_TablePPI
(Order_Number      INTEGER Not Null
,Customer_Number  INTEGER
,Order_Date       DATE
,Order_Total      Decimal (10,2))
PRIMARY INDEX(Order_Number)
```

```
PARTITION BY RANGE_N
(Order_Date BETWEEN
     date '2010-01-01'
    AND date '2010-12-31'
EACH INTERVAL '1' DAY);
```

ALTERING a PPI Table

```
ALTER TABLE Order_TablePPI
MODIFY PRIMARY INDEX
DROP RANGE BETWEEN
    DATE '2010-01-01'
    AND DATE '2010-12-31'
    EACH INTERVAL '1' DAY
ADD RANGE BETWEEN
    DATE '2011-01-01'
    AND DATE '2011-12-31'
    EACH INTERVAL '1' DAY
WITH DELETE ;
```

The 2010 data after the ALTER will be deleted from the table.

Deleting a Partition and Saving its contents

Creating a PPI Table

```
CREATE TABLE Order_TablePPI
(Order_Number      INTEGER Not Null
,Customer_Number  INTEGER
,Order_Date        DATE
,Order_Total       Decimal (10,2))
PRIMARY INDEX(Order_Number)
```

```
PARTITION BY RANGE_N
(Order_Date BETWEEN
     date '2010-01-01'
  AND date '2010-12-31'
EACH INTERVAL '1' DAY);
```

ALTERING a PPI Table

```
ALTER TABLE Order_TablePPI
MODIFY PRIMARY INDEX
DROP RANGE BETWEEN
    DATE '2010-01-01'
  AND DATE '2010-12-31'
    EACH INTERVAL '1' DAY
ADD RANGE BETWEEN
    DATE '2011-01-01'
  AND DATE '2011-12-31'
    EACH INTERVAL '1' DAY
WITH INSERT INTO
    Order_Table_History ;
```

The 2010 data after the ALTER will be deleted from the table, but the deleted data is stored in Order_Table_History.

Using the PARTITION Keyword in your SQL

```
SELECT PARTITION as "Partition Number"  
      ,Count(*)      as "Rows in Partition"  
FROM   Order_Table_PPI  
GROUP BY 1  
ORDER BY 1 ;
```

Partition Number	Rows in Partition
1	1000
2	1200
3	1100
4	1022
5	1121
6	1132
7	1300
8	1500
9	1610

You can find out how many partitions you have and the number of rows from the above query. You won't see information about empty partitions.

SQL for RANGE_N

```
SELECT RANGE_N (
    Calendar_Date BETWEEN DATE '2012-01-01' AND DATE '2012-04-30'
                      EACH INTERVAL '1' MONTH) AS "Partition"
    ,MIN(Calendar_Date) as "Begin Date"
    ,MAX(Calendar_Date) as "End Date"
FROM Sys_Calendar.Calendar
WHERE Calendar_Date BETWEEN DATE '2012-01-01' and DATE '2012-04-30'
GROUP BY 1
ORDER BY 1 ;
```

<u>Partition</u>	<u>Begin Date</u>	<u>End Date</u>
1	2012-01-01	2012-01-31
2	2012-02-01	2012-02-28
3	2012-03-01	2012-03-31
4	2012-04-01	2012-04-30

You can find out how many partitions you have and the number of rows from the above query. You won't see information about empty partitions.

SQL for CASE_N

```
SELECT CASE_N (
    Order_Total < 1000
    ,Order_Total < 5000
    ,Order_Total < 10000
    ,Order_Total < 20000
    ,NO Case
    ,UNKNOWN) as "Partition"
    ,COUNT(*) as "Counter"
FROM Order_Table_PPI
GROUP BY 1
ORDER BY 1 ;
```

Partition	Counter
1	100
2	125
3	140
4	110
5	3

You can find out how many partitions you have and the number of rows from the above query. You won't see information about empty partitions.

Watch the Video on Partitioning



Tera-Tom Trivia

Tom Coffing graduated with a Bachelor's degree in Speech Communications. Tom spent 10 years in Toastmasters competing in hundreds of speech contests. Tom won speech championships at the city, area, state, and national level.

Click on the **link below** or place it in your browser and watch the video on partitioning.

<http://www.coffingdw.com/TbasicsV12/partitioning.wmv>

Chapter 6

Secondary Indexes

“Hey, every once in awhile the secondary form works better than the original but it's certainly a rarity.”

- Jaime Hernandez

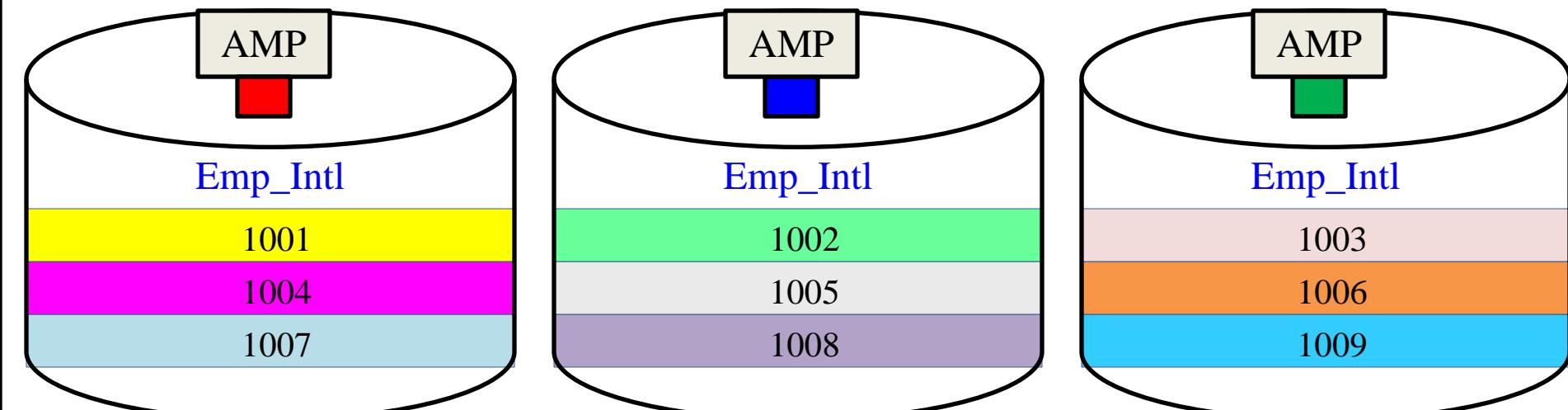
Table of Contents Chapter 6 – Secondary Indexes

- [Review of a Unique Primary Index \(UPI\)](#)
- [Primary Index in the WHERE Clause - Single-AMP Retrieve](#)
- [Review of a Non-Unique Primary Index \(NUPI\)](#)
- [Primary Index in the WHERE Clause - Single-AMP Retrieve](#)
- [Creating a Unique Secondary Index \(USI\)](#)
- [What is in a Unique Secondary Index \(USI\) Subtable?](#)
- [A Unique Secondary Index \(USI\) Subtable is Hashed](#)
- [How the Parsing Engine uses the USI Subtable?](#)
- [A USI is a Two-AMP Operation](#)
- [Review of a Non-Unique Primary Index \(NUPI\)](#)
- [Creating a Non-Unique Secondary Index \(NUSI\)](#)
- [What is in a Unique Secondary Index \(USI\) Subtable?](#)
- [Non-Unique Secondary Index \(NUSI\) Subtable is AMP Local](#)
- [How the Parsing Engine uses the NUSI Subtable?](#)
- [Creating a Value-Ordered NUSI](#)
- [Watch the Videos on Secondary Indexes](#)

Review of a Unique Primary Index (UPI)

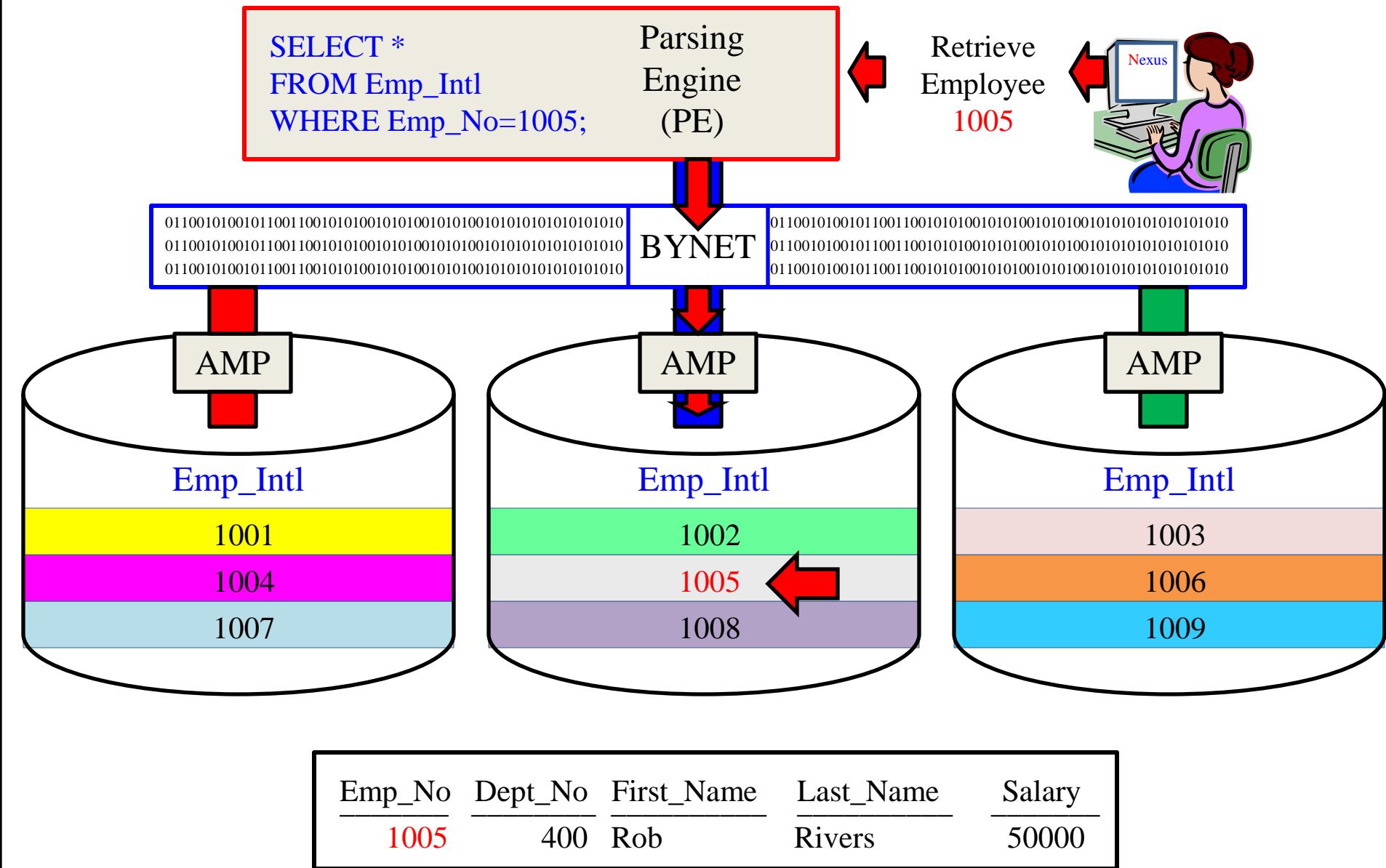
```
CREATE TABLE Emp_Intl
(Emp_No          INTEGER,
Dept_No         SMALLINT,
First_Name      VARCHAR(12),
Last_Name       CHAR(20),
Salary          DECIMAL(10,2))
UNIQUE Primary Index( Emp_No )
```

Unique Primary Index		Emp_Intl			
	Emp_No	Dept_No	First_Name	Last_Name	Salary
	1001	100	Rafael	Minal	90000.00
	1002	200	Maria	Gomez	80000.00
	1003	300	Charl	Kertzel	70000.00
	1004	400	Kyle	Stover	60000.00
	1005	400	Rob	Rivers	50000.00
	1006	300	Inna	Kinski	50000.00
	1007	200	Sushma	Davis	50000.00
	1008	100	Mo	Khan	60000.00
	1009	300	Mo	Swartz	70000.00



A Unique Primary Index(UPI) spreads the rows of a table evenly across the AMPS.

Primary Index in the WHERE Clause - Single-AMP Retrieve

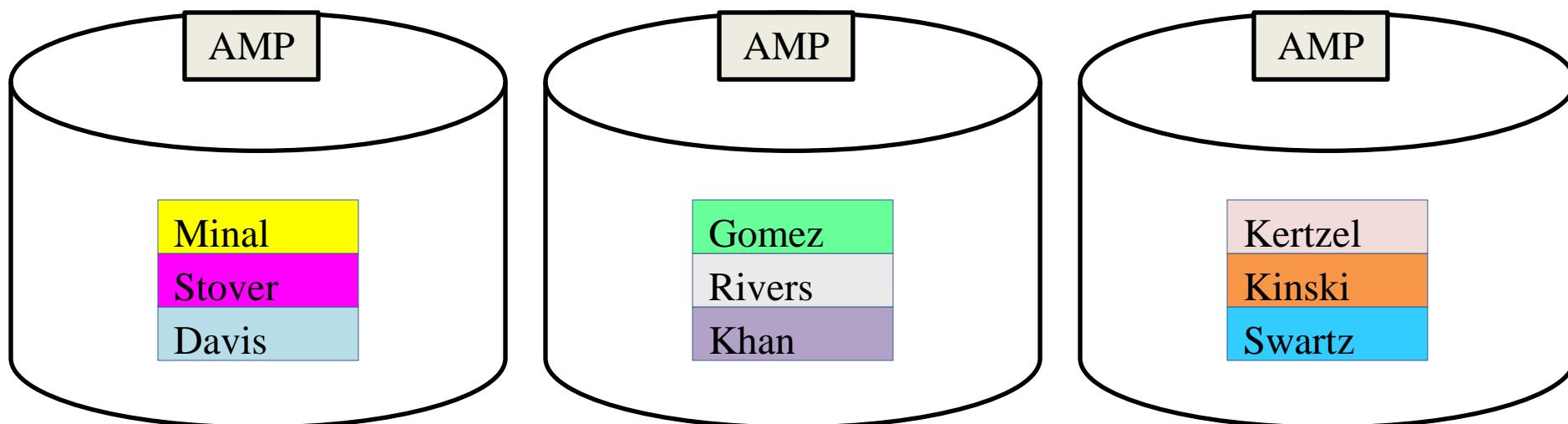


Use the Primary Index column in your SQL WHERE clause and only 1 AMP retrieves.

Review of a Non-Unique Primary Index (NUPI)

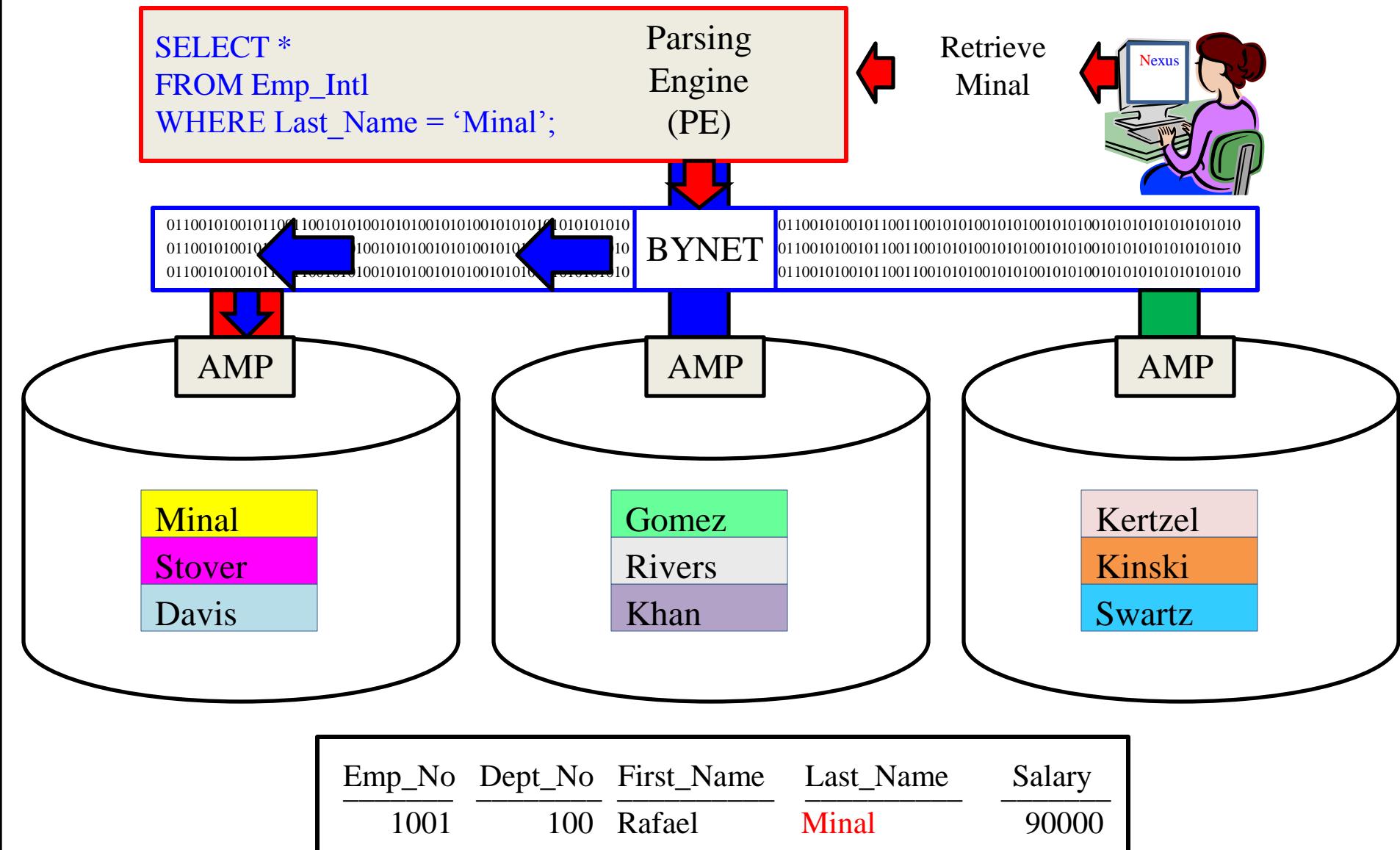
```
CREATE TABLE Emp_Intl
(Emp_No          INTEGER,
Dept_No         SMALLINT,
First_Name      VARCHAR(12),
Last_Name       CHAR(20),
Salary          DECIMAL(10,2))
Primary Index( Last_Name)
```

Emp_Intl			NUPI	
Emp_No	Dept_No	First_Name	Last_Name	Salary
1001	100	Rafael	Minal	90000.00
1002	200	Maria	Gomez	80000.00
1003	300	Charl	Kertzel	70000.00
1004	400	Kyle	Stover	60000.00
1005	400	Rob	Rivers	50000.00
1006	300	Inna	Kinski	50000.00
1007	200	Sushma	Davis	50000.00
1008	100	Mo	Khan	60000.00
1009	300	Mo	Swartz	70000.00



A Non-Unique Primary Index(NUPI) will have duplicates grouped together on the same AMP so data will always be skewed (uneven). The above skew is reasonable and OK.

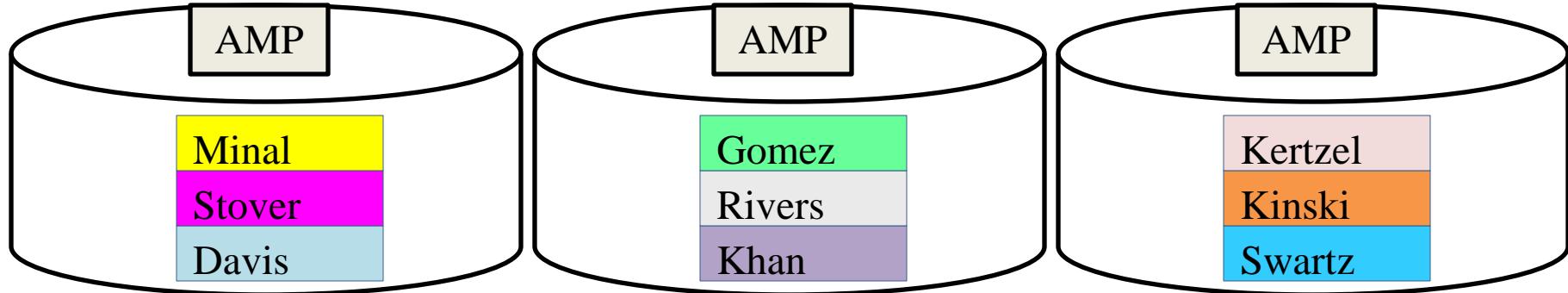
Primary Index in the WHERE Clause - Single-AMP Retrieve



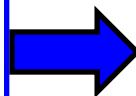
Use the Primary Index column in your SQL WHERE clause and only 1 AMP retrieves.

Creating a Unique Secondary Index (USI)

Before

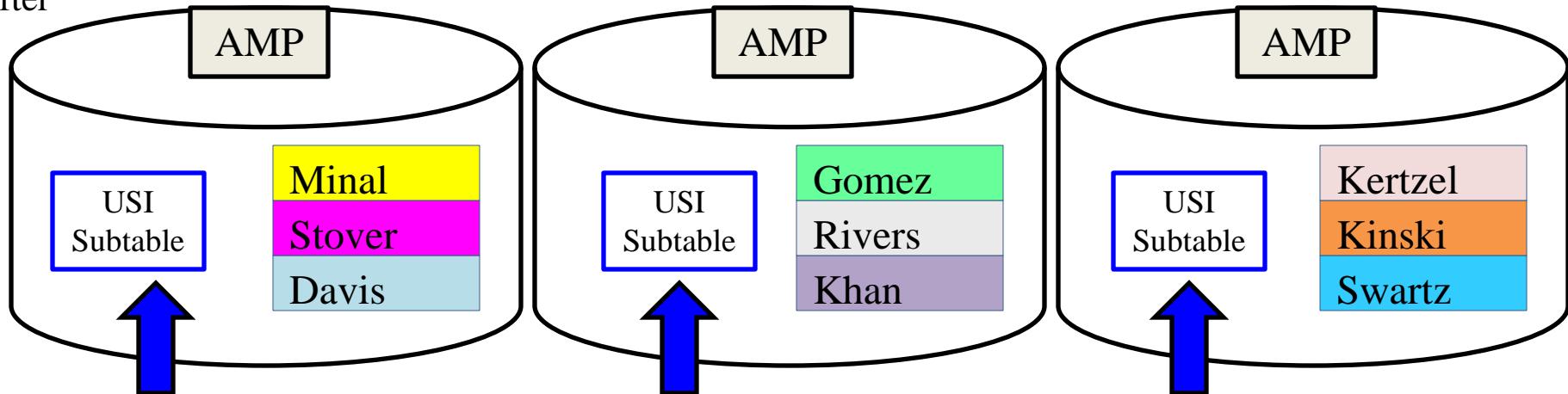


Syntax to
Create a
USI



CREATE UNIQUE INDEX(Emp_No) on Emp_Intl ;

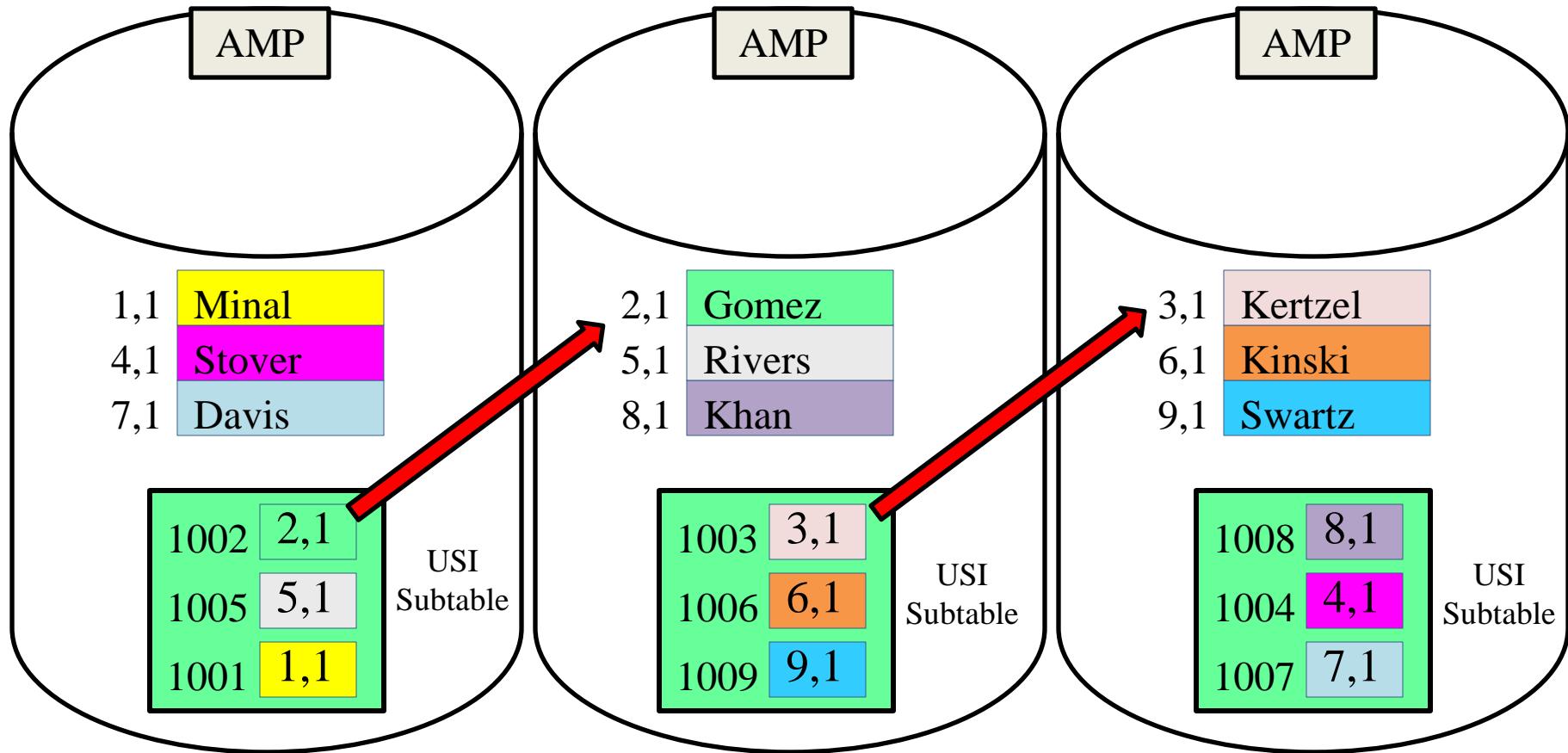
After



A Subtable is created on each AMP

When a Secondary Index is created a subtable is created on each AMP.

What is in a Unique Secondary Index (USI) Subtable?

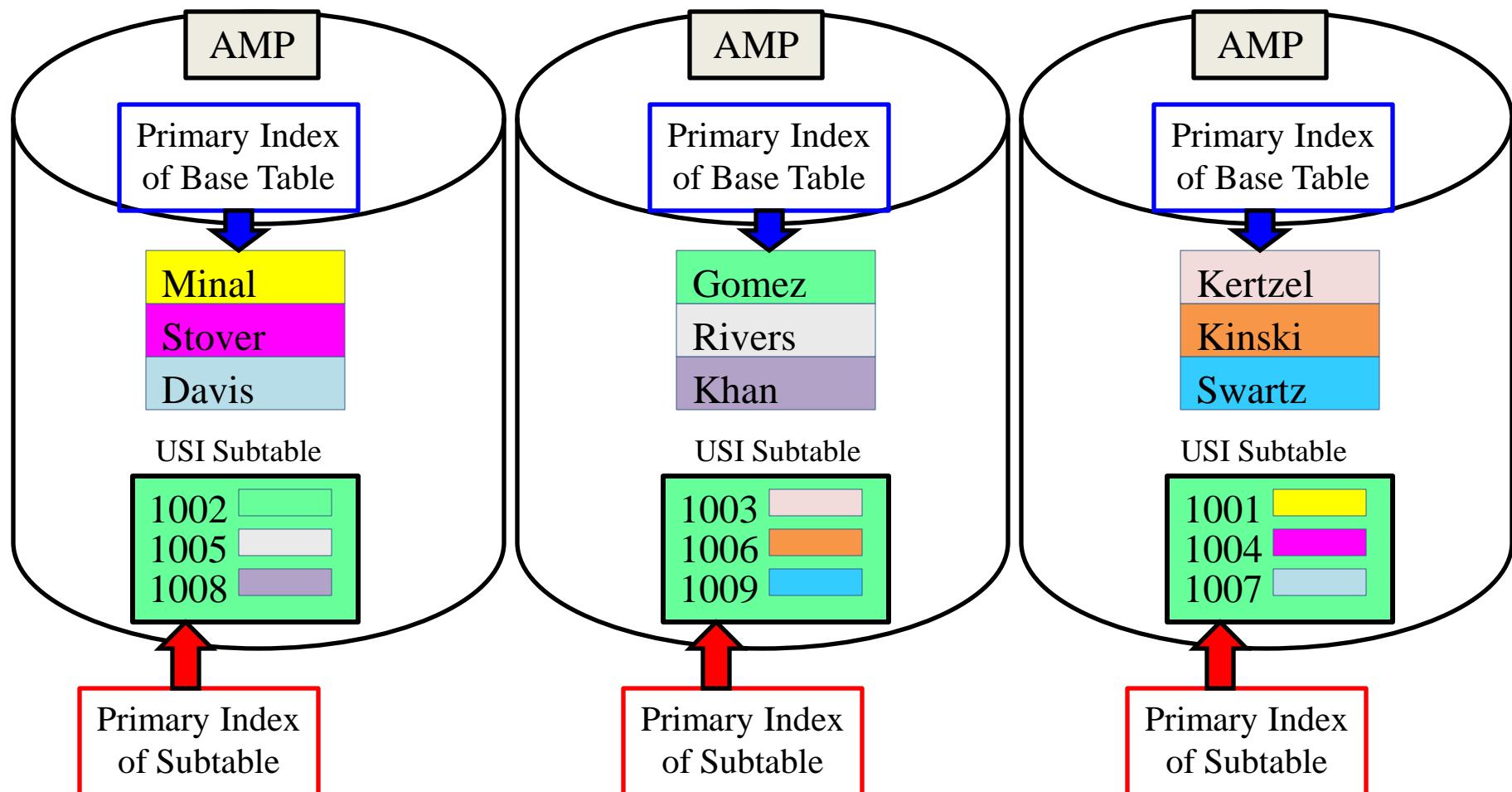


The USI Subtable only contains two columns:

1. **Emp_No** (The USI column)
2. **Row-ID** of the real Primary Index of the base table

Inside the secondary index subtable is **Emp_No** column (USI Column). Then there is the corresponding Row-ID of the real Primary Index of the table.

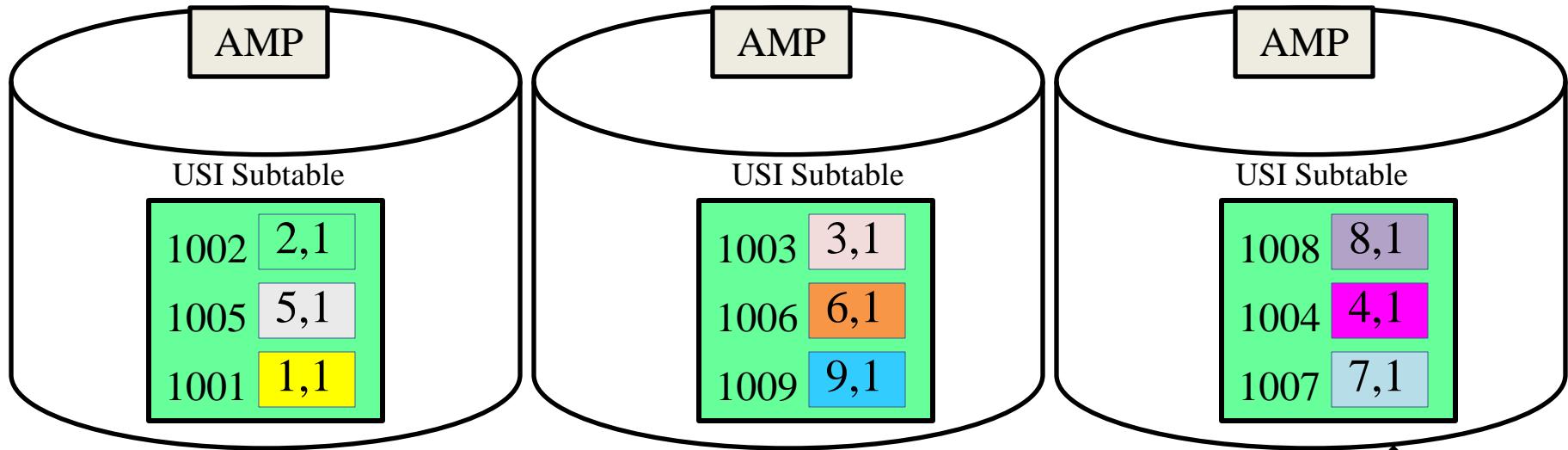
A Unique Secondary Index (USI) Subtable is Hashed



The USI Subtable spreads the rows evenly among the AMPs. The Primary Index of the Subtable is Emp_No (The USI column).

Subtable rows are hashed by their Primary Index (Emp_No) and distributed evenly.

How the Parsing Engine uses the USI Subtable



```
SELECT * FROM Emp_Intl  
WHERE Emp_No = 1004 ;
```

Row-ID of
Base Table

Parsing Engines Plan – A Two-AMP Operation

Emp_No is a Unique Secondary Index!

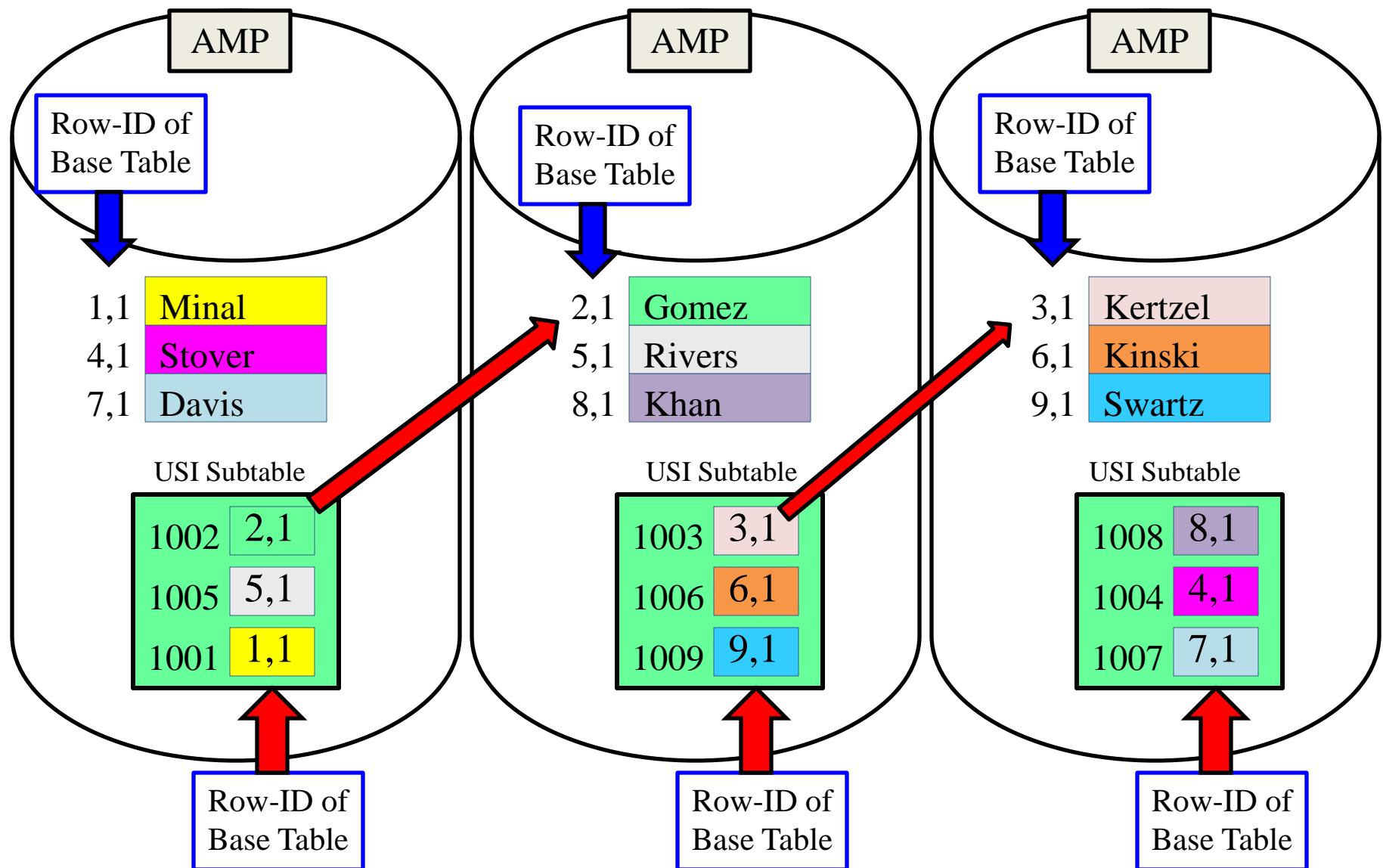
Hash 1004 and see which AMP holds the row in its subtable. (AMP 3)

Contact AMP 3 and have it retrieve row 1004.

Find the real Row-ID of the base table row. (4,1)

Use the Row-ID to find the base table row with a single-AMP retrieve.

A USI is a Two-AMP Operation

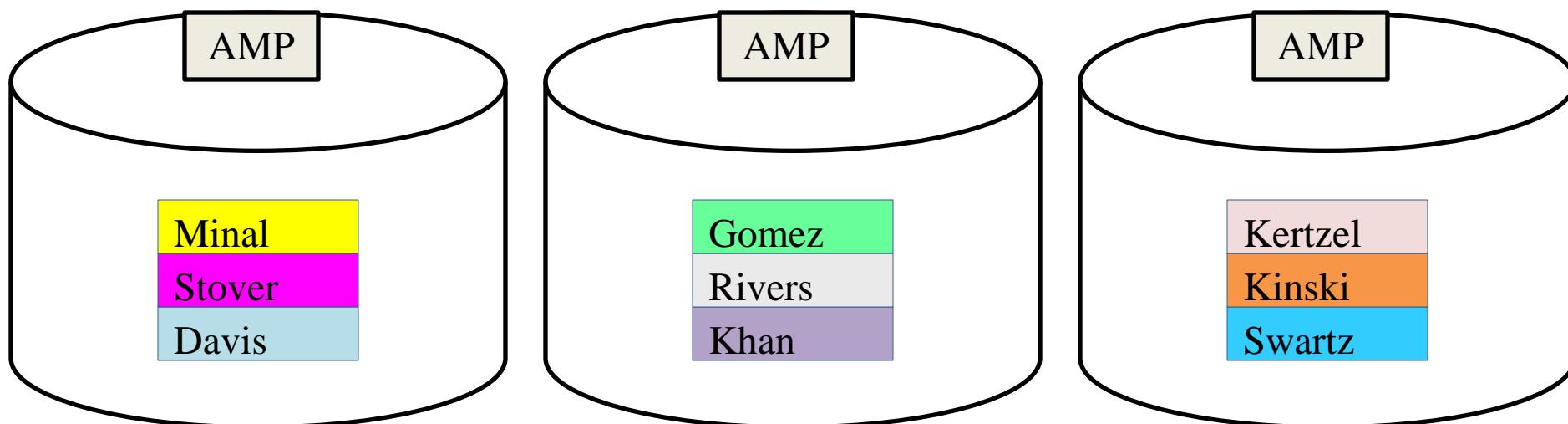


The first AMP is assigned to read the subtable and the second the base table.

Review of a Non-Unique Primary Index (NUPI)

```
CREATE TABLE Emp_Intl
(Emp_No          INTEGER,
Dept_No         SMALLINT,
First_Name      VARCHAR(12),
Last_Name       CHAR(20),
Salary          DECIMAL(10,2))
Primary Index( Last_Name)
```

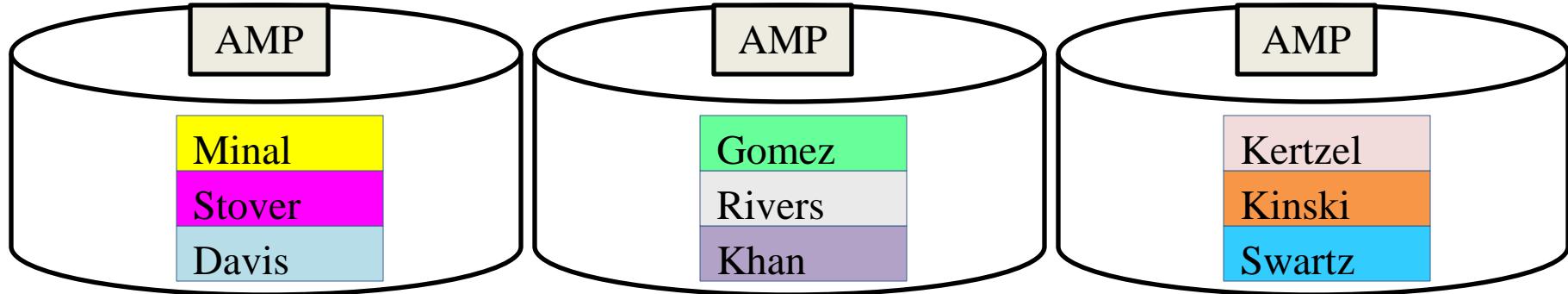
Emp_Intl			NUPI	
Emp_No	Dept_No	First_Name	Last_Name	Salary
1001	100	Rafael	Minal	90000.00
1002	200	Maria	Gomez	80000.00
1003	300	Charl	Kertzel	70000.00
1004	400	Kyle	Stover	60000.00
1005	400	Rob	Rivers	50000.00
1006	300	Inna	Kinski	50000.00
1007	200	Sushma	Davis	50000.00
1008	100	Mo	Khan	60000.00
1009	300	Mo	Swartz	70000.00



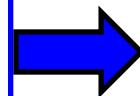
A Non-Unique Primary Index(NUPI) will have duplicates grouped together on the same AMP so data will always be skewed (uneven). The above skew is reasonable and OK.

Creating a Non-Unique Secondary Index (NUSI)

Before

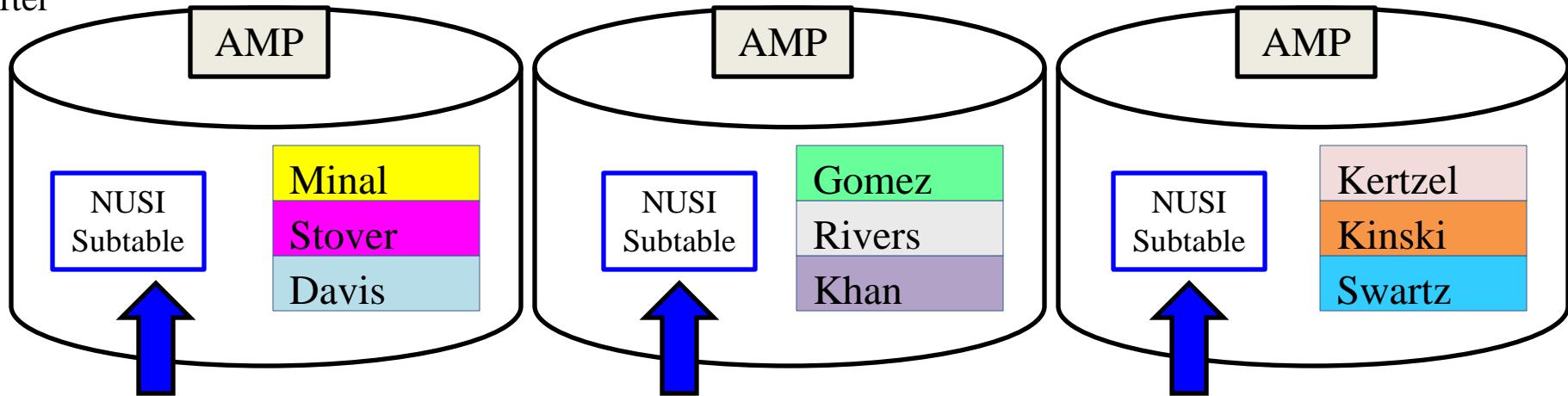


Syntax to
Create a
NUSI



CREATE INDEX(First_Name) on Emp_Intl ;

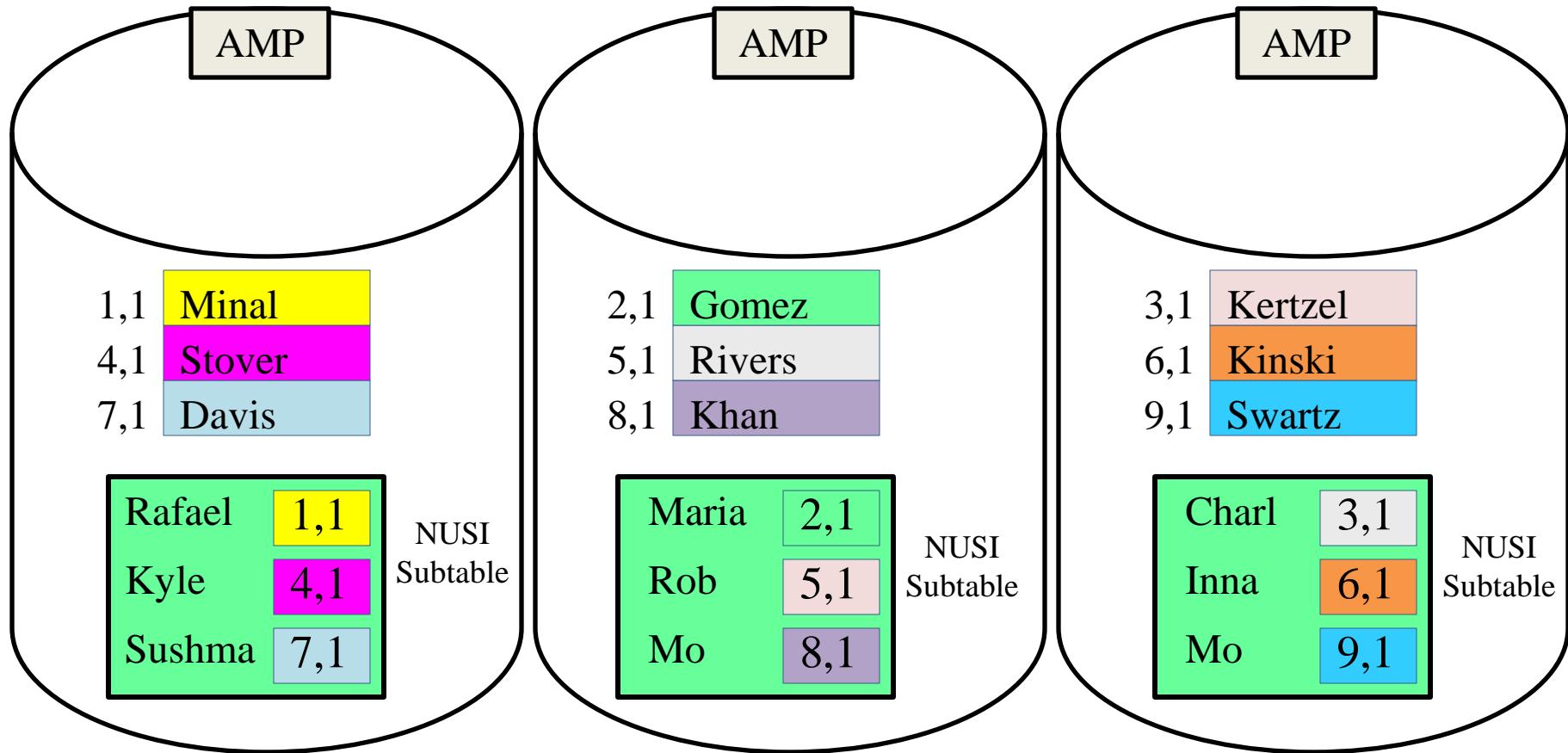
After



A Subtable is created on each AMP

When a Secondary Index is created a subtable is created on each AMP.

What is in a Unique Secondary Index (USI) Subtable?

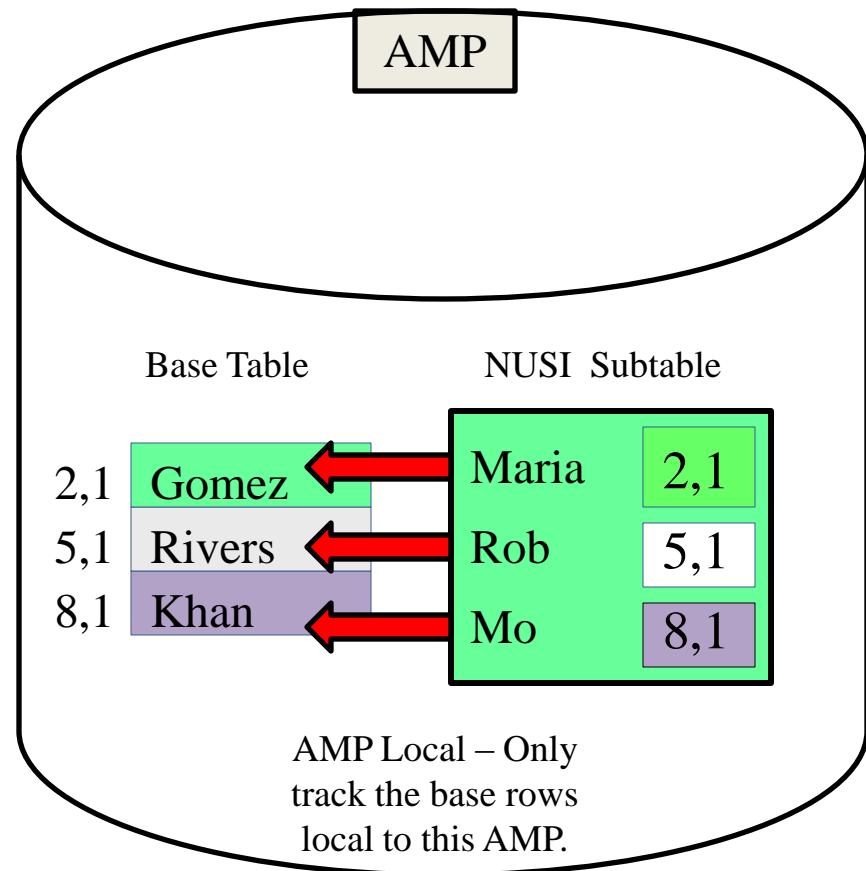
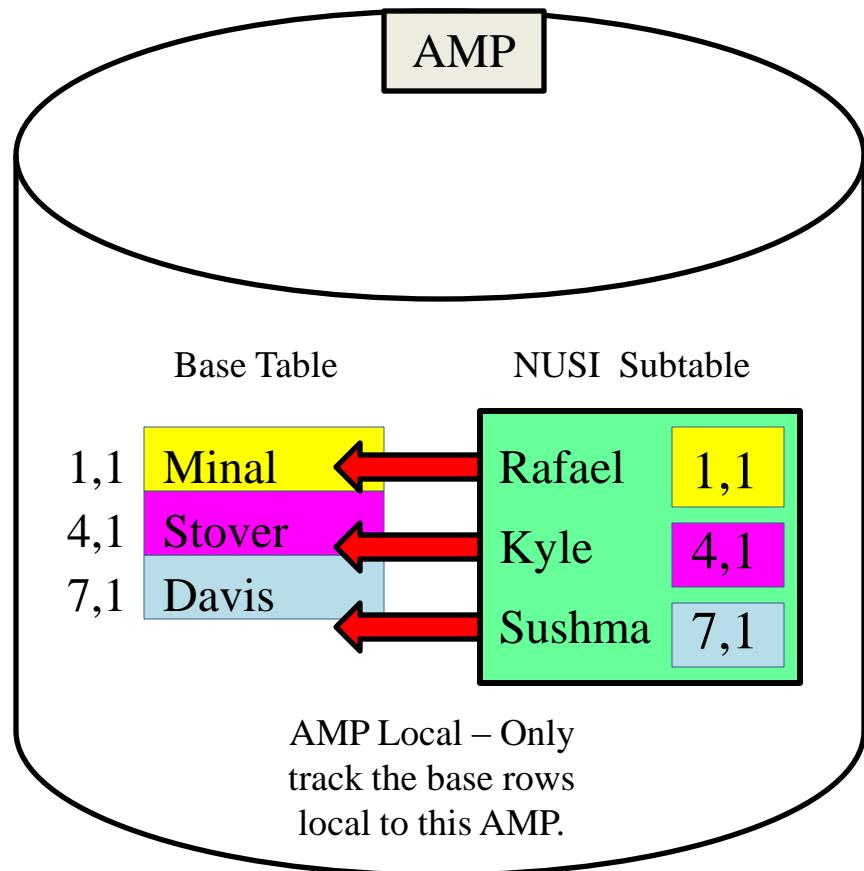


The NUSI Subtable only contains two columns:

1. First_Name (The NUSI column)
2. Row-ID of the real Primary Index of the base table

Inside the secondary index subtable is First_Name column (NUSI Column). Then there is the corresponding Row-ID of the real Primary Index of the table.

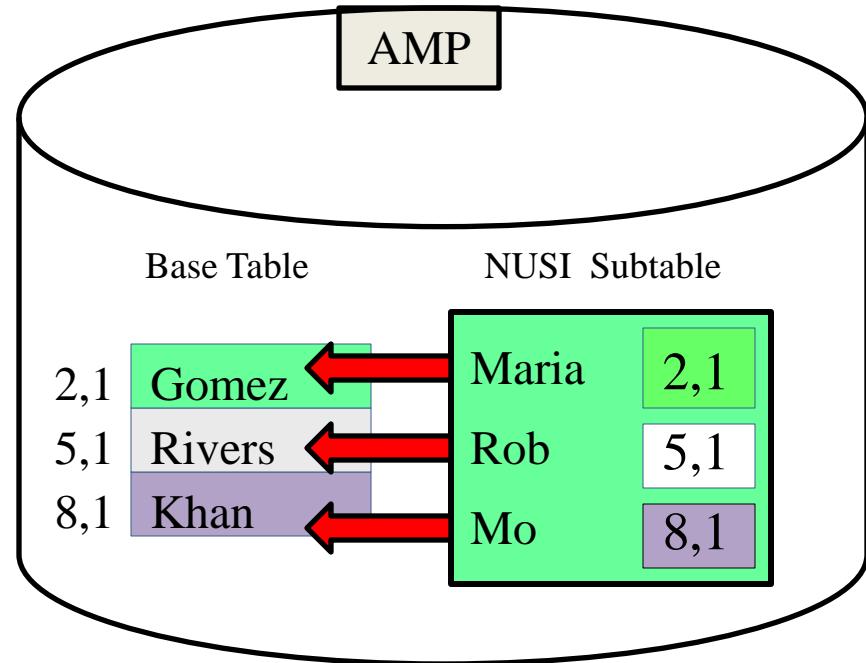
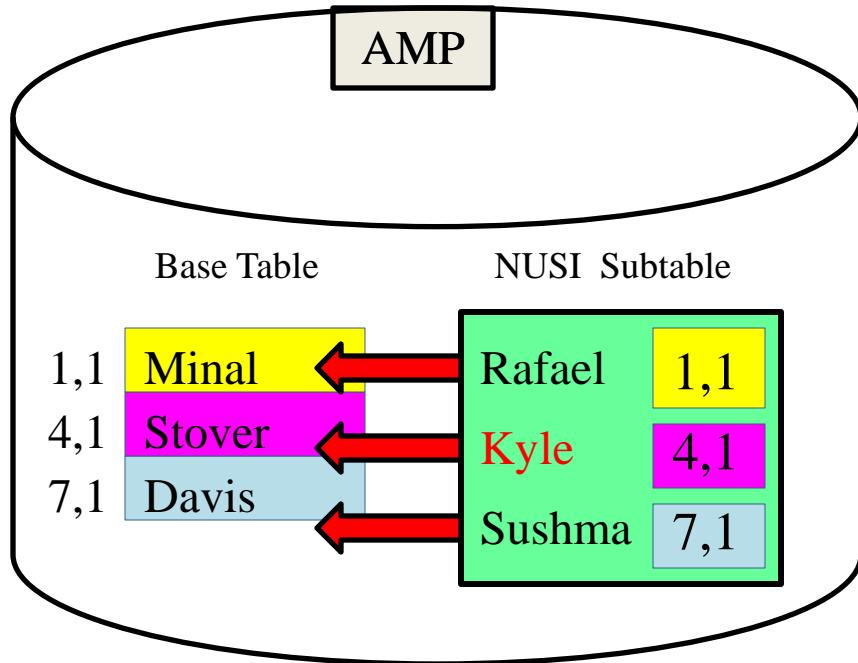
Non-Unique Secondary Index (NUSI) Subtable is AMP Local



The NUSI Subtable is AMP local because all values in the subtable are of those in the base table (on the same AMP).

Subtable rows match those of the base rows on the same AMP, which is why they call it AMP Local.

How the Parsing Engine uses the NUSI Subtable



```
SELECT * FROM Emp_Intl  
WHERE First_Name = 'Kyle';
```

Parsing Engines Plan – An ALL-AMP Operation

First_Name is a Non-Unique Secondary Index!

Have each AMP check if they have a 'Kyle' in their NUSI Subtable.

If an AMP has a 'Kyle' have them retrieve the base row (AMP Local).

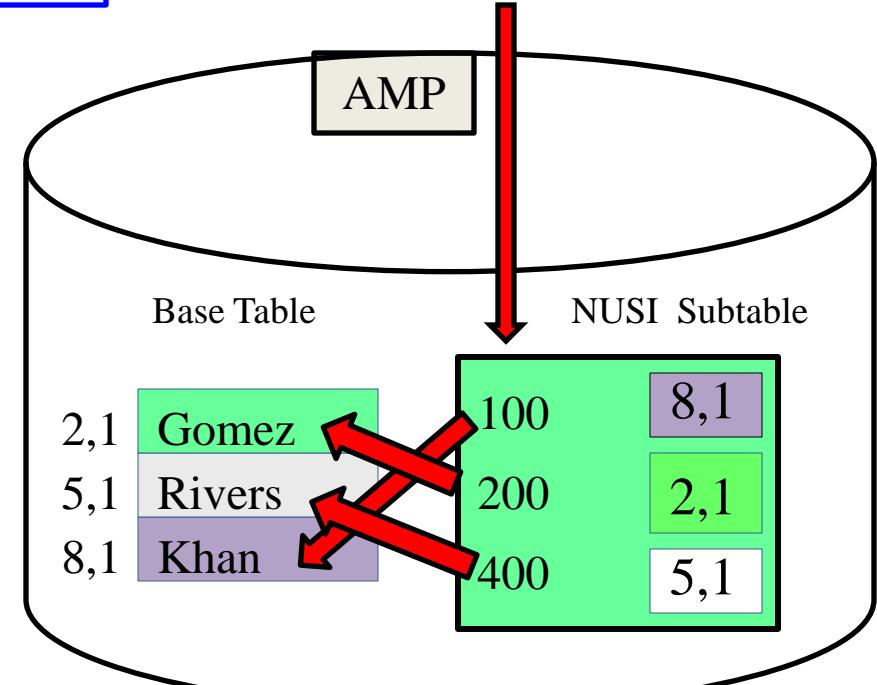
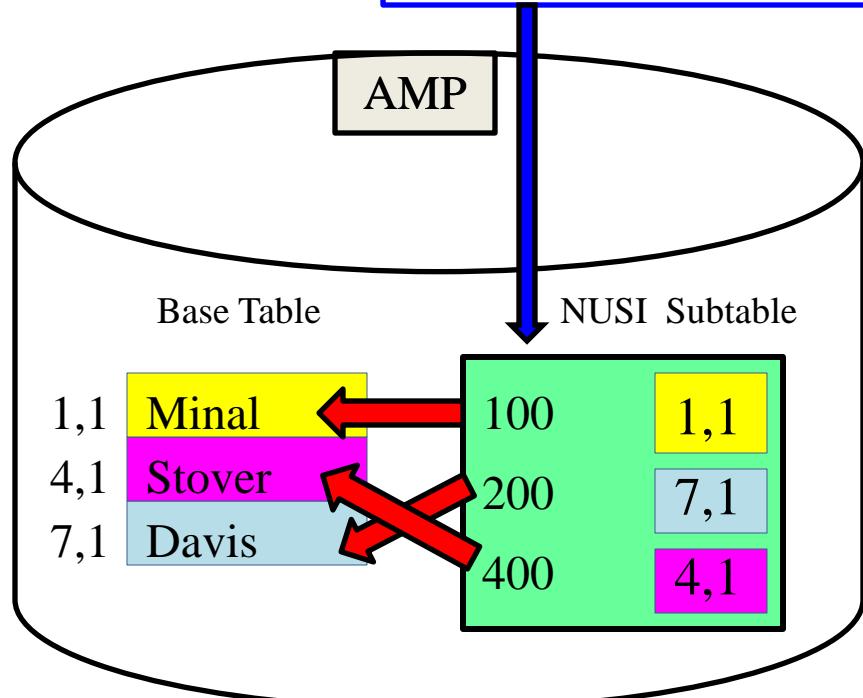
Creating a Value-Ordered NUSI

Syntax
to
Create a
Value-
Ordered
NUSI

CREATE INDEX(Dept_No) Order by values
on Emp_Intl ;

Dept_No is sorted in the subtable
by values and not hash

Dept_No is sorted in the subtable
by values and not hash



The Secondary Index Subtable will be sorted by the Values and not in Hash Order.

Watch the Videos on Secondary Indexes



Tera-Tom Trivia

Tom Coffing spent years as an actor and has done television commercials, radio shows and standup-comedy. Tom won two orchid awards as a lead actor in Cincinnati's historic Mariemont theatre. Tom spent years in Toastmasters and years performing as an actor to better his communication and teaching skills.

Click on the link(s) below or place it in your browser and watch the video on secondary indexes.

<http://www.coffingdw.com/TbasicsV12/USI.wmv>

<http://www.coffingdw.com/TbasicsV12/NUSI.wmv>

Chapter 7

Columnar Tables

“Go for the gold: better one great column and some undistinguished ones than constant mediocrity.”

- Allan Sloan

Table of Contents Chapter 7 – Columnar Tables

- [Columnar Tables have NO Primary Index](#)
- [This is NOT a NoPI Table](#)
- [NoPI Tables Spread rows across all-AMPs Evenly](#)
- [NoPI Tables used as Staging Tables for Data Loads](#)
- [NoPI Table Capabilities](#)
- [NoPI Table Restrictions](#)
- [What does a Columnar Table look like?](#)
- [Comparing Normal Table Vs Columnar Tables](#)
- [Columnar Table Fundamentals](#)
- [Example of Columnar CREATE Statement](#)
- [Columnar can move just One Container to Memory](#)
- [Containers on AMPs match up Perfectly to rebuild a Row](#)
- [Indexes can be used on Columns \(Containers\)](#)
- [Indexes can be used on Columns \(Containers\)](#)
- [Visualize a Columnar Table](#)
- [Single-Column Vs Multi-Column Containers](#)
- [Comparing Normal Table Vs Columnar Tables](#)
- [Columnar Row Hybrid CREATE Statement](#)
- [Columnar Row Hybrid Example](#)
- [AMP 1](#)
- [Review of Row Based Partition Primary Index \(PPI\)](#)
- [Visual of Row Partitioning \(PPI Tables\) by Month](#)
- [CREATE Statement for both Row and Column Partition](#)
- [Visual of Row Partitioning \(PPI Tables\)and Columnar](#)

Continued on next page

Table of Contents Chapter 7 – Columnar Tables Continued

- [How to Load into a Columnar Table](#)
- [Columnar NO AUTO COMPRESS](#)
- [Auto Compress in Columnar Tables](#)
- [Auto Compress Techniques in Columnar Tables](#)
- [When and When NOT to use Columnar Tables](#)
- [Watch the Video on the contest for the Teradata Search-off](#)

Columnar Tables have NO Primary Index

```
CREATE Table Employee_NoPI  
(  
    Emp_No          Integer  
    ,Dept_No        Integer  
    ,First_Name     Varchar(20)  
    ,Last_Name      Char(20)  
    ,Salary         Decimal (10,2)  
)  
No Primary Index ;
```

This table is called a No Primary Index ([NoPI](#)) Table!

Before discussing how to CREATE a Columnar table it is important that you understand that Teradata allows for **NOPI** tables, which distinctly state **NO Primary Index**.

This is NOT a NoPI Table

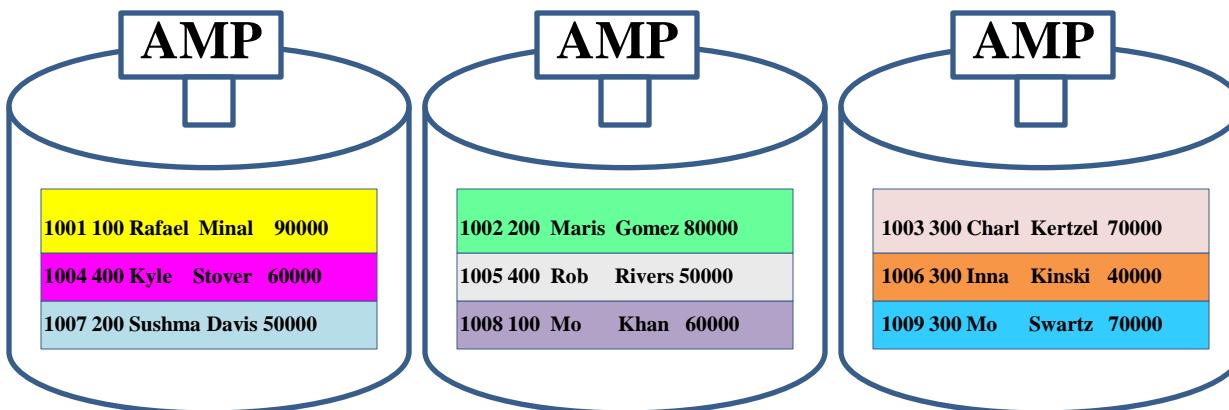
```
CREATE Table Employee_NoPI  
(  
    Emp_No          Integer  
    ,Dept_No        Integer  
    ,First_Name     Varchar(20)  
    ,Last_Name      Char(20)  
    ,Salary         Decimal (10,2)  
) ;
```

This table is NOT a No Primary Index (NoPI) Table!
This will make the first column the Primary Index
and automatically make it Non-Unique (NUPI)

If you forget to put in a Primary Index Teradata thinks you made a mistake and it will make the first column a Non-Unique Primary Index. If you want a NoPI table you must state NO PRIMARY INDEX.

NoPI Tables Spread rows across all-AMPs Evenly

Emp_No	Dept_No	First_Name	Last_Name	Salary
1001	100	Rafael	Minal	90000
1002	200	Maria	Gomez	80000
1003	300	Charl	Kertzel	70000
1004	400	Kyle	Stover	60000
1005	400	Rob	Rivers	50000
1006	300	Inna	Kinski	40000
1007	200	Sushma	Davis	50000
1008	100	Mo	Khan	60000
1009	300	Mo	Swartz	70000

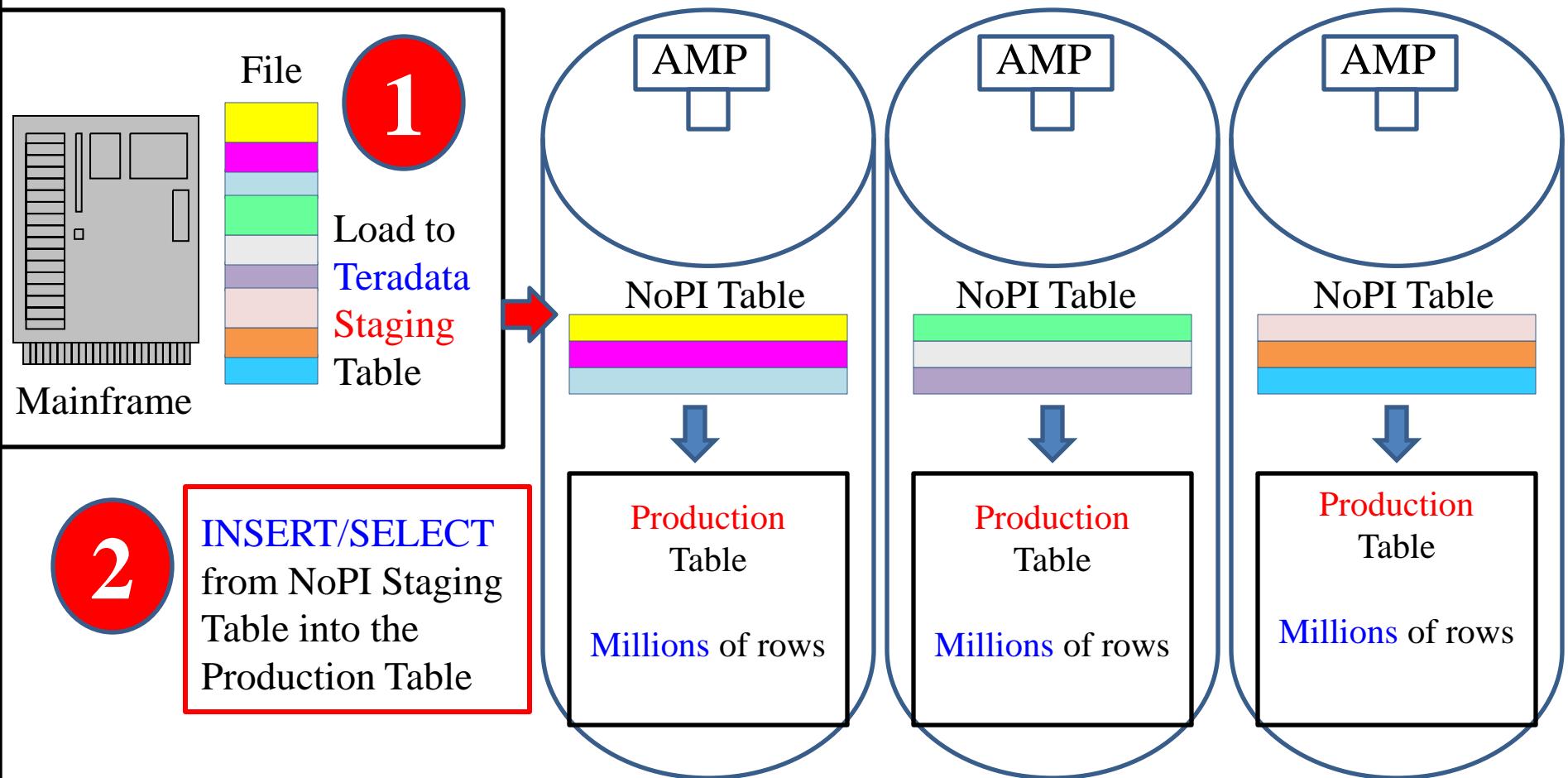


This **NoPI** Table has **NO** Primary Index.

The Rows are distributed **randomly**, but **evenly** across the AMPS.

The purpose of a NoPI table is to spread the rows evenly across the AMPs. This is why a NoPI table is often used as a staging table. The concept is to get data into Teradata with the rows spread evenly and then to use INSERT/SELECTs into the production tables. NoPI tables are also used in Columnar Tables for V14!

NoPI Tables used as Staging Tables for Data Loads



NoPI tables were first designed to be staging tables. Data from a Mainframe can be loaded onto Teradata perfectly evenly and quickly. Then an INSERT/SELECT can be done to move the data from the staging table (on Teradata) to the production table (also on Teradata). The data can be transformed in staging and there are no Load Restrictions with an INSERT/SELECT.

NoPI Table Capabilities

NoPI tables:

Are always Multi-Set Tables

Have Secondary Indexes (USI or NUSI)

Have Join Indexes

Be Volatile or Global Temporary Tables

Can COLLECT STATISTICS

Be FALBACK Protected

Have Triggers

Be Large Objects (LOBs)

Have Primary Key Foreign Key Constraints

A (**NoPI**) Table always spreads the data perfectly evenly!

Above are some of the capabilities of NoPI tables. Next we will show the restrictions.

NoPI Table Restrictions

NoPI tables are limited and below are some restrictions:

No Primary Indexes allowed

No SET Tables

No Partition Primary Index (PPI) tables

No Queue Tables

No Hash Indexes

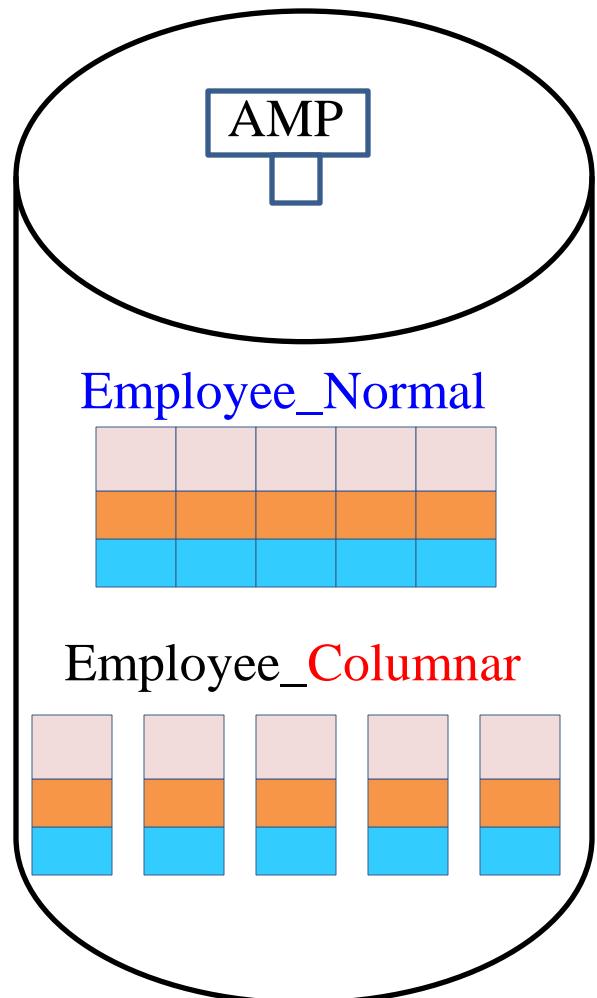
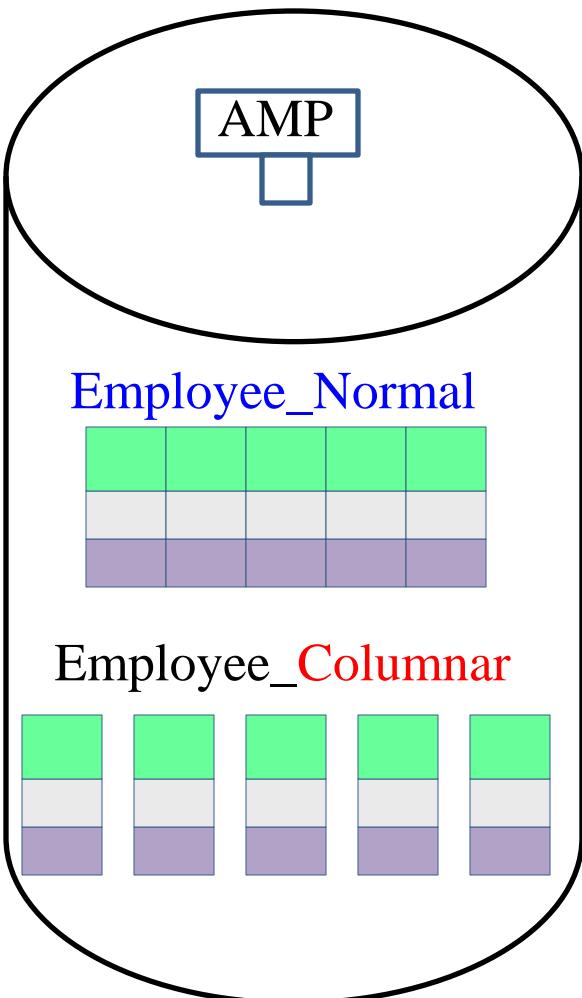
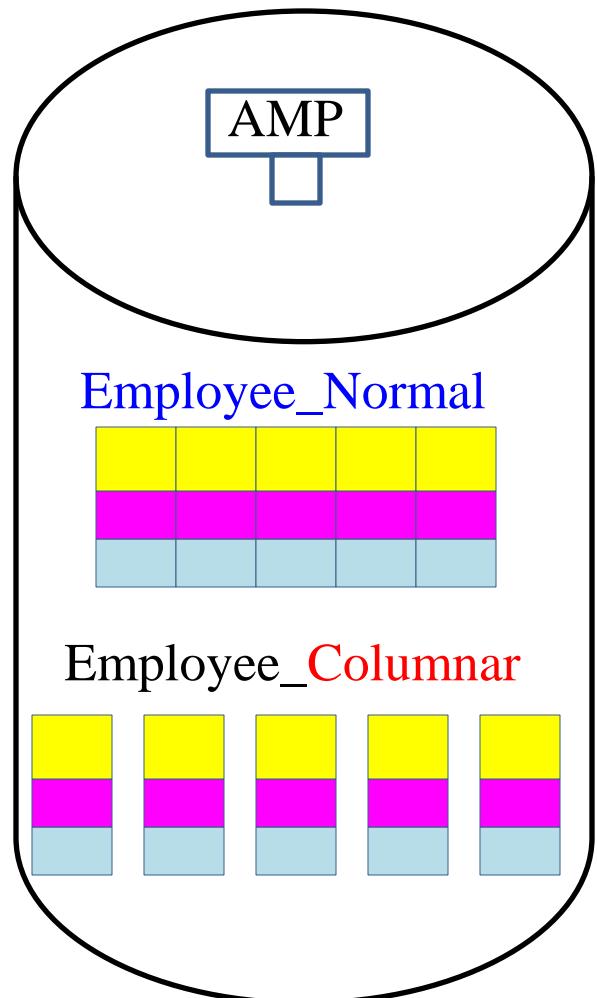
No Identity Columns

No Permanent Journaling

Can't be the Target Table for any UPDATE, UPSERT or
MERGE-INTO Statements

A (NoPI) Table is used as a **staging** table or in a **Columnar** Table!

What does a Columnar Table look like?



The two tables above contain the same Employee data, but one is a Columnar table. Employee_Normal has placed 3 rows on each AMP with 5 columns. The other table Employee_Columnar has 5 Containers each with one column.

Comparing Normal Table vs Columnar Tables

AMP

Employee_Normal

Emp_No	Dept_No	First_Name	Last_Name	Salary
1001	100	Rafael	Minal	90000.00
1004	400	Kyle	Stover	60000.00
1007	200	Sushma	Davis	50000.00

Employee_Columnar

Emp_No	Dept_No	First_Name	Last_Name	Salary
1001	100	Rafael	Minal	90000.00
1004	400	Kyle	Stover	60000.00
1007	200	Sushma	Davis	50000.00

Both tables in the example contain the same data. The first is a normal table and the second is a columnar table. Both have the same three rows, but the columnar table almost appears to be 5 different tables (each with 1 column), referred to as **containers**.

Columnar Table Fundamentals

- Columnar Tables must be a **NoPI** Table so **No Primary Index (NoPI)**.
- The NoPI brings **even distribution** to the table.
- Columnar Tables allow **Columns** to be Partitioned.
- An AMP still **holds the entire row**, but partitions vertically.
- Columns are placed inside their own individual **Container**.
- All Containers have the **same amount** of rows in the **exact order**.
- Single Columns or Multi-Columns can be placed inside containers.
- Each container looks like a small table for I/O purposes.
- **Add up all the containers** and you rebuild the **row**.
- Columnar Tables make sense when users query only certain columns.
- When a row is deleted it is **NOT Physically Deleted** but **marked deleted**.

Emp_No
1001
1004
1007

Dept_No
100
400
200

First_Name
Rafael
Kyle
Sushma

Last_Name
Minal
Stover
Davis

Salary
90000.00
60000.00
50000.00

Understand the fundamentals above and you already have a good handle on Columnar.

Example of Columnar CREATE Statement

```
CREATE Table Employee_Columnar  
(  
    Emp_No          Integer  
    ,Dept_No        Integer  
    ,First_Name     Varchar(20)  
    ,Last_Name      Char(20)  
    ,Salary         Decimal (10,2)
```

```
)
```

No Primary Index

Partition By Column ;

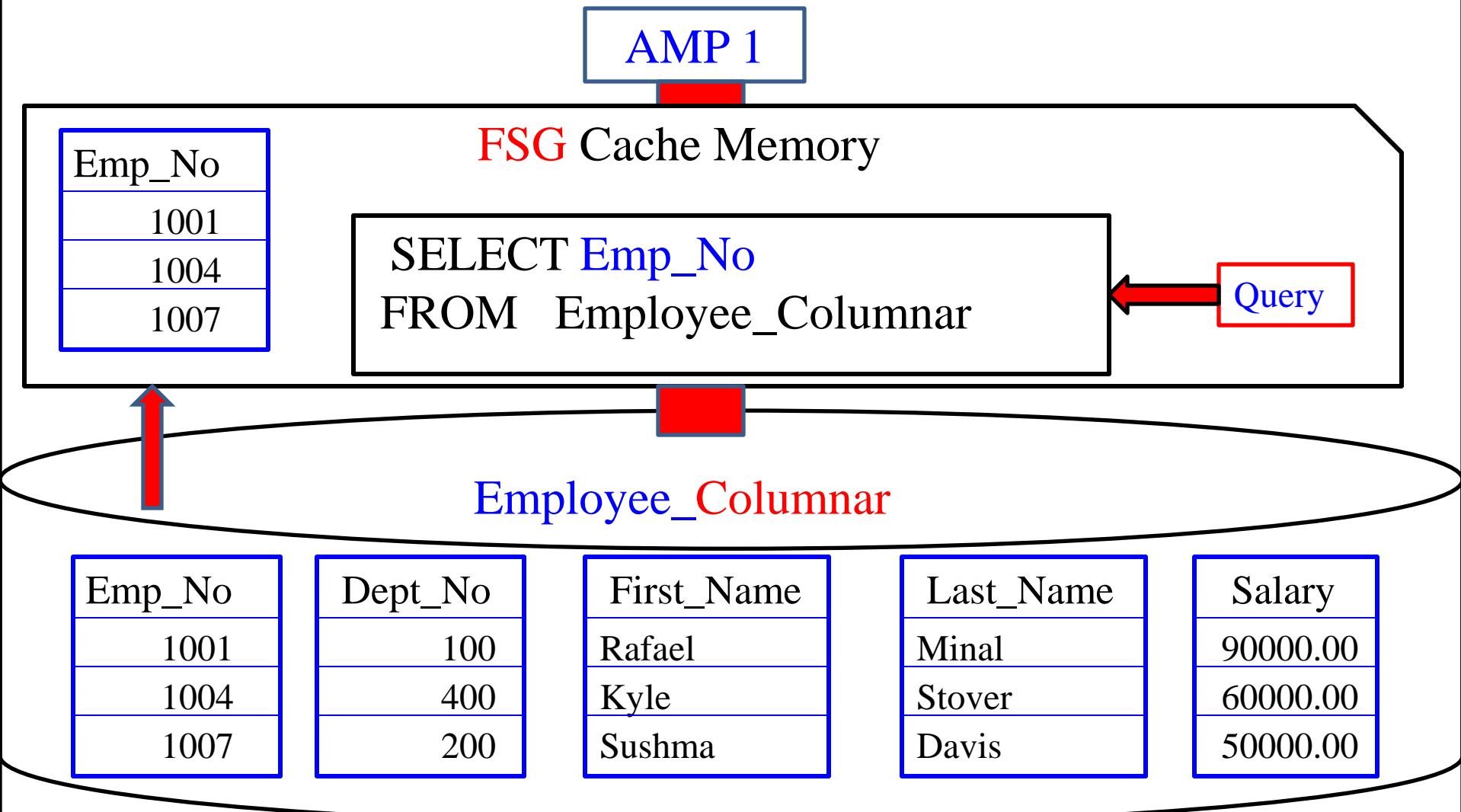


COLUMN
Keyword

Must be a
NoPI Table

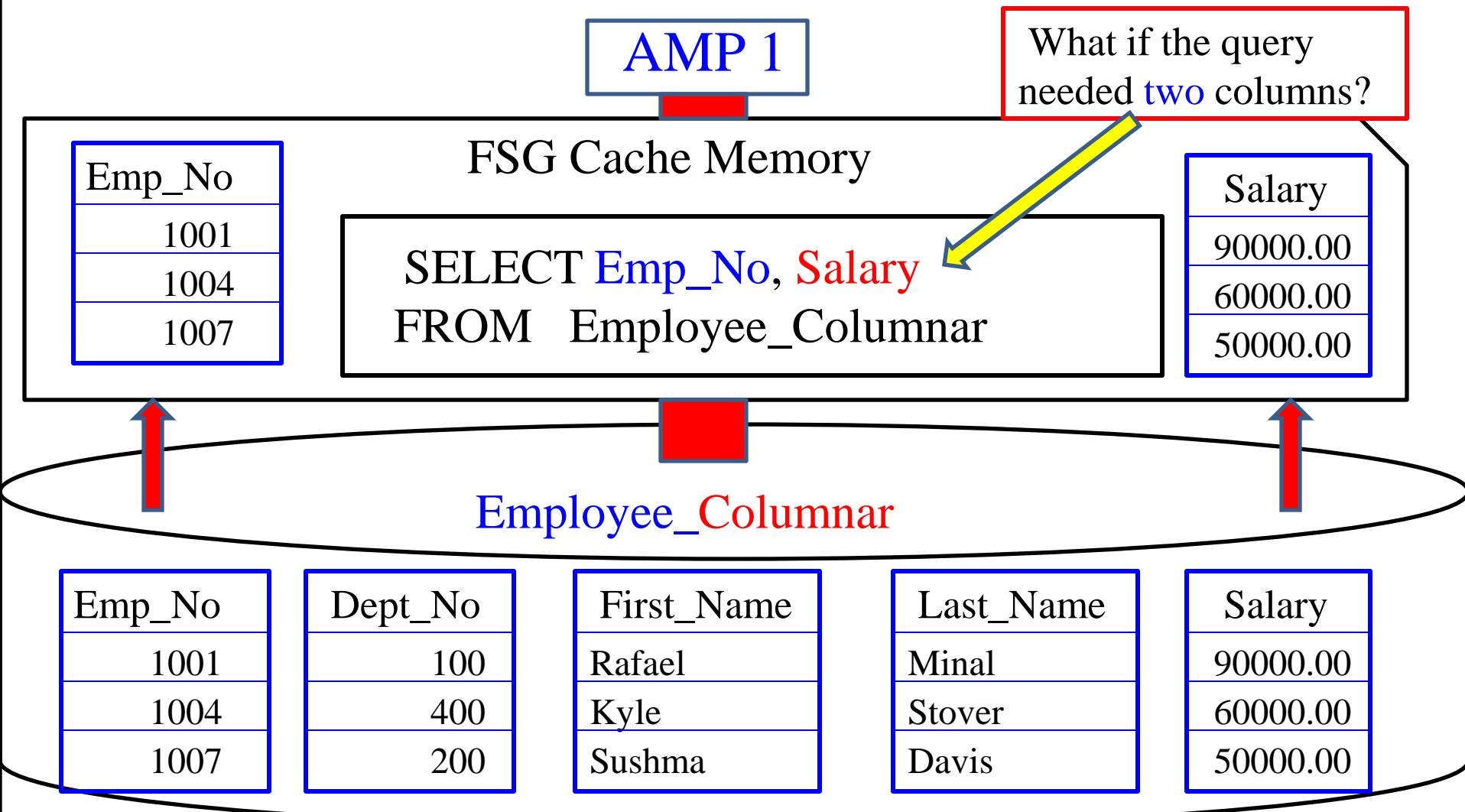
When a table is created (Teradata V14 and beyond) the creator can specify that they want the table to be a Columnar table. The table has to be a NoPI table (No Primary Index) and you must define the PARTITION BY COLUMN.

Columnar can move just One Container to Memory



The query above only asks for the column Emp_No to satisfy the query. How many columns will be placed inside this AMP's FSG Cache? **ONE!** This is because the Container for Emp_No is all that is needed. Less movement is the value of Columnar.

Containers on AMPs match up Perfectly to rebuild a Row



How many columns will be placed inside this AMP's FSG Cache? Two! This is because the Containers for Emp_No and Salary are inside their own block. Columnar tables allow for smaller blocks to move to and from memory to disk. 1007 makes 50000.00!

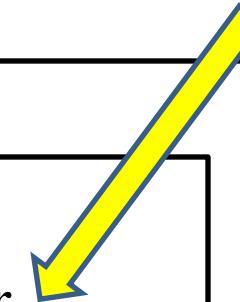
Indexes can be used on Columns (Containers)

AMP 1

What about this query?

FSG Cache Memory

```
SELECT Last_Name  
FROM Employee_Columnar  
WHERE Emp_No = 1001 ;
```



Employee_Columnar

Emp_No
1001
1004
1007

Dept_No
100
400
200

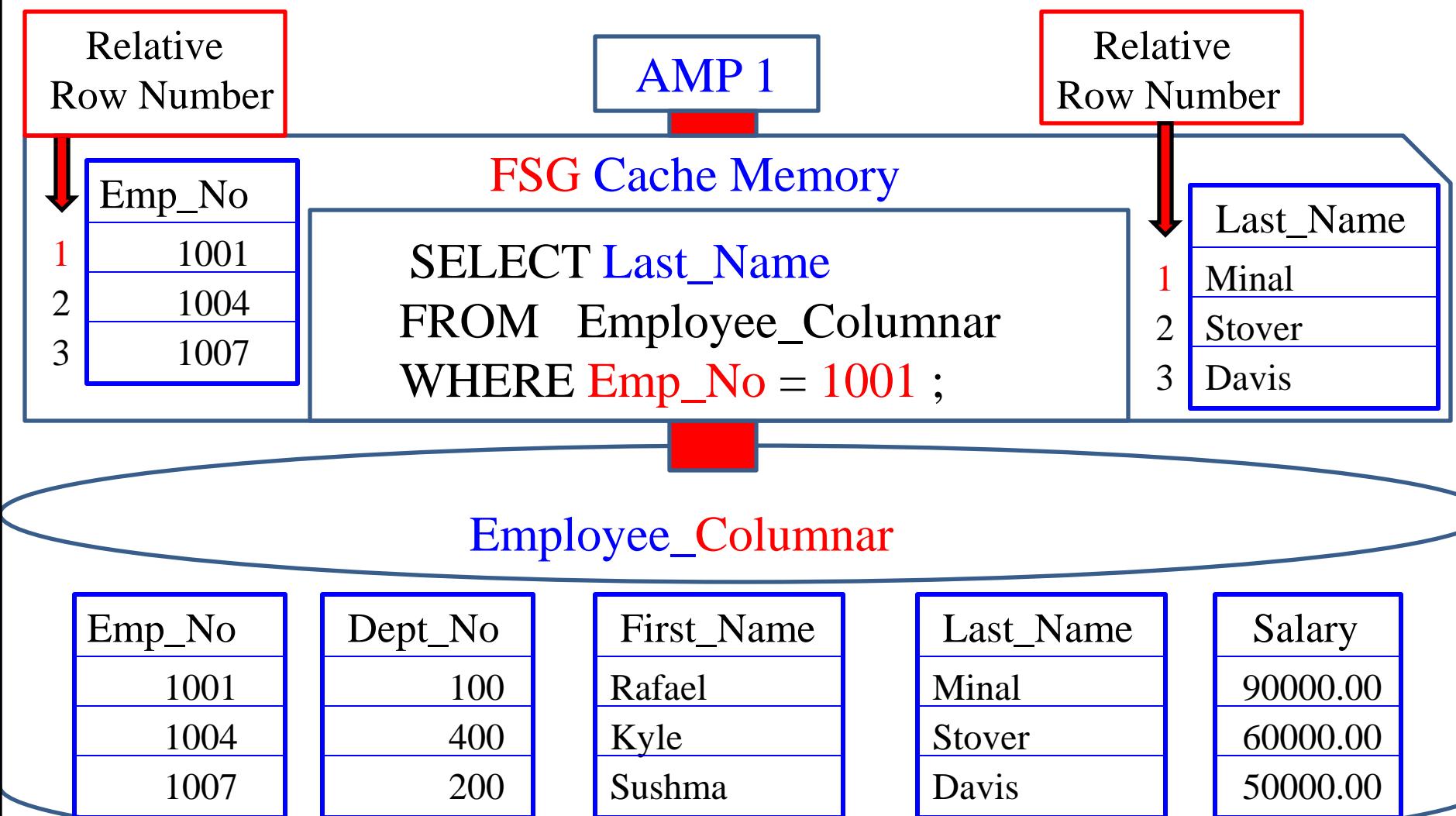
First_Name
Rafael
Kyle
Sushma

Last_Name
Minal
Stover
Davis

Salary
90000.00
60000.00
50000.00

The query above only asks for the column Last_Name in the SELECT list, but has a WHERE clause to filter for Emp_No 1001. How many **columns** will be placed inside this AMP's FSG Cache? Two interesting points are about to happen! See next slide!

Indexes can be used on Columns (Containers)



Two interesting points to note for your understanding. 1) The Emp_No container is moved into memory using an INDEX. 2) The Last_Name container is also placed into memory and Minal has the same relative row number (1) as 1001 so its found easily.

Visualize a Columnar Table

AMP 1

Emp_No		Dept_No		First_Name		Last_Name		Salary	
1	1001	1	100	1	Rafael	1	Minal	1	90000.00
2	1004	2	400	2	Kyle	2	Stover	2	60000.00
3	1007	3	200	3	Sushma	3	Davis	3	50000.00

This AMP is assigned 3 Employee Rows

All AMPS hold 3 different Employee Rows also

Each Row has 5 Columns

This Columnar Table partitions in 5 separate containers

Each container has a relative row number (1, 2, 3)

Each container has the exact same number of rows

Above are some fundamentals to visualize when thinking about Columnar Tables.

Single-Column Vs Multi-Column Containers

```
CREATE Table Employee_Columnar2  
( Emp_No          Integer  
 ,Dept_No         Integer  
 ,First_Name      Varchar(20)  
 ,Last_Name       Char(20)  
 ,Salary          Decimal (10,2))
```

No Primary Index

Partition By Column

```
( Emp_No  
 ,Dept_No  
 ,(First_Name , Last_Name, Salary)) ;
```

Single-Column
Containers

Multi-Column
Container

The syntax here is special because we have placed Emp_No into a single-column container and Dept_No into a single-column container, but we have First_Name, Last_Name and Salary all sharing a third container.

Comparing Normal Table Vs Columnar Tables

AMP

Employee_Normal

Emp_No	Dept_No	First_Name	Last_Name	Salary
1001	100	Rafael	Minal	90000.00
1004	400	Kyle	Stover	60000.00
1007	200	Sushma	Davis	50000.00

Single
Container

Single
Container

Multi
Container

Employee_Columnar2

Emp_No	Dept_No	First_Name	Last_Name	Salary
1001	100	Rafael	Minal	90000.00
1004	400	Kyle	Stover	60000.00
1007	200	Sushma	Davis	50000.00

Notice that Employee_Columnar2 has two single-column containers and one multi.

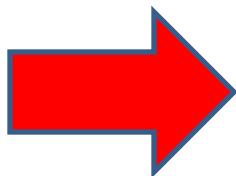
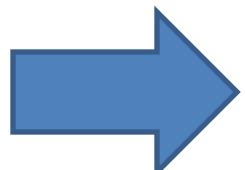
Columnar Row Hybrid CREATE Statement

```
CREATE Table Employee_Hybrid  
(    Emp_No          Integer  
     ,Dept_No        Integer  
     ,First_Name     Varchar(20)  
     ,Last_Name      Char(20)  
     ,Salary         Decimal (10,2))
```

No Primary Index

**Partition By
Column**

```
( Emp_No  No Auto Compress  
   ,Dept_No  
   ,Row (First_Name , Last_Name, Salary)  
           No Auto Compress );
```



The syntax here is special because we have combined the words **Column** with **Row**.

Columnar Row Hybrid Example

AMP

Employee_Normal

Emp_No	Dept_No	First_Name	Last_Name	Salary
1001	100	Rafael	Minal	90000
1004	400	Kyle	Stover	60000
1007	200	Sushma	Davis	50000

Column
Store
Container

Column
Store
Container

Employee_Hybrid

Row
Store

→

Emp_No
Part 1 Row 1
1001
1004
1007

Dept_No
Part 2 Row 1
100
400
200

Partition	HB	Row#	First_Name	Last_Name	Salary
0	n	1	Rafael	Minal	90000
0	n	2	Kyle	Stover	60000
0	n	3	Sushma	Davis	50000

The small arrows point out the Partition Number and starting Row Number.

AMP 1

```
SELECT First_Name,  
      Last_Name,  
      Salary  
FROM Employee_Hybrid
```

FSG Cache Memory

Row Store

First_Name	Last_Name	Salary
Rafael	Minal	90000
Kyle	Stover	60000
Sushma	Davis	50000

Emp_No	Dept_No
Part 1 Row 1	Part 2 Row 1
1001	100
1004	400
1007	200

Partition	HB	Row#	First_Name	Last_Name	Salary
0	n	1	Rafael	Minal	90000
0	n	2	Kyle	Stover	60000
0	n	3	Sushma	Davis	50000

A reason to use a Row Hybrid is mostly for compression opportunities. In this case we used it for 3 columns we expect to be used together in the SELECT List of user's SQL.

Review of Row-Based Partition Primary Index (PPI)

```
/* Creating a PPI Table */
```

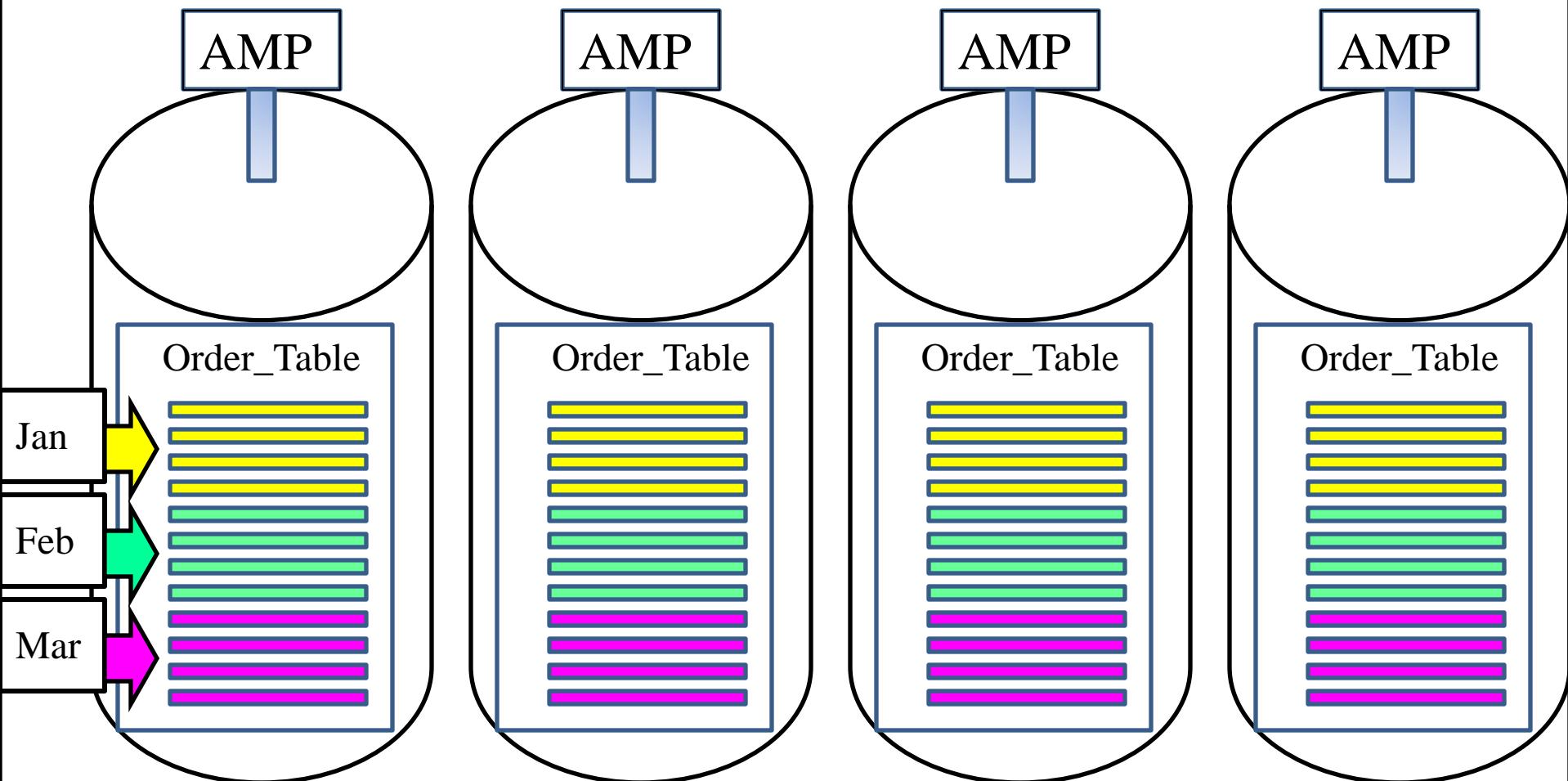
```
CREATE TABLE Order_Table_PPI
(Order_Number      INTEGER  Not Null
,Customer_Number  INTEGER
,Order_Date        DATE
,Order_Total       Decimal (10,2)
) PRIMARY INDEX(Order_Number)
PARTITION BY RANGE_N (Order_Date
    BETWEEN date '2012-01-01'
        AND date '2012-12-31'
    EACH INTERVAL '1' Month);
```

There are three different types of Row Partitioning:

- 1 Simple
- 2 RANGE_N
- 3 CASE_N

The above example is NOT Columnar, but a review of PPI tables, which partition the rows. The above is an example of a Range_N Partition. What this does is organize the AMPs rows by a date. As you can see, at the end of the CREATE Statement, we put our Interval. We've set it for '1' Month. What this means is that by the end of the year, the table will have 12 Partitions on each AMP! The next page shows a visual.

Visual of Row Partitioning (PPI Tables) by Month



The purpose of a Row Partitioned Table is to eliminate rows not needed to satisfy Range Queries. Notice that all January Orders are in the top partition (yellow) so if a user wants all orders in January each AMP reads 1 partition (top partition).

CREATE Statement for both Row and Column Partition

```
CREATE TABLE Order_Table_PPI_Col  
(Order_Number      INTEGER  Not Null  
,Customer_Number  INTEGER  
,Order_Date        DATE  
,Order_Total       Decimal (10,2)  
)  
NO PRIMARY INDEX  
PARTITION BY (Column  
, RANGE_N (Order_Date  
BETWEEN date '2012-01-01'  
AND date '2012-12-31'  
EACH INTERVAL '1' Month));
```

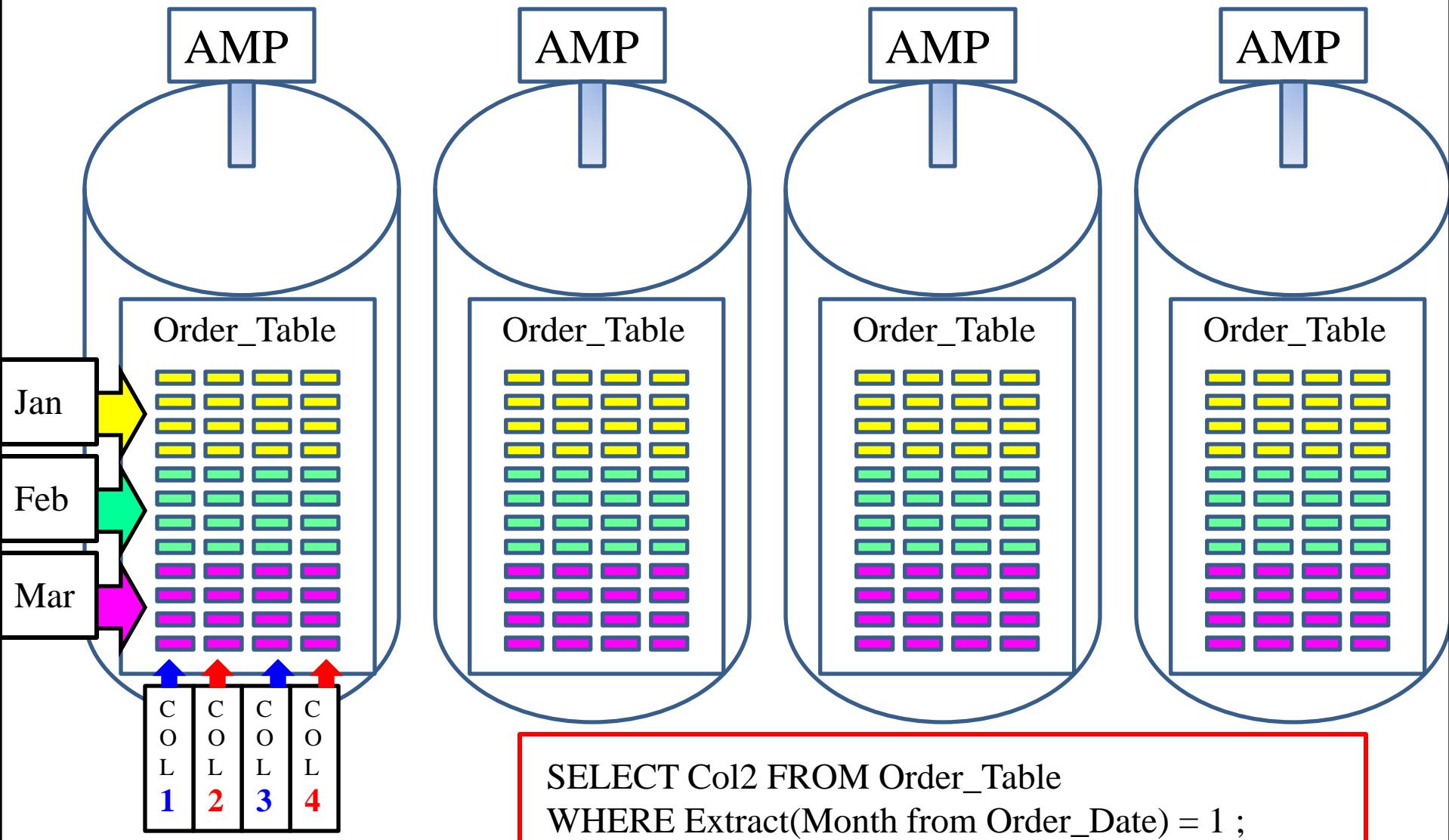
NoPI Table

RANGE_N

Columnar

This type of table will be one that has four column partitions for Order_Number, Customer_Number, Order_Date and Order_Total. Each column partition or “Container” will Row Partition by Month of Order_Date.

Visual of Row Partitioning (PPI Tables)and Columnar



This combines Row Partitioning with Column Partitioning. This is perfect for queries (see above query) that don't want to select all columns or all rows in the table.

How to Load into a Columnar Table

```
CREATE Table Emp_Staging  
( Emp_No      Integer  
,Dept_No     Integer  
,First_Name  Varchar(20)  
,Last_Name   Char(20)  
,Salary      Decimal (10,2)  
 ) No Primary Index ;
```

NoPI Staging Table

```
CREATE Table Emp_Columnar  
( Emp_No      Integer  
,Dept_No     Integer  
,First_Name  Varchar(20)  
,Last_Name   Char(20)  
,Salary      Decimal (10,2)  
 ) No Primary Index  
PARTITION BY COLUMN ;
```

Columnar Table

```
INSERT INTO Emp_Columnar  
Select * from Emp_Staging ;
```

You first load to the NoPI Staging Table on Teradata and then you do an INSERT/SELECT from the Staging Table directly into the Columnar Table. FastLoad and MultiLoad won't work on a Columnar Table.

Columnar NO AUTO COMPRESS

```
CREATE Table Employee_Columnar2
( Emp_No          Integer
,Dept_No          Integer
,First_Name       Varchar(20)
,Last_Name        Char(20)
,Salary           Decimal (10,2))
No Primary Index
Partition By Column
( Emp_No          No Auto Compress
,Dept_No          No Auto Compress
,First_Name       No Auto Compress
,Last_Name        No Auto Compress
,Salary           No Auto Compress) ;
```



Teradata compresses Columnar columns unless **NO AUTO COMPRESS** is stated.

Auto Compress in Columnar Tables

Compressed unless **NO AUTO COMPRESS** stated.

Teradata uses **many different** Compression techniques.

Teradata will decide **NOT** to Compress some Partitions.

Decompression **automatic** on column retrieval.

Compression **at its best** for **single-column** partitions.

Overhead in deciding best compression techniques.

No Overhead with **NO AUTO COMPRESS** in CREATE.

Teradata will automatically Compress each container if applicable!

Auto Compress Techniques in Columnar Tables

Trim Compression will compress leading zeros and trailing spaces to reduce space.

NULL Compression will compress NULL Values if the column is defined as **NULLABLE**.

Local Dictionary Compression is similar to Multi-Value Compression where a list of values are compressed.

Run Length Encoding Compression will have values only once and then maintain an associated count.

Unicode to UTF8 Compression for all Unicode (2-byte) characters that are ASCII so only (1-Byte) is needed.

Delta on Mean Compression will get the MEAN or AVERAGE and store -1, -2, + 3 to show the difference.

When and When NOT to use Columnar Tables

USE:

Queries access varying columns or column subsets.

The WHERE Clause is “Selective”.

Ad Hoc Queries are an excellent candidate.

Data Analytics are an excellent candidate.

Do **NOT** Use If:

Queries will run on data that is being updated/deleted.

Queries need to be tactical sub-second/OLTP queries.

A query is CPU bound.

There are some do's and don'ts you will want to know about.

Watch the Video on the contest for the Teradata Search-off



Tera-Tom Trivia

Tom Coffing is a professional golf coach and caddy. Tom has caddied in over 100 professional events for his daughter Carling Coffing. Carling won the Golf Channel reality TV show “The Big Break” and she has made the cut in all three LPGA events. Carling and Tom continue to work together today. Carling continues to have enormous success as a TV personality.

Click on the link below or place it in your browser and watch the video on the famous contest the Teradata Search-off.

<http://www.coffingdw.com/TbasicsV12/searchoff.wmv>

Chapter 8

Temporal Tables

Create Functions

“One thing you can’t recycle is wasted time.”

- Anonymous

Table of Contents Chapter 8 – Temporal Tables Create Functions

- [Three types of Temporal Tables](#)
- [CREATING a Bi-Temporal Table](#)
- [PERIOD Data Types](#)
- [Bi-Temporal Data Type Standards](#)
- [Bi-Temporal Example – Tera-Tom buys!](#)
- [A Look at the Temporal Results](#)
- [Bi-Temporal Example – Tera-Tom Sells!](#)
- [Bi-Temporal Example – How the data looks!](#)
- [Normal SQL for Bi-Temporal Tables](#)
- [NONSEQUENCED SQL for Temporal Tables](#)
- [AS OF SQL for Temporal Tables](#)
- [NONSEQUENCED for Both](#)
- [Bi-Temporal Example – Socrates is DELETED!](#)
- [Property Owners Before DELETE on April 1st](#)

Three types of Temporal Tables

The Three types of Temporal Tables are:

- 1 Valid Time Temporal Tables
- 2 Transaction Time Temporal Tables
- 3 Bi-Temporal Tables containing both Valid Time and Transaction Time.

Temporal Tables use a Valid Time or Transaction Time or combine both Valid Time and Transaction Time to form Bi-Temporal tables.

CREATING a Bi-Temporal Table

```
CREATE MULTISET TABLE Property_Owners  
( Cust_No          INTEGER  
 ,Prop_No          INTEGER  
 ,Prop_Val_Time   PERIOD (DATE) NOT NULL as VALIDTIME  
 ,Prop_Tran_Time  PERIOD (TIMESTAMP(6) with TIME ZONE)  
                  NOT NULL as TRANSACTIONTIME  
 ) PRIMARY INDEX(Prop_No) ;
```

This is a Bi-Temporal Table because one column is aliased **VALIDTIME** and another column is aliased **TRANSACTIONTIME**. This makes this table a Bi-Temporal Table.

PERIOD Data Types

```
CREATE MULTISET TABLE Property_Owners  
( Cust_No          INTEGER  
,Prop_No           INTEGER  
,Prop_Val_Time    PERIOD (DATE) NOT NULL as VALIDTIME  
,Prop_Tran_Time   PERIOD (TIMESTAMP(6) with TIME ZONE)  
                           NOT NULL as TRANSACTIONTIME  
) PRIMARY INDEX(Prop_No);
```

A Period Data Type means a beginning and ending date or Timestamp:



2011-01-01 , 99999-12-31

Or

2011-01-01 , 2012-06-30

Or

2011-01-01 08:09:290000-05:00, 9999-12-31 23:59:59.999999+00:00

A new data type PERIOD has been introduced. This means two dates (a begin and end date) or it could be two Timestamps (a begin and ending Timestamp).

Bi-Temporal Data Type Standards

```
CREATE MULTISET TABLE Property_Owners
(
    Cust_No          INTEGER
    ,Prop_No         INTEGER
    ,Prop_Val_Time   PERIOD (DATE) NOT NULL as VALIDTIME
    ,Prop_Tran_Time  PERIOD (TIMESTAMP(6) with TIME ZONE)
                      NOT NULL as TRANSACTIONTIME
)
PRIMARY INDEX(Prop_No) ;
```

What PERIOD Data Types do ValidTime and TransactionTime require?

- **ValidTime** can be either a **date** or a **Timestamp**
- **TransactionTime** must be a **Timestamp** written **exactly** as above!

The example above is perfect for your PERIOD Data type for TRANSACTIONTIME. You have options for the VALIDTIME, as it can be either a Date or Timestamp.

Bi-Temporal Example – Tera-Tom buys!

```
CREATE MULTISET TABLE Property_Owners  
(  
    Cust_No          INTEGER  
    ,Prop_No         INTEGER  
    ,Prop_Val_Time   PERIOD (DATE) NOT NULL as VALIDTIME  
    ,Prop_Tran_Time  PERIOD (TIMESTAMP(6) with TIME ZONE)  
                      NOT NULL as TRANSACTIONTIME  
)  
PRIMARY INDEX(Prop_No) ;
```

```
INSERT INTO PROPERTY_OWNERS  
        (Cust_No, Prop_No)  
VALUES (1, 100) ;
```

On January 1, 2011 Tera-Tom buys property 100 which is beach front property. Tera-Tom is Cust_No 1 in your table and number 1 in your heart.

A Look at the Temporal Results

```
INSERT INTO PROPERTY OWNERS  
  (Cust_No, Prop_No)  
VALUES (1, 100) ;
```

Below is what the table looks like internally

TransactionTime should
be displayed as Timestamp
,but not enough room here.

Property_Owners

Cust_No	Prop_No	ValidTime	TransactionTime
1	100	2011-01-01, 9999-12-31	2011-01-01 , 9999-12-31



On January 1, 2011 Tera-Tom buys property 100 and this is what the Bi-Temporal table looks like. Notice the 9999-12-31 dates. That means this is an OPEN Date.

Bi-Temporal Example – Tera-Tom Sells!

```
UPDATE Property_Owners  
SET Cust_No = 2  
WHERE Prop_No = 100 ;
```

How will the table below change after the UPDATE?

TransactionTime should be displayed as Timestamp, but not enough room here.

Property_Owners			
Cust_No	Prop_No	ValidTime	TransactionTime
1	100	2011-01-01, 9999-12-31	2011-01-01 , 9999-12-31

On January 1, 2011 Tera-Tom buys property 100 and then Tera-Tom sells to Socrates (Cust_No 2) on February 14th, 2011.

Bi-Temporal Example – How the data looks!

Property_Owners **Before** Update

Cust_No	Prop_No	ValidTime	TransactionTime
1	100	2011-01-01, 9999-12-31	2011-01-01 , 9999-12-31

Property_Owners **After** Update

Cust_No	Prop_No	ValidTime	TransactionTime
1	100	2011-01-01, 9999-12-31	2011-01-01 , 2011-02-14
1	100	2011-01-01, 2011-02-14	2011-02-14 , 9999-12-31
2	100	2011-02-14, 9999-12-31	2011-02-14 , 9999-12-31

Here is how the new table looks like with three rows. In the bottom table example there is only 1-row that is still open. Do you know which one? The last one!

Normal SQL for Bi-Temporal Tables

Property_Owners Table

Cust_No	Prop_No	ValidTime	TransactionTime
1	100	2011-01-01, 9999-12-31	2011-01-01 , 2011-02-14
1	100	2011-01-01, 2011-02-14	2011-02-14 , 9999-12-31
2	100	2011-02-14, 9999-12-31	2011-02-14 , 9999-12-31

SELECT * FROM Property_Owners ;

Normal
SQL

Cust_No Prop_No
2 100

Shows only
open rows

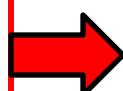
It is special SQL that allows Bi-Temporal tables to work so effectively. You will see a wide variety of SQL Keywords before the real SQL starts. The first is normal SQL.

NONSEQUENCED SQL for Temporal Tables

Property_Owners Table

Cust_No	Prop_No	ValidTime	TransactionTime
1	100	2011-01-01, 9999-12-31	2011-01-01 , 2011-02-14
1	100	2011-01-01, 2011-02-14	2011-02-14 , 9999-12-31
2	100	2011-02-14, 9999-12-31	2011-02-14 , 9999-12-31

Nonsequenced
Keyword



NONSEQUENCED VALIDTIME

SELECT * FROM Property_Owners ;

<u>Cust No</u>	<u>Prop No</u>	<u>Prop Val Time</u>
1	100	'2011-01-01 , 2011-02-14
2	100	'2011-02-14 , 9999-12-31

Shows all
customers

It is special SQL that allows Bi-Temporal tables to work so effectively. Here is a look at the keyword NONSEQUENCED. This brings back all customers.

AS OF SQL for Temporal Tables

Property_Owners Table

Cust_No	Prop_No	ValidTime	TransactionTime
1	100	2011-01-01, 9999-12-31	2011-01-01 , 2011-02-14
1	100	2011-01-01, 2011-02-14	2011-02-14 , 9999-12-31
2	100	2011-02-14, 9999-12-31	2011-02-14 , 9999-12-31

VALIDTIME AS OF DATE ‘2011-01-30’

SELECT * FROM Property_Owners ;

Cust_No	Prop_No
1	100

AS OF SQL

Point of view
as of Date.

It is special SQL that allows Bi-Temporal tables to work so effectively. The AS OF DATE ‘2011-01-30’ reports the state of Property_Owners on that exact date.

NONSEQUENCED for Both

Cust_No	Prop_No	ValidTime	TransactionTime
1	100	2011-01-01, 9999-12-31	2011-01-01 , 2011-02-14
1	100	2011-01-01, 2011-02-14	2011-02-14 , 9999-12-31
2	100	2011-02-14, 9999-12-31	2011-02-14 , 9999-12-31

**NONSEQUENCED VALIDTIME
AND NONSEQUENCED TRANSACTIONTIME**

```
SELECT * FROM Property_Owners ;
```

Cust_No	Prop_No	Prop_Val_Time	Prop_Tran_Time
1	100	'2011-01-01 , 2011-02-14	'2011-01-01 , 2011-02-14
1	100	'2011-01-01 , 2011-02-14	'2011-02-14 , 9999-12-31
2	100	'2011-02-14 , 9999-12-31	'2011-02-14 , 9999-12-31

It is special SQL that allows Bi-Temporal tables to work so effectively. Above is NONSEQUENCED VALIDTIME and NONSEQUENCED TRANSACTIONTIME.

Bi-Temporal Example – Socrates is DELETED!

```
DELETE FROM Property_Owners  
WHERE Prop_No = 100 ;
```

How will the table change below after the DELETE?

Property_Owners **Before DELETE**

Cust_No	Prop_No	ValidTime	TransactionTime
1	100	2011-01-01, 9999-12-31	2011-01-01 , 2011-02-14
1	100	2011-01-01, 2011-02-14	2011-02-14 , 9999-12-31
2	100	2011-02-14, 9999-12-31	2011-02-14 , 9999-12-31

On April Fools day, April 1, 2011 Socrates sells the property, but through another Mortgage company, so since the mortgage company no longer owns the property, Socrates is DELETED. How will the table look after the Delete.

Property_Owners Before DELETE on April 1st

Cust_No	Prop_No	ValidTime	TransactionTime
1	100	2011-01-01, 9999-12-31	2011-01-01 , 2011-02-14
1	100	2011-01-01, 2011-02-14	2011-02-14 , 9999-12-31
2	100	2011-02-14, 9999-12-31	2011-02-14 , 9999-12-31

Property_Owners AFTER DELETE on April 1st

Cust_No	Prop_No	ValidTime	TransactionTime
1	100	2011-01-01, 9999-12-31	2011-01-01 , 2011-02-14
1	100	2011-01-01, 2011-02-14	2011-02-14 , 9999-12-31
2	100	2011-02-14, 9999-12-31	2011-02-14 , 2011-04-01
2	100	2011-02-14, 2011-04-01	2011-04-01 , 9999-12-31

Here is the table and it has no Open Rows. The bold red shows why the row is closed.

Chapter 9

How Joins Work

Internally

“A Join Index walked up to two tables in a bar and said,
mind if I Join you?”

Tera-Tom Coffing

Table of Contents Chapter 9 – How Joins Work Internally

- [Teradata Join Quiz](#)
- [Teradata Join Quiz Answer](#)
- [If the Join Condition is the Primary Index no Movement](#)
- [How the Parsing Engine Decides on a Join Plan](#)
- [Quiz – Redistribute the Employees by their Dept_No](#)
- [Quiz – Employees Dept_No landed on AMP with Matches](#)
- [When Rows are on the same AMP they can be Joined](#)
- [Redistribution and then a Row Hash Match Scan](#)
- [Quiz – Redistribute the Orders to the Proper AMP](#)
- [Answer to Redistribute the Employees by their Dept_No Quiz](#)
- [A Visual of the Join in Action](#)
- [Nexus Query Chameleon Example](#)
- [The Big Table/Small Table Join causes Duplication](#)
- [Visual of Duplication of the Smaller Table across All-AMPs](#)
- [Duplication of the Smaller Table across All-AMPs](#)
- [A Visual of Duplication of the Smaller Table on a Single AMP](#)
- [A Visual of Redistribution on a Single AMP](#)

Teradata Join Quiz

Which Statement is NOT true!

1. Each Table in Teradata has a Primary Index, unless it is a NoPI table.
2. The Primary Index is the mechanism that allows Teradata to physically **distribute** the rows of a table across the AMPs.
3. Each AMP Sorts its rows by the Row-ID, unless it is a Partitioned table, and then the sort is first by the Partition and then by Row-ID.
4. For two rows to be Joined together Teradata insists that both rows are physically on the same AMP.
5. Teradata will either **Redistribute** one or both of the tables or **Duplicate** the smaller table across all AMPs to ensure matching rows are on the same AMP, even if it is only for the life of the Join.

Do you know which statement above is False?

Teradata Join Quiz Answer

Which Statement is NOT true!

1. Each Table in Teradata has a Primary Index, unless it is a NoPI table.
2. The Primary Index is the mechanism that allows Teradata to physically **distribute** the rows of a table across the AMPs.
3. Each AMP Sorts their rows by the Row-ID, unless it is a Partitioned table, and then the sort is first by the Partition and then by Row-ID.
4. For **two rows** to be Joined together Teradata insists that **both rows** are **physically** on the same AMP.  **TRUE**
5. Teradata will either **Redistribute** one or both of the tables or **Duplicate** the smaller table across all AMPs to ensure matching rows are on the same AMP, even if it is only for the life of the Join.

Do you know which statement above is False? All statements above are **TRUE!**

If the Join Condition is the Primary Index no Movement

```
SELECT Last_Name, Department_Name  
FROM Employee_Table as E  
INNER JOIN Department_Table as D  
ON E.Dept_No = D.Dept_No ;
```

Is Dept_No the
Primary Index of
Employee_Table?

NO

Is Dept_No the
Primary Index of
Department_Table?

YES

```
CREATE Table Employee_Table  
( Employee_No      INTEGER  
,Dept_No          INTEGER  
,Last_Name        CHAR(20)  
,First_Name       VARCHAR(20)  
,Salary           DECIMAL(10,2)  
) UNIQUE PRIMARY INDEX(Employee_No);
```

```
CREATE Table Department_Table  
(Dept_No          INTEGER  
,Department_Name  CHAR(20)  
,Mgr_No           INTEGER  
,Budget           DECIMAL(10,2)  
)  
UNIQUE PRIMARY INDEX(Dept_No);
```

Teradata knows that it can only JOIN two rows together if they are physically on the same AMP. This can occur naturally if the join condition columns are the Primary Indexes of their respective tables, but most likely Teradata will have to move data to get the matching rows on the same AMP. What will the Parsing Engine decide to do next?

How the Parsing Engine Decides on a Join Plan

```
SELECT Last_Name, E.Dept_No, Department_Name  
FROM Employee_Table as E,  
     Department_Table as D  
WHERE E.Dept_No = D.Dept_No Order BY 1 ;
```

Is Dept_No the Primary Index of Employee_Table?
NO

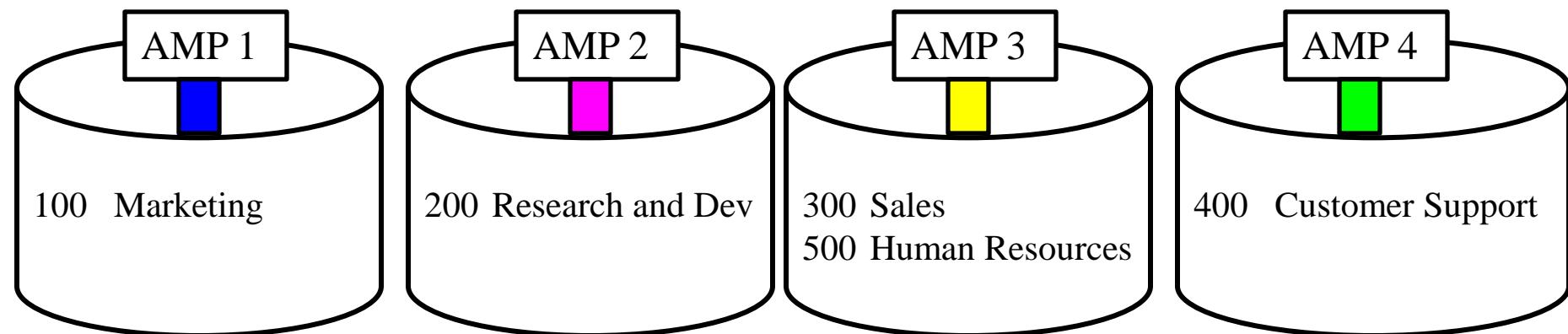
Is Dept_No the Primary Index of Department_Table?
YES

Redistribute the Employee_Table by Dept_No into Spool for this join.

The Parsing Engine (PE) knows that the Dept_No column is the Primary Index for the Department_Table. It also knows that the Dept_No column is NOT the Primary Index for the Employee_Table, so the PE commands the AMPs to Redistribute the entire Employee_Table by Dept_No into spool. This is equivalent to loading the Employee_Table with a Primary Index of Dept_No. Now all matching rows can join.

Quiz – Redistribute the Employees by their Dept_No

Employee_Table					Department_Table	
Employee_No	Dept_No	Last_Name	First_Name	Salary	Dept_No	Department_Name
1232578	100	Chambers	Mandee	48850.00	100	Marketing
1256349	400	Harrison	Herbert	54500.00	200	Research and Dev
2341218	400	Reilly	William	36000.00	300	Sales
2312225	300	Larkins	Lorraine	40200.00	400	Customer Support
2000000	?	Jones	Squiggy	32800.50	500	Human Resources
1000234	10	Smythe	Richard	32800.00		
1121334	400	Strickling	Cletus	54500.00		
1324657	200	Coffing	Billy	41888.88		
1333454	200	Smith	John	48000.00		

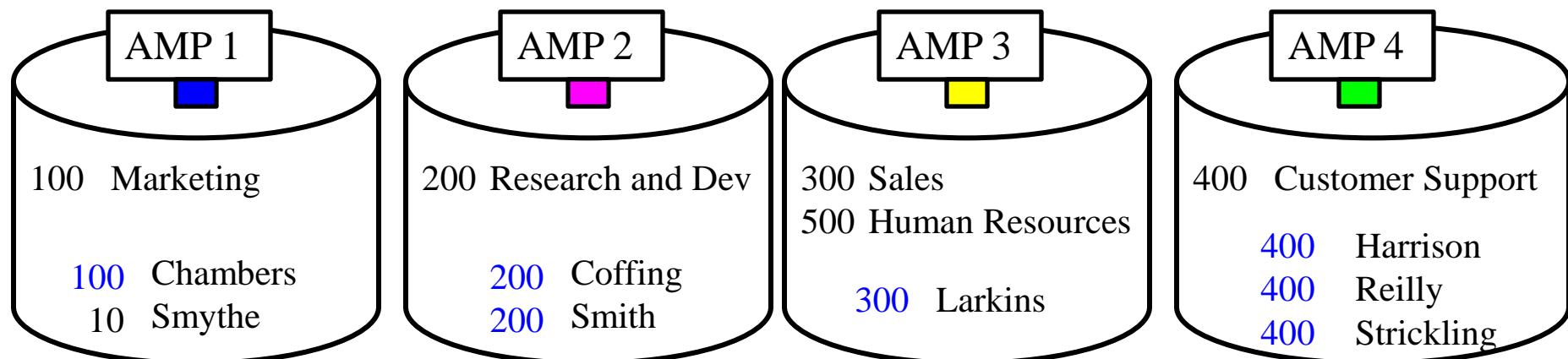


If Teradata decides to Redistribute the Employee_Table by Dept_No which AMPs will hold which Employees? Place their Dept_No and Last_Name on the AMP after Redistribution.

Fill in the quiz above. This is a great opportunity to understand the Teradata engine.

Quiz – Employees' Dept_No landed on AMP with Matches

Employee_Table					Department_Table	
Employee_No	Dept_No	Last_Name	First_Name	Salary	Dept_No	Department_Name
1232578	100	Chambers	Mandee	48850.00	100	Marketing
1256349	400	Harrison	Herbert	54500.00	200	Research and Dev
2341218	400	Reilly	William	36000.00	300	Sales
2312225	300	Larkins	Lorraine	40200.00	400	Customer Support
2000000	?	Jones	Squiggy	32800.50	500	Human Resources
1000234	10	Smythe	Richard	32800.00		
1121334	400	Strickling	Cletus	54500.00		
1324657	200	Coffing	Billy	41888.88		
1333454	200	Smith	John	48000.00		

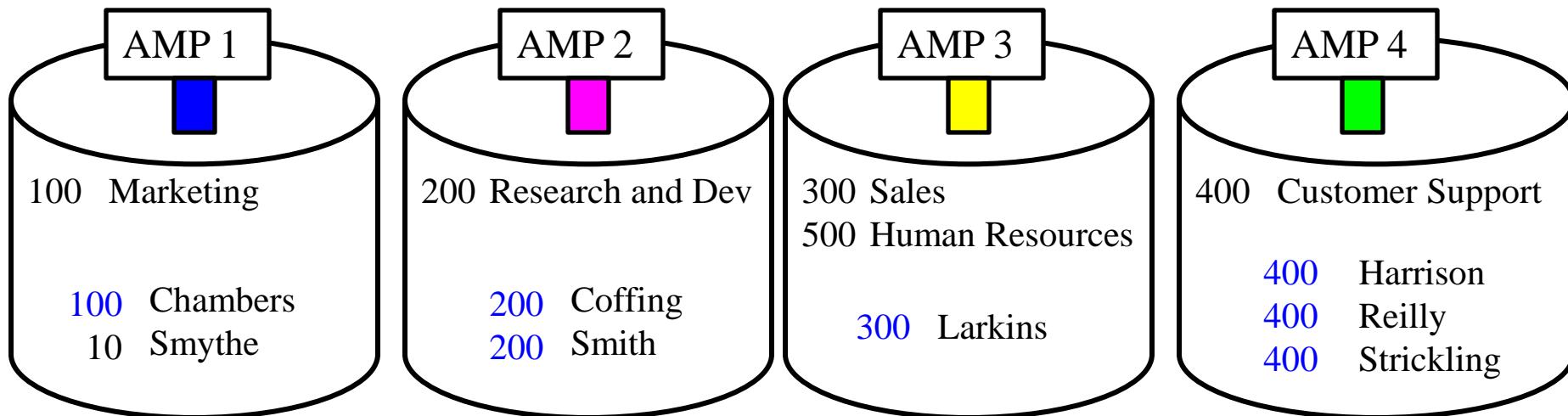


Each redistributed row landed on the same AMP as its matching row. Notice that Squiggy Jones has a NULL department so Teradata will not redistribute that row on an Inner Join. Smythe in Dept_No 10 hashes to AMP 1 but has no match. Turn the page.

When Rows are on the same AMP they can be Joined

Query

```
SELECT Last_Name, E.Dept_No, Department_Name  
FROM Employee_Table as E, Department_Table as D  
WHERE E.Dept_No = D.Dept_No ORDER BY 2, 1 ;
```



Report

Last_Name	Dept_No	Department_Name
Chambers	100	Marketing
Coffing	200	Research and Dev
Smith	200	Research and Dev
Larkins	300	Sales
Harrison	400	Customer Support
Reilly	400	Customer Support
Strickling	400	Customer Support

Redistribution and then a Row Hash Match Scan

Query

```
SELECT Customer_Name, Order_Total  
FROM Customer_Table as C  
INNER JOIN  
    Order_Table as O  
ON      C.Customer_Number = O.Customer_Number ;
```

Parsing
Engine
Analysis

Is the Primary Index for
the Customer_Table
Customer_Number?

YES

Is the Primary Index for
the Order_Table
Customer_Number?

NO

Parsing
Engine
Analysis

Parsing
Engine
PLAN

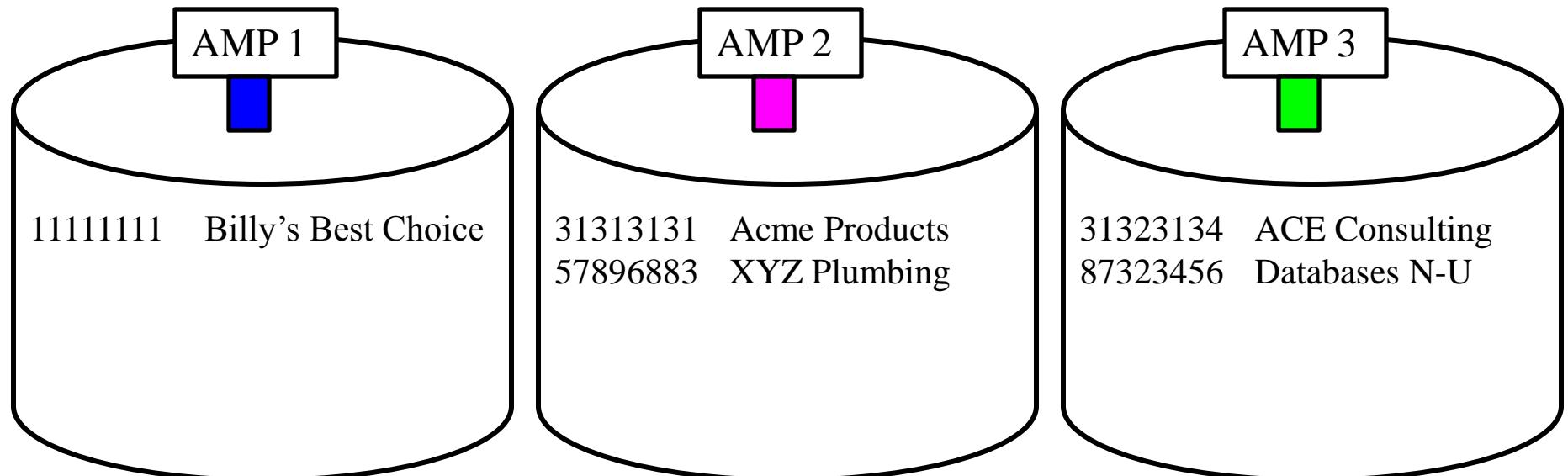
Retrieve the Customer_Number and the Order_Total from the Order_Table via a
Full Table Scan and then REDISTRIBUTE the rows by Customer_Number!

Now that the Order_Table columns needed have been Redistributed to the proper
AMP we can perform a ROW HASH MATCH SCAN.

Above resides the Query, the Parsing Engine's (PE) Analysis and a summary (in laymen's terms) of the PE's PLAN. First we need to get all matching rows to the corresponding AMP so the join can be performed between the two tables.

Quiz – Redistribute the Orders to the Proper AMP

Customer_Table		Order_Table		
Customer_Number	Customer_Name	Order_Number	Customer_Number	Order_Total
11111111	Billy's Best Choice	123456	11111111	12347.53
31313131	Acme Products	123512	11111111	8005.91
31323134	ACE Consulting	123552	31323134	5111.47
57896883	XYZ Plumbing	123585	87323456	15231.62
87323456	Databases N-U	123777	57896883	23454.84

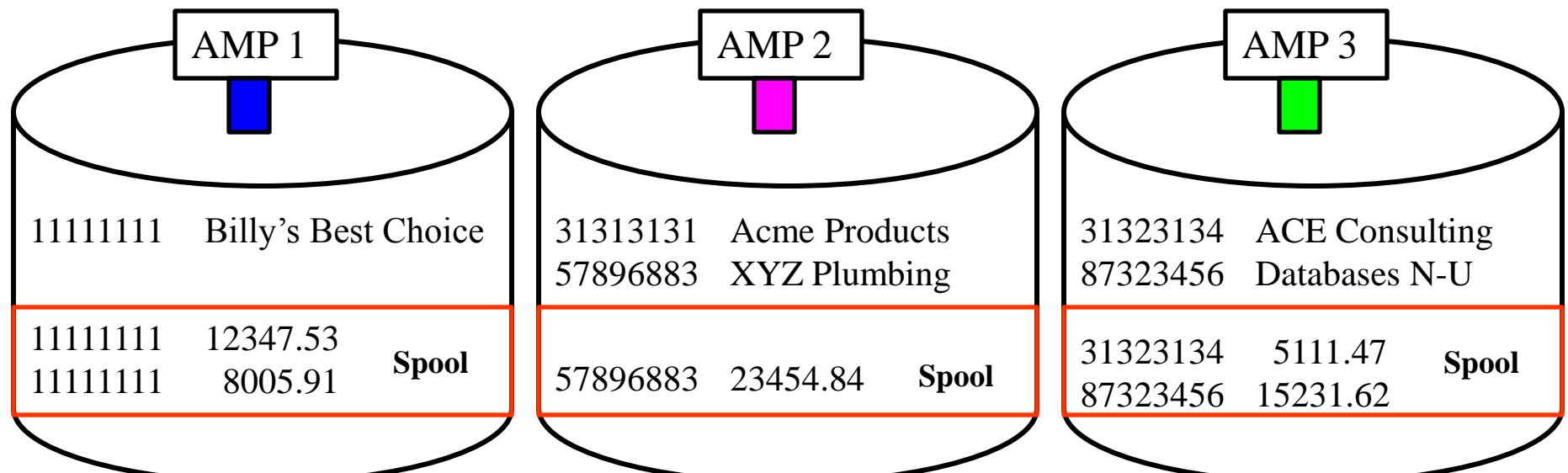


If Teradata decides to Redistribute the **Order_Table** by **Customer_No** which AMPs will hold which Orders? Place their Customer_Number and Order_Total on the AMP after Redistribution.

Fill in the quiz above. This is a great opportunity to understand the Teradata engine.

Answer to Redistribute the Employees by their Dept_No Quiz

Customer_Table		Order_Table		
Customer_Number	Customer_Name	Order_Number	Customer_Number	Order_Total
11111111	Billy's Best Choice	123456	11111111	12347.53
31313131	Acme Products	123512	11111111	8005.91
31323134	ACE Consulting	123552	31323134	5111.47
57896883	XYZ Plumbing	123585	87323456	15231.62
87323456	Databases N-U	123777	57896883	23454.84



The Teradata Hashing Formula is consistent. It is used to load a Table's rows via the Primary Index of the table. Teradata follows the same Hash Formula to Redistribute for Joins.

Each redistributed row landed on the same AMP as its matching row. Turn the page.

A Visual of the Join in Action

Query

```
SELECT Customer_Name, Order_Total  
FROM Customer_Table as C  
INNER JOIN Order_Table as O  
ON C.Customer_Number = O.Customer_Number ;
```

AMP 1

AMP 2

AMP 3

11111111 Billy's Best Choice

11111111 12347.53
11111111 8005.91 Spool

31313131 Acme Products
57896883 XYZ Plumbing

57896883 23454.84 Spool

31323134 ACE Consulting
87323456 Databases N-U

31323134 5111.47
87323456 15231.62 Spool

Report

Customer_Name	Order_Total
Billy's Best Choice	12347.53
Billy's Best Choice	8005.91
XYZ Plumbing	23454.84
ACE Consulting	5111.47
Databases N-U	15231.62

Nexus Query Chameleon

File Edit View Query Tools Windows Web Help

System: Teradata Database: sql_class

Execute

Query1*

```
SELECT Customer_Name, Order_Total
FROM Customer_Table as C
INNER JOIN
      Order_Table as O
ON      C.Customer_Number = O.Customer_Number ;
```

The PE's EXPLAIN PLAN is Below.

1) First, we **lock** a distinct SQL_CLASS."pseudo table" for **read** on a RowHash to prevent global deadlock for SQL_CLASS.O.
2) Next, we **lock** a distinct SQL_CLASS."pseudo table" for **read** on a RowHash to prevent global deadlock for SQL_CLASS.C.
3) We **lock** SQL_CLASS.O for **read**, and we **lock** SQL_CLASS.C for **read**.
4) We do an **all-AMPs RETRIEVE step** from SQL_CLASS.O by way of an **all-rows scan** with a condition of ("NOT (SQL_CLASS.O.Customer_Number IS NULL)") into **Spool 2 (all_amps)**, which is **redistributed** by the hash code of (SQL_CLASS.O.Customer_Number) to **all AMPs**. Then we do a **SORT** to order **Spool 2** by row hash. The size of **Spool 2** is **estimated** with **low confidence** to be **6 rows** (150 bytes). The **estimated time** for this **step** is **0.01 seconds**.
5) We do an **all-AMPs JOIN step** from **Spool 2 (Last Use)** by way of a RowHash match **scan**, which is joined to SQL_CLASS.C by way of a RowHash match **scan**. **Spool 2** and SQL_CLASS.C are joined using a

Systems

DB2 Greenplum Netezza Oracle Sandbox SQL SERVER

Teradata

SQL_CLASS Tables Addresses Addresses2 Claims Course_table Customer_table Department_dept Dept_Agg_G Emp_Job_tat Employee_ta Employee_ta Hierarchy_tal

Services Stats_table Student_Cou Student_table Subscribers

REDISTRIBUTE by Customer_Number

Row Hash Match Scan Joins the data

EXPLAIN SELECT Command Complete. 25 rows processed.

Time elapsed: 00:00:01 System: Teradata User: dbc Teradata

The Big Table/Small Table Join causes Duplication

Employee_Table					Department_Table	
Employee_No	Dept_No	Last_Name	First_Name	Salary	Dept_No	Department_Name
	100	Marketing				
	200	Research and Dev				
	300	Sales				
	400	Customer Support				

What if the Employee_Table Contained 1,000,000 rows

Query

```
SELECT Last_Name, Department_Name  
FROM Employee_Table as E , Department_Table as D  
WHERE E.Dept_No = D.Dept_No ;
```

PE Analysis
Phase 1

Is Dept_No the Primary Index of Employee_Table?

Is Dept_No the Primary Index of Department_Table?

NO

YES

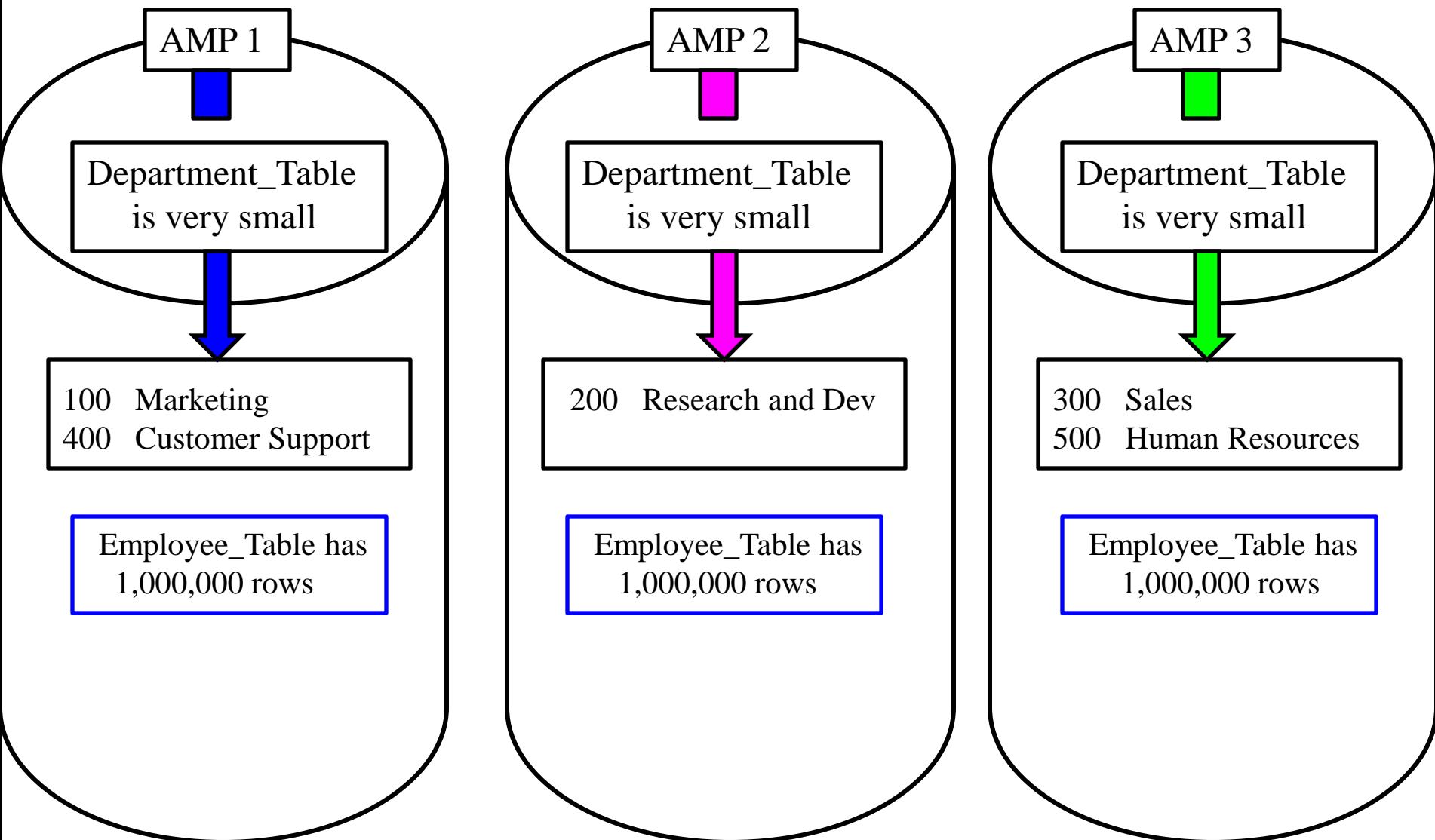
PE Analysis
Phase 2

Is the Employee_Table much larger than the Department_Table? YES

PLAN

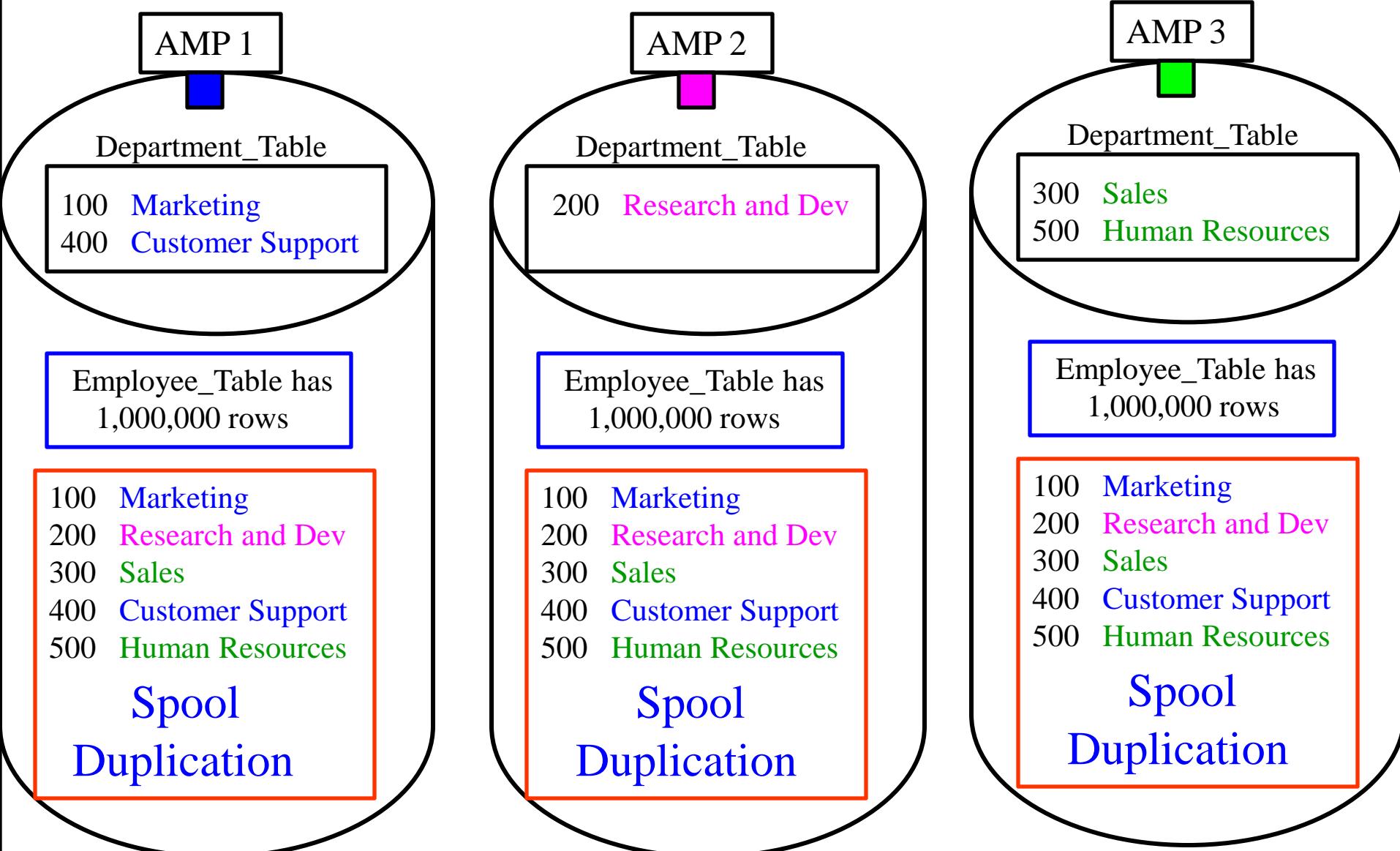
Stop! Don't Redistribute the 1,000,000 rows!
Duplicate the smaller Department_Table across all AMPs

Visual of Duplication of the Smaller Table across All-AMPs



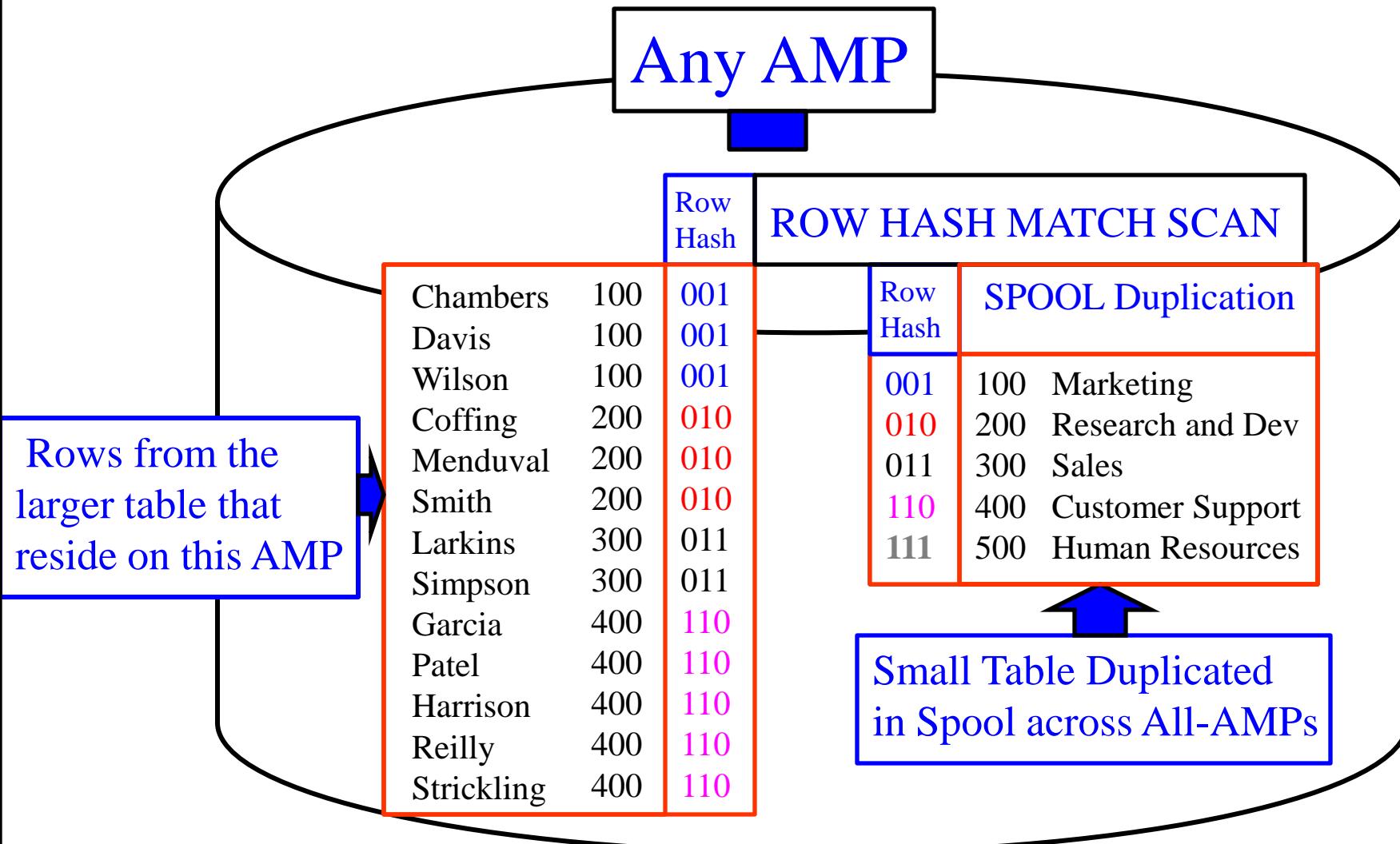
The next slide will demonstrate how Teradata **Duplicates** the **smaller** table (`Department_Table`) across all AMPs.

Duplication of the Smaller Table across All-AMPs



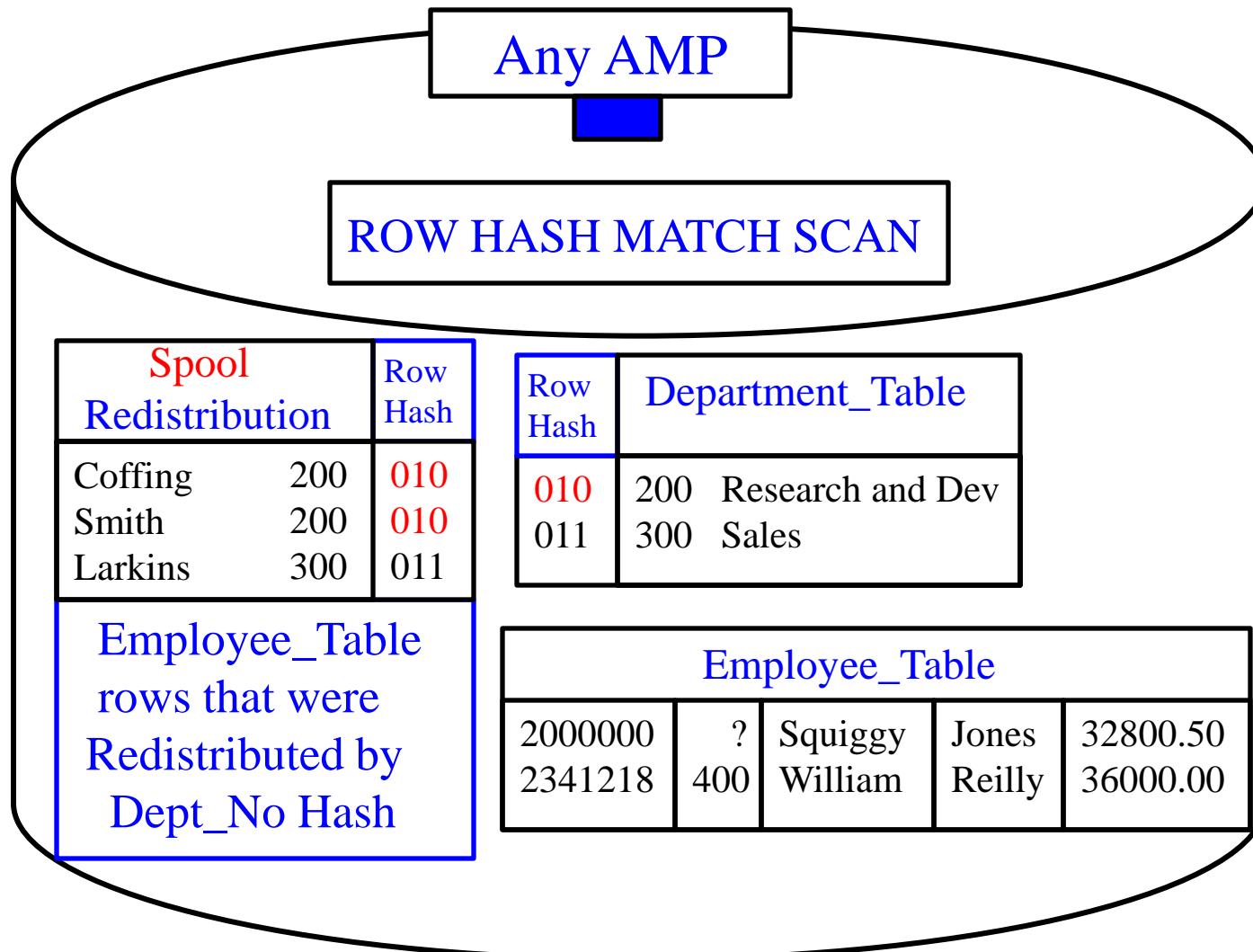
Teradata took the `Department_Table` and gathered up all 5-rows and then in Spool Duplicated the entire 5-row Table across all AMPS. Now the join can happen!

A Visual of Duplication of the Smaller Table on a Single AMP



The picture above shows the smaller table Duplicated in its entirety on this AMP in spool. This same spool table resides on every AMP for only the life of this join. Notice that both Dept_No's in the Table and Spool have corresponding Row Hashes.

A Visual of Redistribution on a Single AMP



The picture above shows the Department_Table on the top right that holds two rows on this particular AMP. The Employee_Table has been redistributed by Hash Code on Dept_No and the rows in spool have matches on the same AMP.

Chapter 10

Join Indexes

“A wise man thinks it more advantageous not to join the battle than to win.”

- Francois de La Rochefoucauld

Table of Contents Chapter 10 – Join Indexes

- [Creating a Multi-Table Join Index](#)
- [Visual of a Join Index](#)
- [Outer Join Multi-Table Join Index](#)
- [Visual of a Left Outer Join Index](#)
- [Compressed Multi-Table Join Index](#)
- [A Visual of a Compressed Multi-Table Join Index](#)
- [Creating a Single-Table Join Index](#)
- [Conceptual of a Single Table Join Index on an AMP](#)
- [Compressed Single-Table Join Index](#)
- [Aggregate Join Index](#)
- [Sparse Join Index](#)
- [A Global Multi-Table Join Index](#)
- [Creating a Hash Index](#)
- [Join Index Details](#)

Creating a Multi-Table Join Index

```
CREATE JOIN INDEX EMP_DEPT_IDX AS
SELECT Employee_No
      ,E.Dept_No
      ,First_Name
      ,Last_Name
      ,Salary
      ,Department_Name
FROM   Employee_Table as E
INNER JOIN
      Department_Table as D
ON     E.Dept_No = D.Dept_No
PRIMARY INDEX (Employee_No) ;
```

The Syntax above will create a Multi-Table Join Index with a NUPI on Employee_No. The next slide will illustrate a visual so you can see the data in the Join Index. Join Indexes are created so data doesn't have to move to satisfy the join. The Join Index essentially pre-joins the table and keeps it hidden for the Parsing Engine to utilize.

Visual of a Join Index

Employee_Table					Department_Table	
Employee_No	Dept_No	Last_Name	First_Name	Salary	Dept_No	Department_Name
1232578	100	Chambers	Mandee	48850.00	100	Marketing
1256349	400	Harrison	Herbert	54500.00	200	Research and Dev
2341218	400	Reilly	William	36000.00	300	Sales
2312225	300	Larkins	Lorraine	40200.00	400	Customer Support
2000000	?	Jones	Squiggy	32800.50	500	Human Resources
1000234	10	Smythe	Richard	32800.00		
1121334	400	Strickling	Cletus	54500.00		
1324657	200	Coffing	Billy	41888.88		
1333454	200	Smith	John	48000.00		

Join Index named **EMP_DEPT_IDX**

Employee_No	Dept_No	Last_Name	First_Name	Salary	Department_Name
1232578	100	Chambers	Mandee	48850.00	Marketing
1256349	400	Harrison	Herbert	54500.00	Customer Support
2341218	400	Reilly	William	36000.00	Customer Support
2312225	300	Larkins	Lorraine	40200.00	Sales
1121334	400	Strickling	Cletus	54500.00	Customer Support
1324657	200	Coffing	Billy	41888.88	Research and Dev
1333454	200	Smith	John	48000.00	Research and Dev

The Join Index looks like an Answer Set, but each row is stored like a normal table in that the rows of the Join Index are spread amongst the AMPs. Users can't query the Join Index, but the Parsing Engine gets data from the Join Index when it chooses.

Outer Join Multi-Table Join Index

```
CREATE JOIN INDEX EMP_DEPT_LEFTY AS  
SELECT Employee_No  
      ,E.Dept_No  
      ,First_Name  
      ,Last_Name  
      ,Salary  
      ,Department_Name  
FROM Employee_Table as E  
LEFT OUTER JOIN  
        Department_Table as D  
ON    E.Dept_No = D.Dept_No  
PRIMARY INDEX (Employee_No) ;
```

All Columns from the Outer Table must be in the SELECT List.

At least one Column from the INNER Table must be defined as NOT NULL.

A Multi-Table Outer Join Index has some very specific rules above to remember. Turn the page to see a visual of the data.

Visual of a Left Outer Join Index

Employee_Table					Department_Table	
Employee_No	Dept_No	Last_Name	First_Name	Salary	Dept_No	Department_Name
1232578	100	Chambers	Mandee	48850.00	100	Marketing
1256349	400	Harrison	Herbert	54500.00	200	Research and Dev
2341218	400	Reilly	William	36000.00	300	Sales
2312225	300	Larkins	Lorraine	40200.00	400	Customer Support
2000000	?	Jones	Squiggy	32800.50	500	Human Resources
1000234	10	Smythe	Richard	32800.00		
1121334	400	Strickling	Cletus	54500.00		
1324657	200	Coffing	Billy	41888.88		
1333454	200	Smith	John	48000.00		

Join Index named **EMP_DEPT_LEFTY**

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

The Outer Join Index has the additional rows that did NOT match.

Compressed Multi-Table Join Index

```
CREATE JOIN INDEX Cust_Order_IDX AS
SELECT
    (C.Customer_Number, Customer_Name),
    (Order_Number, Order_Date, Order_Total)
FROM Customer_Table as C
INNER JOIN
    Order_Table as O
ON      C.Customer_Number =
        O.Customer_Number
PRIMARY INDEX (Customer_Number) ;
```

A Compressed Multi-Table Join Index won't keep repeating the same Customer_Number and Customer_Name, but only list it once. A visual example follows on the next page.

A Visual of a Compressed Multi-Table Join Index

Customer_Table		Order_Table		
Customer_Number	Customer_Name	Order_Number	Customer_Number	Order_Total
11111111	Billy's Best Choice	123456	11111111	12347.53
31313131	Acme Products	123512	11111111	8005.91
31323134	ACE Consulting	123552	31323134	5111.47
57896883	XYZ Plumbing	123585	87323456	15231.62
87323456	Databases N-U	123777	57896883	23454.84

Not
Repeated

Join Index named **CUST_ORDER_IDX**

Customer_Number	Customer_Name	Order_Number	Order_Date	Order_Total
11111111	Billy's Best Choice	123456	05/04/1998	12347.53
		123512	01/01/1999	8005.91
31323134	ACE Consulting	123552	10/01/1999	5111.47
57896883	XYZ Plumbing	123777	09/09/1999	15231.62
87323456	Databases N-U	123585	10/10/1999	23454.84

Billy's Best Choice is Customer_Number 11111111 and they have placed two orders, but the Customer_Number and Customer_Name don't repeat unnecessarily.

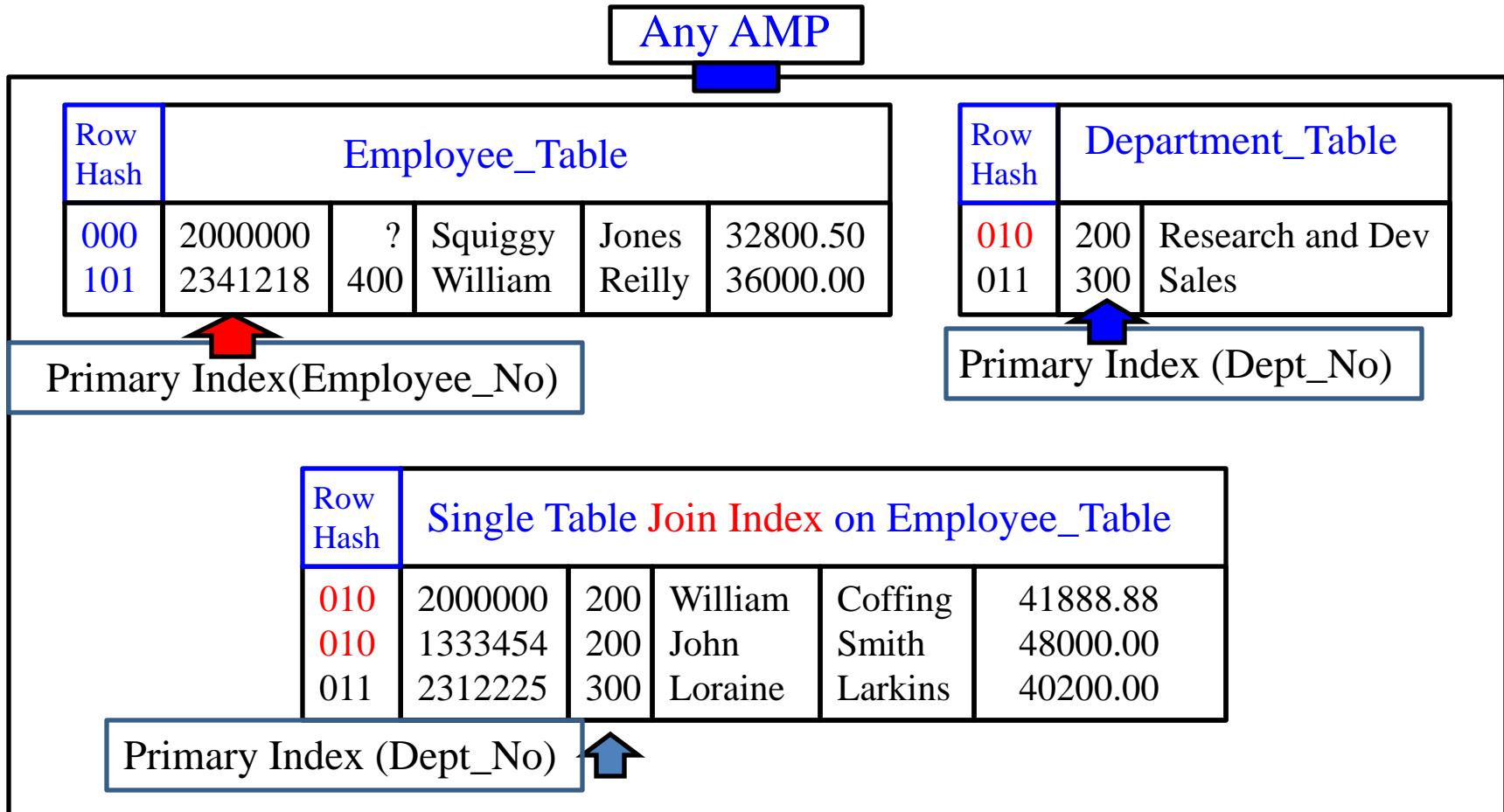
Creating a Single-Table Join Index

```
CREATE JOIN INDEX Employee_IDX  
AS  
SELECT Employee_No  
      ,Dept_No  
      ,First_Name  
      ,Last_Name  
      ,Salary  
FROM Employee_Table  
PRIMARY INDEX (Dept_No) ;
```

We've duplicated the Employee_Table with a different Primary Index.

If a USER queries with the Dept_No in the WHERE clause this will be a Single-AMP retrieve. If the USER joins the Employee and Department Tables together then Teradata won't need to Redistribute or Duplicate to get the data AMP local. The next page will give you a visual of how that looks on a particular AMP.

Conceptual of a Single Table Join Index on an AMP

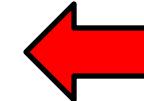


Notice the Primary Indexes on both tables and the Single Table Join Index. The Join Index gives the Parsing Engine options. If a query is run against the Employee_Table with Employee_No in the WHERE clause it will use the normal table, but if a user Uses Dept_No in the WHERE clause it will use the Join Index. If a user needs to join the Department_Table to the Employee_Table the Join Index is used so no data moves.

Compressed Single-Table Join Index

```
CREATE JOIN INDEX Order_IDX  
AS  
SELECT (Customer_Number)  
       ,(Order_Number  
       ,Order_Date  
       ,Order_Total)  
FROM   Order_Table  
PRIMARY INDEX (Customer_Number) ;
```

Parentheses
around
Customer_Number



We've duplicated the **Order_Table** with a **different Primary Index**.

This is the compressed version of a Single-Table Join Index. Notice the parentheses around Customer_Number in the SELECT list. A single row is used for each customer, with repeating orders per customer inside the Join Index.

Aggregate Join Index

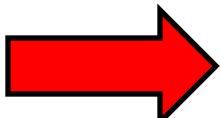
```
CREATE JOIN INDEX Agg_Order_IDX AS
SELECT
    Customer_Number
    ,Extract(Year from Order_Date) As Yr
    ,Extract(Month from Order_Date) As Mon
    ,Count(*) as County
    ,Sum(Order_Total) as Summy
FROM Order_Table
Group by 1, 2, 3;
```

Only **Sum** and **Count** can be used with an Aggregate Join Index.

Users never query the Join Index directly. It is the Parsing Engine that commands the AMPs to pull the data from the Join Index. You can extract the Month and Year and calculations are updated as the Base Table changes. Count and Sum are required to be a data type of FLOAT. Why can you only have Count and Sum? Max and Min aren't that important to know and Average isn't all that important either. Business users want to know how much and how many so AVERAGE, MIN, and MAX are excluded!

Sparse Join Index

```
CREATE JOIN INDEX Sparse_Order_IDX AS
SELECT Order_Number
      ,Customer_Number
      ,Order_Total
      ,Order_Date
FROM Order_Table
WHERE Order_Date BETWEEN
      '1999-10-01' AND '1999-12-31'
Primary Index (Customer_Number);
```



A **Sparse** Join Index is a Join Index with a **WHERE Clause!**

A **Sparse** Join Index has a **WHERE clause** so it doesn't take all the rows in the table, but only a portion. This is a very effective way to save space and focus on the latest data.

A Global Multi-Table Join Index

```
CREATE JOIN INDEX EMP_DEPT_Glob
AS SELECT Employee_No
        ,E.Dept_No
        ,First_Name
        ,Last_Name
        ,E.ROWID as EmpRI
        ,Department_Name
        ,D.ROWID as DeptRI
FROM Employee_Table as E
INNER JOIN
        Department_Table as D
ON      E.Dept_No = D.Dept_No
PRIMARY INDEX (Dept_No) ;
```

With the ROWID inside the Join Index the PE can get columns in the User's SQL **NOT** specified in the Join Index directly from the Base Table by using the **Row-ID**.

Creating a Hash Index

Example 1

```
CREATE HASH INDEX EMP_Hash_IDX  
(Dept_No ,First_Name ,Last_Name)  
ON Employee_Table;
```

Ordered by **Hash** of Primary Index

Example 2

```
CREATE HASH INDEX EMP_Hash_Val  
(Dept_No ,First_Name ,Last_Name)  
ON Employee_Table  
ORDER BY VALUES (Dept_No);
```

Ordered by the **Value** Dept_No

A Hash Index can be Ordered by **Values** or by **Hash**.

Join Index Details

- Join Indexes are **physically** stored exactly like normal Teradata tables.
- Users can't query the Join Index directly, but **PE** will decide when to use.
- Join Indexes are **automatically** updated as base tables change.
- Join Indexes can have Non-Unique Primary Indexes (**NUPI**) only.
- Join Indexes can have Non-Unique Secondary (**NUSI**) Indexes.
- Max **64** Columns per Table per Join Index.
- BLOB and CLOB types **cannot** be defined.
- Triggers with Join Indexes allowed V2R6.2.
- After Restoring a Table, Drop and Recreate the Join Index.
- **FastLoad/MultiLoad** won't load to tables with a Join Index defined.

Chapter 11

Basic SQL Functions

“A journey of a thousand miles begins with a single step”

- Lao-tzu

Table of Contents Chapter 11 - Basic SQL Functions

- [Introduction](#)
- [SELECT * \(All Columns\) in a Table](#)
- [SELECT Specific Columns in a Table](#)
- [Using Good Form](#)
- [Using the Best Form for Writing SQL](#)
- [Place your Commas in front for better Debugging Capabilities](#)
- [Sort the Data with the ORDER BY Keyword](#)
- [ORDER BY Defaults to Ascending](#)
- [Use the Name or the Number in your ORDER BY Statement](#)
- [ORDER BY Defaults to Ascending](#)
- [Two Examples of ORDER BY using Different Techniques](#)
- [NULL Values sort First in Ascending Mode \(Default\)](#)
- [NULL Values sort Last in Descending Mode \(DESC\)](#)
- [Multiple Sort Keys using Names vs. Numbers](#)
- [Major Sort vs. Minor Sorts](#)
- [Sorts are Alphabetical, NOT Logical](#)
- [Using A CASE Statement to Sort Logically](#)
- [How to ALIAS a Column Name](#)
- [A Missing Comma can by Mistake become an Alias](#)
- [The Title Command and Literal Data](#)
- [Comments using Double Dashes](#)
- [Comments for Multi-Lines](#)
- [Comments using Double Dashes for Multiple Lines](#)

Introduction

Student_Table

<u>Student_ID</u>	<u>Last_Name</u>	<u>First_Name</u>	<u>Class_Code</u>	<u>Grade_Pt</u>
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

The **Student_Table** above will be used
in our early SQL Examples

This is a pictorial of the **Student_Table** which we will use to present some basic examples of SQL and get some hands-on experience with querying this table. This book attempts to show you the table, show you the query, and show you the result set.

SELECT * (All Columns) in a Table

Student_Table				
Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

```
SELECT *
FROM    Student_Table ;
```

Mostly every SQL statement will consist of a SELECT and a FROM. You SELECT the columns you want to see on your report and an Asterisk (*) means you want to see all columns in the table on the returning answer set!

SELECT Specific Columns in a Table

Student_Table

<u>Student_ID</u>	<u>Last_Name</u>	<u>First_Name</u>	<u>Class_Code</u>	<u>Grade_Pt</u>
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

```
Select First_Name, Last_Name, Class_Code, Grade_Pt  
FROM Student_Table ;
```

Column names must be separated by commas. The next page will show perfect syntax, which will capitalize keywords and place each column on its own line.

Using Good Form

Student_Table

Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

```
SELECT First_Name,  
       Last_Name,  
       Class_Code,  
       Grade_Pt  
FROM   Student_Table ;
```

This is a great way to show the **columns** you are **selecting** from the **Table_Name**. Let me show you an even better technique!

Using the Best Form for Writing SQL

Student_Table

Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

```
SELECT First_Name,  
       Last_Name,  
       Class_Code,  
       Grade_Pt  
FROM   Student_Table ;
```

```
SELECT First_Name  
      ,Last_Name  
      ,Class_Code  
      ,Grade_Pt  
FROM   Student_Table ;
```

Can you
spot the
difference
between
these two
examples?

Why is the example on the right better even though they are functionally equivalent?

Place your Commas in front for better Debugging Capabilities

Student_Table

Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

```
SELECT First_Name,  
       Last_Name,  
       Class_Code,  
       Grade_Pt,  
       Student_ID,  
FROM    Student_Table ;
```

Sometimes
if you **Add**
or **Remove**
a COLUMN
you could
overlook a
Comma!

```
SELECT First_Name  
      ,Last_Name  
      ,Class_Code  
      ,Grade_Pt  
      ,Student_ID  
FROM    Student_Table ;
```

Having commas in front to separate column names makes it easier to debug.

Sort the Data with the ORDER BY Keyword

Student_Table

Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

Sorts the
Answer Set
In **Ascending**
Order by default

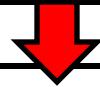
SELECT *
FROM Student_Table
ORDER BY Last_Name ;

Rows typically come back to the report in random order. To order the result set, you must use an ORDER BY. When you order by a column, it will order in ASCENDING order. This is called the Major Sort!

ORDER BY Defaults to Ascending

Sorts the Answer Set
In **Ascending** Order
By **Last_Name**

```
SELECT      *  
FROM    Student_Table  
ORDER BY Last_Name ;
```



Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
322133	Bond	Jimmy	JR	3.95
324652	Delaney	Danny	SR	3.35
125634	Hanson	Henry	FR	2.88
260000	Johnson	Stanley	?	?
423400	Larkins	Michael	FR	0.00
280023	McRoberts	Richard	JR	1.90
123250	Phillips	Martin	SR	3.00
333450	Smith	Andy	SO	2.00
234121	Thomas	Wendy	FR	4.00
231222	Wilson	Susie	SO	3.80

Rows typically come back to the report in random order, but we decided to use the ORDER BY statement. Now the data comes back ordered by Last_Name.

Use the Name or the Number in your ORDER BY Statement

Sorts the Answer Set
by Column 2 which is
Last_Name

```
SELECT      *  
FROM    Student_Table  
ORDER BY 2 ;
```



Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
322133	Bond	Jimmy	JR	3.95
324652	Delaney	Danny	SR	3.35
125634	Hanson	Henry	FR	2.88
260000	Johnson	Stanley	?	?
423400	Larkins	Michael	FR	0.00
280023	McRoberts	Richard	JR	1.90
123250	Phillips	Martin	SR	3.00
333450	Smith	Andy	SO	2.00
234121	Thomas	Wendy	FR	4.00
231222	Wilson	Susie	SO	3.80

The ORDER BY can use a number to represent the sort column. The number 2 represents the second column on the report.

ORDER BY Defaults to Ascending

Sorts the Answer Set In **DESC** Order By **Last_Name**

```
SELECT *  
FROM Student_Table  
ORDER BY Last_Name DESC;
```

Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
333450	Smith	Andy	SO	2.00
123250	Phillips	Martin	SR	3.00
280023	McRoberts	Richard	JR	1.90
423400	Larkins	Michael	FR	0.00
260000	Johnson	Stanley	?	?
125634	Hanson	Henry	FR	2.88
324652	Delaney	Danny	SR	3.35
322133	Bond	Jimmy	JR	3.95

If you want the data sorted in descending order just place DESC at the end.

Two Examples of ORDER BY using Different Techniques

Student_Table

Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

```
SELECT      *
FROM        Student_Table
ORDER BY 5 ;
```

← Same Query →

```
SELECT      *
FROM        Student_Table
ORDER BY Grade_Pt ;
```



You have the option of using a number instead of the column name. The columns number is represented by what position it is in the SELECT statement, not the table. If you use an * in your Select Statement, then the columns number is represented by the position it is in the table.

NULL Values sort First in Ascending Mode (Default)

```
SELECT *
FROM Student_Table
ORDER BY 5 ;
```

```
SELECT *
FROM Student_Table
ORDER BY Grade_Pt ;
```

<u>Student_ID</u>	<u>Last_Name</u>	<u>First_Name</u>	<u>Class_Code</u>	<u>Grade_Pt</u>
260000	Johnson	Stanley	?	?
423400	Larkins	Michael	FR	0.00
280023	McRoberts	Richard	JR	1.90
333450	Smith	Andy	SO	2.00
125634	Hanson	Henry	FR	2.88
123250	Phillips	Martin	SR	3.00
324652	Delaney	Danny	SR	3.35
231222	Wilson	Susie	SO	3.80
322133	Bond	Jimmy	JR	3.95
234121	Thomas	Wendy	FR	4.00

Nulls

The default for ORDER BY is Ascending mode (ASC). Notice that this places the Null Values at the beginning of the Answer Set.

NULL Values sort Last in Descending Mode (DESC)

```
SELECT      *
FROM    Student_Table
ORDER BY 5 DESC ;
```

```
SELECT      *
FROM    Student_Table
ORDER BY Grade_Pt DESC ;
```

<u>Student_ID</u>	<u>Last_Name</u>	<u>First_Name</u>	<u>Class_Code</u>	<u>Grade_Pt</u>
234121	Thomas	Wendy	FR	4.00
322133	Bond	Jimmy	JR	3.95
231222	Wilson	Susie	SO	3.80
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
125634	Hanson	Henry	FR	2.88
333450	Smith	Andy	SO	2.00
280023	McRoberts	Richard	JR	1.90
423400	Larkins	Michael	FR	0.00
260000	Johnson	Stanley	?	?

Nulls are
Last in
DESC
Order



You can ORDER BY in descending order by putting a DESC after the column name or its corresponding number. Null Values will sort Last in DESC order.

Multiple Sort Keys using Names vs. Numbers

```
SELECT Employee_No  
      ,Dept_No  
      ,First_Name  
      ,Last_Name  
      ,Salary  
FROM Employee_Table  
ORDER BY Dept_No DESC  
        ,Last_Name ASC  
        ,Salary DESC ;
```

```
SELECT Employee_No  
      ,Dept_No  
      ,First_Name  
      ,Last_Name  
      ,Salary  
FROM Employee_Table  
ORDER BY 2 DESC,  
        4,  
        5 DESC ;
```

These queries sort identically

Queries can have a multiple columns in the ORDER BY. The first column in an ORDER BY is called the **MAJOR SORT**. Those after it are **MINOR SORTS**.

Both these Queries do the same thing. Once they sort Dept_No column in DESC order, they'll sort any ties by LAST_NAME and then if any ties still occur, they'll sort by SALARY. Let me show you a real world example in the next slide!

Major Sort vs. Minor Sorts

```
SELECT * FROM Student_Table  
ORDER BY Class_Code DESC,  
Grade_Pt ASC;
```

Major Sort on
Class_Code DESCENDING

Minor Sort on
Grade_Pt Ascending

Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
123250	Phillips	Martin	SR	3.00
324652	Delaney	Danny	SR	3.35
333450	Smith	Andy	SO	2.00
231222	Wilson	Susie	SO	3.80
280023	McRoberts	Richard	JR	1.90
322133	Bond	Jimmy	JR	3.95
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
234121	Thomas	Wendy	FR	4.00
260000	Johnson	Stanley	?	?

Major
Sorts
First
In
DESC
Order

Minor
Sorts
On
Ties

Major sort is how things are sorted, but a minor sort kicks in if there are Major Sort ties.

Sorts are Alphabetical, NOT Logical

```
SELECT * FROM Student_Table  
ORDER BY Class_Code ;
```

Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
260000	Johnson	Stanley	?	?
234121	Thomas	Wendy	FR	4.00
125634	Hanson	Henry	FR	2.88
423400	Larkins	Michael	FR	0.00
322133	Bond	Jimmy	JR	3.95
280023	McRoberts	Richard	JR	1.90
231222	Wilson	Susie	SO	3.80
333450	Smith	Andy	SO	2.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00

This sorts alphabetically, but Sophomores (SO) logically come after Freshman

Change the query to Order BY Class_Code logically (FR, SO, JR, SR, ?)

Using A CASE Statement to Sort Logically

```
SELECT * FROM Student_Table  
ORDER BY CASE Class_Code  
          WHEN 'FR' THEN 1  
          WHEN 'SO' THEN 2  
          WHEN 'JR' THEN 3  
          WHEN 'SR' THEN 4  
          ELSE 5  
      END;
```

Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
234121	Thomas	Wendy	FR	4.00
125634	Hanson	Henry	FR	2.88
423400	Larkins	Michael	FR	0.00
333450	Smith	Andy	SO	2.00
231222	Wilson	Susie	SO	3.80
322133	Bond	Jimmy	JR	3.95
280023	McRoberts	Richard	JR	1.90
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
260000	Johnson	Stanley	?	?

How to ALIAS a Column Name

```
SELECT First_Name AS Fname  
      ,Last_Name      Lname  
      ,Class_Code    "Class Code"  
      ,Grade_Pt     AS "AVG"  
      ,Student_ID   AS STU_ID  
FROM   Student_Table ;
```

Above are techniques for Aliasing

ALIAS Rules!

- 1) AS is optional
- 2) Use Double Quotes when Spaces are in the Alias name
- 3) Use Double Quotes when the Alias is a reserved word

When you ALIAS a column you give it a new name for the report header. You should always reference the column using the ALIAS everywhere else in the query. You never need Double Quotes in SQL unless you are Aliasing.

A Missing Comma can by Mistake become an Alias

```
SELECT First_Name, Last_Name, Class_Code Grade_Pt  
FROM Student_Table ;
```

Missing a Comma

First_Name	Last_Name	Grade_Pt
Michael	Larkins	FR
Susie	Wilson	SO
Richard	McRoberts	JR
Jimmy	Bond	JR
Henry	Hanson	FR
Andy	Smith	SO
Danny	Delaney	SR
Stanley	Johnson	?
Wendy	Thomas	FR
Martin	Phillips	SR

Aliased as Grade_Pt

Column names must be separated by commas. Notice in this example, there is a comma missing between Class_Code and Grade_Pt. What this will result in is only three columns appearing on your report with one being titled wrong.

The Title Command and Literal Data

```
SELECT 'Character Data'  
      , 'Character Data' (TITLE 'Character // Data')  
      , 123 (TITLE 'Numeric Data')  
      , 'Character Data' (TITLE ' My//Stacked//Example') ;
```

Stacks the Report Header

Character Data	Character Data	Numeric Data	MY Stacked Example
Character Data	Character Data	123	Character Data

A Literal Value brings back the Literal Value! Also notice that the word ‘Character’ is stacked over the ‘Data’ portion of the heading for the second column using the Nexus Query Chameleon. So, as an alternative, a TITLE can be used instead of an alias and allows the user to include spaces in the output title.

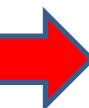
The difference between an ALIAS and a TITLE is that the ALIAS can be used in the SQL again such as in the ORDER BY or WHERE statements, but a TITLE is only good for the report heading. Notice that Title uses Single Quotes not double quotes.

Comments using Double Dashes

Student_Table

Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

Comment



-- Double Dashes provide a single line comment

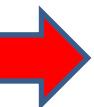
```
SELECT      Distinct Class_Code, Grade_Pt  
FROM        Student_Table  
ORDER BY    Class_Code, Grade_Pt;
```

Double dashes make a single line comment that will be ignored by the system.

Comments for Multi-Lines

Student_Table				
Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

Comment



```
/* This is a multi-line comment that will
   allow me to comment anything      */
SELECT Distinct Class_Code, Grade_Pt
FROM    Student_Table ;
```

Slash Asterisk starts a multi-line comment and Asterisk Slash ends the comment.

Comments using Double Dashes for Multiple Lines

Student_Table

Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

Comment

-- This is a multi-line comment that will
-- allow me to comment anything

Comment

```
SELECT Distinct Class_Code, Grade_Pt  
FROM Student_Table ;
```

Double Dashes in front of both lines comments both lines out and they're ignored.

Chapter 12

The WHERE Clause

“I have a dream that my four little children will one day live in a nation where they will not be judged by the color of their skin, but by the content of their character.

- Martin Luther King, Jr.

Table of Contents Chapter 12 - The WHERE Clause

- [The WHERE Clause limits Returning Rows](#)
- [Using a Column ALIAS throughout the SQL](#)
- [Double Quoted Aliases are for Reserved Words and Spaces](#)
- [Character Data needs Single Quotes in the WHERE Clause](#)
- [Character Data needs Single Quotes, but Numbers Don't](#)
- [NULL means UNKNOWN DATA so Equal \(=\) won't Work](#)
- [Use IS NULL or IS NOT NULL when dealing with NULLs](#)
- [NULL is UNKNOWN DATA so NOT Equal won't Work](#)
- [Use IS NULL or IS NOT NULL when dealing with NULLs](#)
- [Using Greater Than Or Equal To \(>=\)](#)
- [Using GE as Greater Than or Equal To \(>=\)](#)
- [AND in the WHERE Clause](#)
- [Troubleshooting AND](#)
- [OR in the WHERE Clause](#)
- [Troubleshooting OR](#)
- [OR must utilize the Column Name Each Time](#)
- [Troubleshooting Character Data](#)
- [Using Different Columns in an AND Statement](#)
- [Quiz – How many rows will return?](#)
- [Answer to Quiz – How many rows will return?](#)
- [What is the Order of Precedence?](#)
- [Using Parentheses to change the Order of Precedence](#)
- [Using an IN List in place of OR](#)
- [IN List vs. OR brings the same Results](#)
- [Using a NOT IN List](#)

Continued on next page

Table of Contents Chapter 12 - The WHERE Clause Continued

- [A Technique for Handling Nulls with a NOT IN List](#)
- [An IN List with the Keyword ANY](#)
- [A NOT IN List with the Keywords NOT = ALL](#)
- [BETWEEN is Inclusive](#)
- [BETWEEN works for Character Data](#)
- [LIKE uses Wildcards Percent '%' and Underscore '_'](#)
- [LIKE command Underscore is Wildcard for one Character](#)
- [LIKE ALL means ALL conditions must be Met](#)
- [LIKE ANY means ANY of the Conditions can be Met](#)
- [IN ANSI Transaction Mode Case Matters](#)
- [In Teradata Transaction Mode Case Doesn't Matter](#)
- [LIKE Command Works Differently on Char Vs Varchar](#)
- [Troubleshooting LIKE Command on Character Data](#)
- [Quiz – What Data is Left Justified and What is Right?](#)
- [Numbers are Right Justified and Character Data is Left?](#)
- [Answer – What Data is Left Justified and What is Right?](#)
- [An Example of Data with Left and Right Justification](#)
- [A Visual of CHARACTER Data vs. VARCHAR Data](#)
- [Use the TRIM command to remove spaces on CHAR Data](#)
- [TRIM Eliminates Leading and Trailing Spaces](#)
- [Escape Character in the LIKE Command changes Wildcards](#)
- [Escape Characters Turn off Wildcards in the LIKE Command](#)
- [Quiz – Turn off that Wildcard](#)
- [ANSWER – To Find that Wildcard](#)

The WHERE Clause limits Returning Rows

Student_Table

Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

```
SELECT First_Name, Last_Name, Class_Code, Grade_Pt  
FROM Student_Table  
WHERE First_Name = 'Henry';
```



The WHERE Clause here filters how many ROWS are coming back. In this example, I am asking for the report to only rows WHERE the first name is Henry.

Using a Column ALIAS throughout the SQL

```
SELECT First_Name AS Fname  
      ,Last_Name      Lname  
      ,Class_Code     "Class Code"  
      ,Grade_Pt AS   "AVG"  
      ,Student_ID  
FROM   Student_Table  
WHERE Fname = 'Henry';
```

Aliasing
a column

Use the ALIAS in your WHERE Clause!

When you ALIAS a column you give it a new name for the report header, but a good rule of thumb is to refer to the column by the alias throughout the query.

Double Quoted Aliases are for Reserved Words and Spaces

```
SELECT First_Name AS Fname  
      ,Last_Name    Lname  
      ,Class_Code   "Class Code"  
      ,Grade_Pt    AS "AVG"  
      ,Student_ID  
FROM   Student_Table  
WHERE Fname = 'Henry'  
ORDER BY "AVG";
```

If Double Quotes
are used here then use
Double Quotes
here also.

Use the **ALIAS** in your
remaining SQL!

When you **ALIAS** a column you give it a **new name** for the report header, but a good rule of thumb is to refer to the column by the alias throughout the query.

Character Data needs Single Quotes in the WHERE Clause

Student_Table				
Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

```
SELECT First_Name, Last_Name, Class_Code Grade_Pt  
FROM Student_Table  
WHERE First_Name = 'Henry';
```



Character Data always needs Single Quotes in the WHERE Clause, numbers don't!

In the WHERE clause, if you search for Character data such as first name, you need single quotes around it. You Don't single-quote integers.

Character Data needs Single Quotes, but Numbers Don't

Student_Table				
Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

```
SELECT *
FROM Student_Table
WHERE First_Name = 'Henry' ;
```

```
SELECT *
FROM Student_Table
WHERE Grade_Pt = 0.00 ;
```



Character data (letters) need **single quotes**, but you need **NO Single Quotes** for Integers (numbers). Remember you never use double quotes except for aliasing.

NULL means UNKNOWN DATA so Equal (=) won't Work

Student_Table				
Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

```
SELECT *
FROM Student_Table
WHERE Class_Code = NULL ;
```



First thing you need to know about a NULL is it is unknown data. It is NOT a zero. It is missing data. Since we don't know what is in a NULL you can't use an = sign. You must use IS NULL or IS NOT NULL.

Use IS NULL or IS NOT NULL when dealing with NULLs

Student_Table

Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	NULL
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

```
SELECT *
FROM   Student_Table
WHERE  Class_Code IS NULL ;
```



If you are looking for a row that holds NULL value, you need to put 'IS NULL'. This will only bring back the rows with a NULL value in it.

NULL is UNKNOWN DATA so NOT Equal won't Work

Student_Table

Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

```
SELECT *
FROM   Student_Table
WHERE  Class_Code = NOT NULL ;
```



The same goes with = NOT NULL. We can't compare a NULL with any equal sign.
We can only deal with NULL values with IS NULL and IS NOT NULL.

Use IS NULL or IS NOT NULL when dealing with NULLs

Student_Table

Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

```
SELECT *
FROM Student_Table
WHERE Class_Code IS NOT NULL ;
```

Much like before, when you want to bring back the rows that do not have NULLs in them, you put an 'IS NOT NULL' in the WHERE Clause.

Using Greater Than Or Equal To (\geq)

Student_Table

Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

```
SELECT * FROM Student_Table  
WHERE Grade_Pt  $\geq$  3.0 ;
```

Greater than
or Equal to

The WHERE Clause doesn't just deal with 'Equals'. You can look for things that are GREATER or LESSER THAN along with asking for things that are GREATER/LESSER THAN or EQUAL to.

Using GE as Greater Than or Equal To (>=)

Student_Table				
Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

```
SELECT * FROM Student_Table  
WHERE Grade_Pt GE 3.0 ;
```

Greater than
or Equal to

The syntax above uses a Teradata extension (GE) for Greater Than or Equal To!

AND in the WHERE Clause

Student_Table

Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

```
SELECT      *
FROM        Student_Table
WHERE       Class_Code = 'FR' AND First_Name = 'Henry';
```



Notice the WHERE statement and the word AND. In this example, qualifying rows must have a Class_Code = 'FR' and also must have a First_Name of 'Henry'.

Troubleshooting AND

Student_Table				
Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

```
SELECT      *
FROM        Student_Table
WHERE Grade_Pt = 3.0 AND Grade_Pt = 4.0;
```



What is going wrong here? You are using an **AND** to check the **same** column. What you are basically asking with this syntax is to see the rows that have **BOTH** a Grade_Pt of 3.0 and a 4.0. That is impossible so no rows will be returned.

OR in the WHERE Clause

Student_Table

Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

```
SELECT      *
FROM        Student_Table
WHERE       Grade_Pt = 3.0 OR Grade_Pt = 4.0;
```



Notice above in the WHERE Clause we use **OR**. Or allows for either of the parameters to be TRUE in order for the data to qualify and return.

Troubleshooting OR

Student_Table				
Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

```
SELECT      *
FROM        Student_Table
WHERE       Grade_Pt = 3.0 OR 4.0;
```



A Common Mistake

This causes an error! Why? You need to state the column name again before the 4.0.

OR must utilize the Column Name Each Time

Student_Table				
Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

```
SELECT      *
FROM        Student_Table
WHERE Grade_Pt = 3.0 OR Grade_Pt = 4.0;
```



Notice that you must always state the COLUMN NAME along with the parameter.
Even if you are using the same Column Name, you must specify it over again.

Troubleshooting Character Data

Student_Table

<u>Student_ID</u>	<u>Last_Name</u>	<u>First_Name</u>	<u>Class_Code</u>	<u>Grade_Pt</u>
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

```
SELECT      *
FROM        Student_Table
WHERE       Grade_Pt = 3.0 AND Class_Code = SR ;
```

This query errors! What is WRONG with this syntax? No Single quotes around SR.

Using Different Columns in an AND Statement

Student_Table

Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

```
SELECT      *
FROM        Student_Table
WHERE Grade_Pt = 3.0 AND Class_Code = 'SR' ;
```

Notice that AND separates two different columns and the data will come back if both are TRUE.

Quiz – How many rows will return?

Student_Table

Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

```
SELECT      *
FROM        Student_Table
WHERE       Grade_Pt = 4.0 OR Grade_Pt = 3.0
AND Class_Code = 'SR';
```

Which Seniors have a 3.0 or a 4.0 Grade_Pt average. How many rows will return?

- A) 2
- C) Error
- B) 1
- D) 3

Answer to Quiz – How many rows will return?

Student_Table				
Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

```
SELECT *
FROM Student_Table
WHERE Grade_Pt = 4.0 OR Grade_Pt = 3.0
AND Class_Code = 'SR';
```

We had two rows return? Isn't that a mystery! Why?

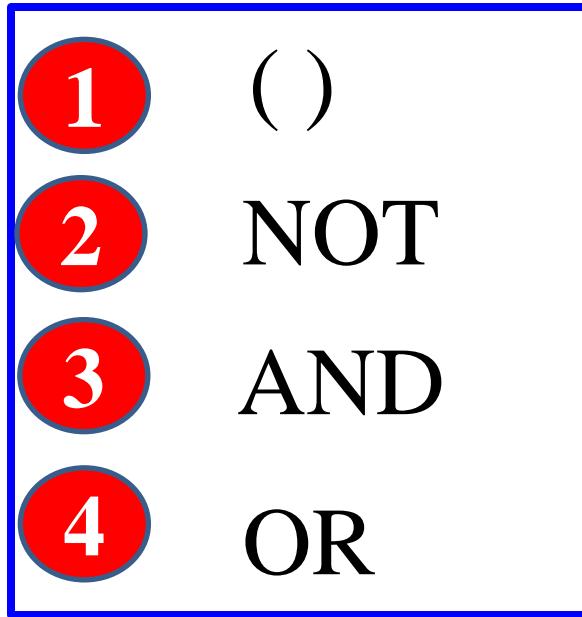
A) 2

B) 1

C) Error

D) 3

What is the Order of Precedence?



```
SELECT *
FROM Student_Table
WHERE Grade_Pt = 4.0 OR Grade_Pt = 3.0
      AND Class_Code = 'SR';
```

Syntax has an ORDER OF PRECEDENCE. It will read anything with parentheses around it first. Then it will read all the NOT statements. Then the AND statements. FINALLY the OR Statements. This is why the last query came out odd. Let's fix it and bring back the right answer set.

Using Parentheses to change the Order of Precedence

Student_Table

Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

```
SELECT *
FROM   Student_Table
WHERE  (Grade_Pt = 3.0 OR Grade_Pt = 4.0) ←
AND    Class_Code = 'SR';
```

Parentheses
Evaluated
First!

This is the proper way of looking for rows that have both a Grade_Pt of 3.0 or 4.0 AND also having a Class_Code of 'SR'. Only ONE row comes back. Parentheses are evaluated first, so this allows you to direct exactly what you want to work first.

Using an IN List in place of OR

Student_Table

<u>Student_ID</u>	<u>Last_Name</u>	<u>First_Name</u>	<u>Class_Code</u>	<u>Grade_Pt</u>
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

```
SELECT *
FROM Student_Table
WHERE Grade_Pt IN (3.0, 4.0)
AND Class_Code = 'SR';
```

The IN List

Using an IN List is a great way of looking for rows that have both a Grade_Pt of 3.0 or 4.0 AND also have a Class_Code of 'SR'. Only ONE row comes back.

IN List vs. OR brings the same Results

Student_Table				
Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

```
SELECT * FROM Student_Table  
WHERE Grade_Pt IN (2.0, 3.0, 4.0);
```

Better
Technique



```
SELECT *  
FROM Student_Table  
WHERE Grade_Pt = 2.0  
OR Grade_Pt = 3.0  
OR Grade_Pt = 4.0;
```

Both
Produce
the same
results

The IN Statement avoids retying the same column name separated by an OR. The IN allows you to search the same column for a list of values. Both queries above are equal, but the IN list is a nice way to keep things easy and organized.

Using a NOT IN List

Student_Table				
Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

```
SELECT *
FROM   Student_Table
WHERE  Grade_Pt NOT IN (2.0, 3.0, 4.0);
```

You can also ask to see the results that ARE NOT IN your parameter list. That requires the column name and a NOT IN. Neither the IN nor NOT IN can search for **NULLs!**

A Technique for Handling Nulls with a NOT IN List

Student_Table				
Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

```
SELECT *
FROM Student_Table
WHERE Grade_Pt NOT IN (2.0, 3.0, 4.0)
OR Grade_Pt IS NULL ;
```



This is a great technique to look for a NULL when using a NOT IN List.

An IN List with the Keyword ANY

Student_Table

<u>Student_ID</u>	<u>Last_Name</u>	<u>First_Name</u>	<u>Class_Code</u>	<u>Grade_Pt</u>
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

```
SELECT *
FROM   Student_Table
WHERE  Grade_Pt = ANY (2.0, 3.0, 4.0);
```



This is the same thing as using an IN. It's just another way of writing your SQL.

A NOT IN List with the Keywords NOT = ALL

Student_Table

Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

```
SELECT *
FROM   Student_Table
WHERE  Grade_Pt NOT = ALL (2.0, 3.0, 4.0);
```



This is an other way of doing a NOT IN. Notice the NOT = ALL and then the list.

BETWEEN is Inclusive

Student_Table

Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

```
SELECT *
FROM Student_Table
WHERE Grade_Pt BETWEEN 2.0 AND 4.0 ;
```

This is a BETWEEN. What this allows you to do is see if a column falls in a range. It is **inclusive** meaning that in our example, we will be getting the rows that also have a 2.0 and 4.0 in their column!

BETWEEN works for Character Data

Student_Table				
Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

```
SELECT *
FROM   Student_Table
WHERE  Last_Name BETWEEN 'L' AND 'LZ';
```

The BETWEEN isn't just used with numbers. You can look to see if words falls between certain letters.

LIKE uses Wildcards Percent ‘%’ and Underscore ‘_’

Student_Table

<u>Student_ID</u>	<u>Last_Name</u>	<u>First_Name</u>	<u>Class_Code</u>	<u>Grade_Pt</u>
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

```
SELECT *
FROM Student_Table
WHERE Last_Name LIKE 'SM%' ;
```

The wildcard percentage sign (%) is a wildcard for any number of characters. We are looking for anyone whose name starts with SM! In this example, the only row that would come back is ‘Smith’.

LIKE command Underscore is Wildcard for one Character

Student_Table				
Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

```
SELECT      *
FROM        Student_Table
WHERE       Last_Name LIKE '_a%';
```



The second wild card is an ‘_’ (underscore). An underscore represents a one character wildcard. Our search finds anyone with an ‘a’ in the second letter of their last name.

LIKE ALL means ALL conditions must be Met

Student_Table				
Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

```
SELECT *
FROM Student_Table
WHERE Last_Name LIKE ALL ('%S%', '%M%');
```

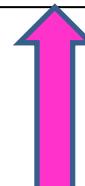
What this syntax is looking for is any row that has a `Last_Name` with an ‘S’ **AND** an ‘M’ in it. It isn’t looking for these in any order. As long as the `Last_Name` has an ‘S’ and an ‘M’ somewhere, it’ll come back.

LIKE ANY means ANY of the Conditions can be Met

Student_Table

Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

```
SELECT *
FROM Student_Table
WHERE Last_Name LIKE ANY ('%S%', '%M%');
```



The word ANY means either an 'S' OR an 'M' in the Last_Name, in any order.

IN ANSI Transaction Mode Case Matters

Student_Table

<u>Student_ID</u>	<u>Last_Name</u>	<u>First_Name</u>	<u>Class_Code</u>	<u>Grade_Pt</u>
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

/* This query is in **ANSI** Transaction mode */

```
SELECT *  
FROM Student_Table  
WHERE Last_Name LIKE ALL ('%S%', '%m%');
```

Capitol S

Small m



When in **ANSI** Transaction Mode, the system is **CASE SENSITIVE**, but it is not case sensitive in Teradata mode, also referred to as **BTET** for Begin and End Transaction.

In Teradata Transaction Mode Case Doesn't Matter

Student_Table

<u>Student_ID</u>	<u>Last_Name</u>	<u>First_Name</u>	<u>Class_Code</u>	<u>Grade_Pt</u>
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

/* This query is in **Teradata (BT/ET)** Transaction mode */

```
SELECT *
FROM   Student_Table
WHERE  Last_Name LIKE ALL ('%S%', '%m%');
```

Capitol S

Small m



LIKE Command Works Differently on Char Vs Varchar

Student_Table				
Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

/* First_Name has a Data Type of VARCHAR (20) */

```
SELECT *
FROM Student_Table
WHERE First_Name LIKE '%y';
```

It is important that you know the data type of the column you are using with your LIKE command. VARCHAR and CHAR data differ slightly.

Troubleshooting LIKE Command on Character Data

Student_Table				
Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

/* Last_Name has a Data Type of CHAR (20) */

```
SELECT *
FROM Student_Table
WHERE Last_Name LIKE '%on';
```

This is a CHAR(20) data type. That means that any words under 20 characters will pad spaces behind them until they reach 20 characters. You will not get any rows back from this example because technically, no row ends in an 'N', but instead ends in a space.

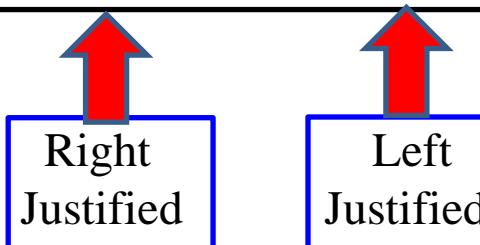
Quiz – What Data is Left Justified and What is Right?

```
SELECT *
FROM   Sample_Table
WHERE  Col1 IS NULL
AND    Col2 IS NULL ;
```

Answer Set

<u>Col1</u>	<u>Col2</u>
?	?

Right Justified Left Justified

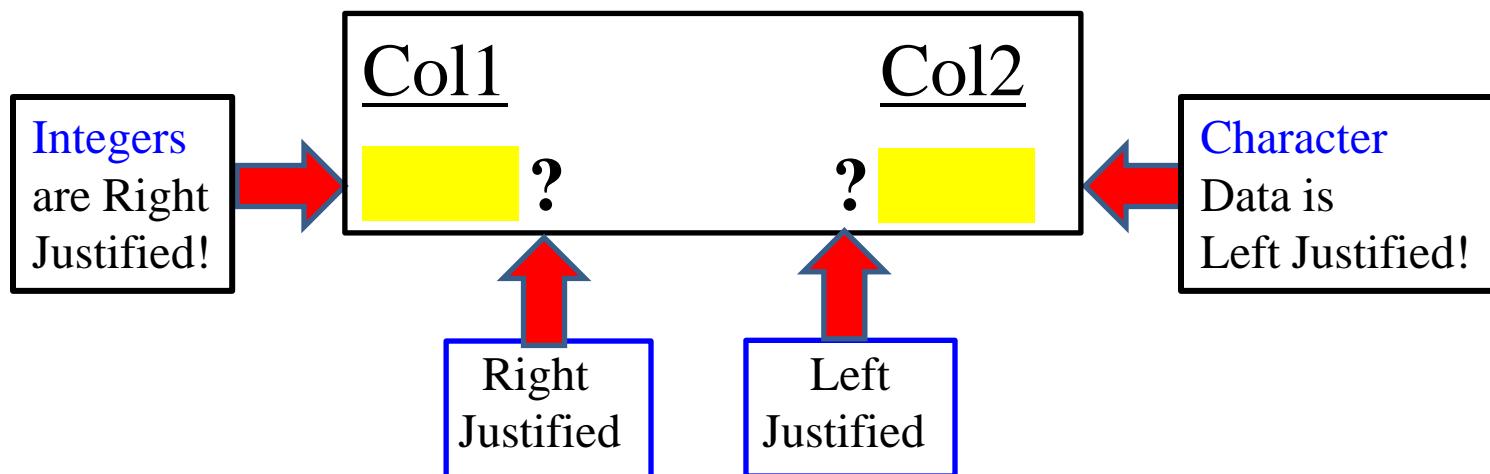


Which Column from the Answer Set could have a DATA TYPE of INTEGER and which could have Character Data?

Numbers are Right Justified and Character Data is Left

```
SELECT *
FROM   Sample_Table
WHERE  Col1 IS NULL
AND    Col2 IS NULL ;
```

Answer Set



All Integers will start from the right and move left. Thus Col1 was defined during the table create statement to hold an INTEGER. Next page shows a clear example.

Answer – What Data is Left Justified and What is Right?

```
SELECT Employee_No, First_Name  
FROM   Employee_Table  
WHERE  Employee_No = 2000000;
```

Answer Set

Employee_No	First_Name
2000000	Squiggy

Integers
are Right
Justified!

Character
Data is
Left Justified!

All Integers will start from the right and move left. All Character data will start from the left and move to the right.

An Example of Data with Left and Right Justification

```
SELECT Student_ID, Last_Name  
FROM    Student_Table ;
```

Integers
are Right
Justified!

Student_ID	Last_Name
423400	Larkins
125634	Hanson
280023	McRoberts
260000	Johnson
231222	Wilson
234121	Thomas
324652	Delaney
123250	Phillips
322133	Bond
333450	Smith

Character
Data is
Left Justified!

This is how a standard result set will look. Notice that the integer type in Student_ID starts from the right and goes left. Character data type in Last_Name moves left to right like we are use to seeing while reading English.

A Visual of CHARACTER Data vs. VARCHAR Data

Character Data on Disk

Last_Name as a Char(20)

Jones	□	□	□	□	□	□	□	□	□	□	□	□	□	□	← Spaces
Hanson	□	□	□	□	□	□	□	□	□	□	□	□	□	□	
McRoberts	□	□	□	□	□	□	□	□	□	□	□	□	□	□	
Johnson	□	□	□	□	□	□	□	□	□	□	□	□	□	□	

Varchar Data on Disk

Last_Name as a Varchar(20)

2-byte
VLI
Variable
Length
Indicator



0 5 Jones

0 6 Hanson

0 9 McRoberts

0 7 Johnson

Character data pads spaces to the right and Varchar uses a 2-byte VLI instead.

Use the TRIM command to remove spaces on CHAR Data

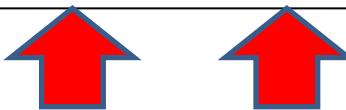
Character Data on Disk

Last_Name as a Char(20)

Jones	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	← Spaces
Hanson	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	
McRoberts	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	
Johnson	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	

/* Last_Name has a Data Type of CHAR (20) */

```
SELECT *
FROM Student_Table
WHERE TRIM (Last_Name) LIKE '%on';
```



By using the TRIM command on the Last_Name column, you are able to trim off any spaces from the end.

TRIM Eliminates Leading and Trailing Spaces

Student_Table

<u>Student_ID</u>	<u>Last_Name</u>	<u>First_Name</u>	<u>Class_Code</u>	<u>Grade_Pt</u>
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

/* Last_Name has a Data Type of CHAR (20) */

```
SELECT      *
FROM        Student_Table
WHERE       TRIM(Last_Name) LIKE '%n';
```

Once we use the TRIM on Last_Name we have eliminated any spaces at the end so now we are set to bring back anyone with a Last_Name that truly ends in 'n'!

Escape Character in the LIKE Command changes Wildcards

Student_Table				
Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00
999999	T_	S%	FR	1.90

/* We just pretended to add a new row to the Student_Table */

/* Can you use the **LIKE** command to find **S%** above? */

Here you will have to utilize a Wildcard Escape Character. Turn the page for more.

Escape Characters Turn off Wildcards in the LIKE Command

Student_Table				
Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00
999999	T_	S%	FR	1.90

/* Can you use the **LIKE** command to find **S%** above? */

```
SELECT      *
FROM        Student_Table
WHERE       First_Name LIKE 'S@%' Escape '@';
```

We can pick our Escape character and we have chosen the @ sign. This turns the wildcard off for 1 character so we find 'S%', without bringing back Stanley or Susie.

Quiz – Turn off that Wildcard

Student_Table				
Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00
999999	T_	S%	FR	1.90

/* Can you use the **LIKE** command to find the
Last_Name of T_ (pronounced Tunderscore!) */

This is a little trickier than you might think so be on your toes.... And get a haircut!

ANSWER – To Find that Wildcard

Student_Table

<u>Student_ID</u>	<u>Last_Name</u>	<u>First_Name</u>	<u>Class_Code</u>	<u>Grade_Pt</u>
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00
999999	T_	S%	FR	1.90

/* Can you use the **LIKE** command to find the Last_Name of **T_** (pronounced Tunderscore!) */

```
SELECT * FROM Student_Table  
WHERE TRIM>Last_Name) LIKE 'T@_' Escape '@';
```

You didn't really need to get a full haircut, but just a TRIM Command and the Escape!

Chapter 13

Distinct Vs Group By

“All things will be clear and **distinct** to the man who does not hurry; haste is blind and improvident.”

- Titus Livius

Table of Contents Chapter 13 - Distinct Vs Group By

- [The Distinct Command](#)
- [Distinct vs. GROUP BY](#)
- [Rules of Thumb for DISTINCT Vs GROUP BY](#)
- [Quiz – How many rows come back from the Distinct?](#)
- [Answer – How many rows come back from the Distinct?](#)

The Distinct Command

Student_Table

Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00



```
SELECT Distinct Class_Code  
FROM Student_Table  
ORDER BY 1;
```

Class_Code
?
FR
SO
JR
SR

DISTINCT eliminates duplicates from returning in the Answer Set.

Distinct vs. GROUP BY

Student_Table

Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

SELECT **Distinct** Class_Code
FROM Student_Table
ORDER BY 1;

SELECT Class_Code
FROM Student_Table
GROUP BY 1 ←
ORDER BY 1;

Class_Code
?
FR
SO
JR
SR

Distinct and GROUP BY in the two examples return the same answer set.

Rules of Thumb for DISTINCT Vs GROUP BY



```
SELECT Distinct Class_Code  
FROM Student_Table  
ORDER BY 1;
```

```
SELECT Class_Code  
FROM Student_Table  
GROUP BY 1  
ORDER BY 1;
```



Rules for Distinct Vs. GROUP BY

- (1) Many Duplicates – use GROUP BY
- (2) Few Duplicates – use DISTINCT
- (3) Space Exceeded - use GROUP BY

Distinct and GROUP BY work similarly, but utilize both for different situations.

Quiz – How many rows come back from the Distinct?

Student_Table

Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00



```
SELECT      Distinct Class_Code, Grade_Pt  
FROM        Student_Table  
ORDER BY    Class_Code, Grade_Pt;
```

How many rows will come back from the above SQL?

Answer – How many rows come back from the Distinct?

```
SELECT      Distinct Class_Code, Grade_Pt  
FROM        Student_Table  
ORDER BY    Class_Code, Grade_Pt;
```

<u>Class_Code</u>	<u>Grade_Pt</u>
?	?
FR	0.00
FR	2.88
FR	4.00
JR	1.90
JR	3.95
SO	2.00
SO	3.80
SR	3.00
SR	3.35

No Rows have the exact same Class_Code and Grade_Pt. Each row is Distinct!

How many rows will come back from the above SQL? 10. All rows came back. Why?
Because there are **no exact duplicates** that contain a **duplicate Class_Code** and **Duplicate Grade_Pt** combined. Each row in the SELECT list is distinct.

Chapter 14

The TOP Command

“But there are advantages to being elected President. The day after I was elected, I had my high school grades classified Top Secret. “

-Ronald Reagan

Table of Contents Chapter 14 - The TOP Command

- [TOP Command](#)
- [TOP Command is brilliant when ORDER BY is Used!](#)
- [The TOP Command WITH TIES](#)
- [How the TOP Command WITH TIES Decides](#)
- [The TOP Command will NOT work with Certain Commands](#)

TOP Command

Student_Table				
<u>Student_ID</u>	<u>Last_Name</u>	<u>First_Name</u>	<u>Class_Code</u>	<u>Grade_Pt</u>
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00



```
SELECT TOP 3 Last_Name,  
        Class_Code,  
        Grade_Pt  
FROM   Student_Table ;
```

This is the TOP command. By putting TOP 3 before the column names, you are telling your system that you only want to see the **top three rows** on your report. This example brings back **3 random rows**. That is not what makes the TOP command great. It becomes great when you **ORDER BY** the data.

TOP Command is brilliant when ORDER BY is Used!

Student_Table

Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

```
SELECT TOP 3 Last_Name,  
          Class_Code,  
          Grade_Pt  
FROM      Student_Table  
ORDER BY Grade_Pt DESC ;
```

Last_Name	Class_Code	Grade_Pt
Thomas	FR	4.00
Bond	JR	3.95
Wilson	SO	3.80



It is incredibly important to use an ORDER BY statement with your TOP Command. By using an Order By, the TOP command becomes GREAT!

The TOP Command WITH TIES

Student_Table

Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

```
SELECT TOP 2 WITH TIES  
    Last_Name, Class_Code, Grade_Pt  
FROM    Student_Table  
ORDER BY Class_Code ;
```

Last_Name	Class_Code	Grade_Pt
Johnson	?	?
Larkins	FR	0.00
Thomas	FR	4.00
Hanson	FR	2.88

By using the TOP WITH TIES Command, this will bring in the TOP amount along with ANY ties. So while you might only ask for the top 2 with ties, you might get 4 rows back. Why did 4 rows return here? Which row came back first?

How the TOP Command WITH TIES Decides

Student_Table

Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
423400	Larkins	Michael	FR 2nd	0.00
125634	Hanson	Henry	FR Tie	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	? 1st	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR Tie	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

```
SELECT TOP 2 WITH TIES
    Last_Name, Class_Code, Grade_Pt
FROM Student_Table
ORDER BY Class_Code ;
```

Last_Name	Class_Code	Grade_Pt
Johnson	?	?
Larkins	FR	0.00
Thomas	FR	4.00
Hanson	FR	2.88

Why did 4 rows return here? Which row came back first? Four rows returned with the first row coming back as a **NULL** for **Class_Code**. Then the next row returned was one of the Freshman. There were two other Freshman that tie. All 'FR' come back in a tie!

The TOP Command will NOT work with Certain Commands

Student_Table				
<u>Student_ID</u>	<u>Last_Name</u>	<u>First_Name</u>	<u>Class_Code</u>	<u>Grade_Pt</u>
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

```
SELECT TOP 3 *
FROM    Student_Table ;
```

Because of the location of TOP within the SELECT and the elimination of some of the rows, TOP is NOT compatible with the following SQL Constructs.

- * DISTINCT and QUALIFY
- * WITH and WITH BY
- * SAMPLE

Chapter 15

Review

“But there are advantages to being elected President. The day after I was elected, I had my high school grades classified Top Secret. “

-Ronald Reagan

Table of Contents Chapter 15 - Review

- [Testing Your Knowledge 1](#)
- [Testing Your Knowledge 2](#)
- [Testing Your Knowledge 3](#)
- [Testing Your Knowledge 4](#)
- [Testing Your Knowledge 5](#)
- [Testing Your Knowledge 6](#)
- [Testing Your Knowledge 7](#)

Testing Your Knowledge 1

Student_Table

Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

SELECT everything (all columns) from the Student_Table.

Testing Your Knowledge 2

Student_Table

<u>Student_ID</u>	<u>Last_Name</u>	<u>First_Name</u>	<u>Class_Code</u>	<u>Grade_Pt</u>
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

Now you should **SELECT** the **Student_ID**, **Last_Name**, **First_Name**, and **Class_Code** from the **Student_Table**.

Testing Your Knowledge 3

Student_Table

<u>Student_ID</u>	<u>Last_Name</u>	<u>First_Name</u>	<u>Class_Code</u>	<u>Grade_Pt</u>
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

Now you should SELECT the Student_ID, Last_Name, First_Name, and Class_Code from the Student_Table. Order By Class_Code.

Testing Your Knowledge 4

Student_Table

<u>Student_ID</u>	<u>Last_Name</u>	<u>First_Name</u>	<u>Class_Code</u>	<u>Grade_Pt</u>
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

Now you should SELECT the Student_ID, Last_Name, First_Name, and Class_Code from the Student_Table. Only show the rows that have the Class_Code FR, JR, SR. Order By Class_Code.

Testing Your Knowledge 5

Student_Table

<u>Student_ID</u>	<u>Last_Name</u>	<u>First_Name</u>	<u>Class_Code</u>	<u>Grade_Pt</u>
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

Now you should SELECT the Student_ID, Last_Name, First_Name, and Class_Code from the Student_Table. Only show the rows that have the Class_Code FR, JR, SR and if the First_Name ends in a 'y'. Order By Class_Code.

Testing Your Knowledge 6

Student_Table

<u>Student_ID</u>	<u>Last_Name</u>	<u>First_Name</u>	<u>Class_Code</u>	<u>Grade_Pt</u>
423400	Larkins	Michael	FR	0.00
125634	Hanson	Henry	FR	2.88
280023	McRoberts	Richard	JR	1.90
260000	Johnson	Stanley	?	?
231222	Wilson	Susie	SO	3.80
234121	Thomas	Wendy	FR	4.00
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
322133	Bond	Jimmy	JR	3.95
333450	Smith	Andy	SO	2.00

Now you should SELECT the Student_ID, Last_Name, First_Name, and Class_Code from the Student_Table. Only show the rows that have the Class_Code FR, JR, SR and if the First_Name ends in a ‘y’ AND if their Last_Name ends in ‘y’. Order By Class_Code.

Testing Your Knowledge 7

Answer Set

<u>Student_ID</u>	<u>Last_Name</u>	<u>First_Name</u>	<u>Class_Code</u>
324652	Delaney	Danny	SR

Select Student_ID, Last_Name, First_Name, Class_Code
From Student_Table

Where Class_Code In ('FR','JR', 'SR')

And First_Name LIKE '%y'

And Trim>Last_Name) LIKE '%y'

Order By Class_Code

This should be your final answer set. The query under it should be approximately what you wrote to attain such an answer set. How'd you do?

Chapter 16

HELP and SHOW

The whole secret of existence is to have no fear. Never fear what will become of you, depend on no one. Only the moment you reject all help are you freed.

- Buddha

Table of Contents Chapter 16 - HELP and SHOW

- [Determining the Release of your Teradata System](#)
- [Basic HELP Commands](#)
- [Other HELP Commands](#)
- [HELP DATABASE](#)
- [HELP USER](#)
- [HELP Table](#)
- [Adding a Comment to a Table](#)
- [Adding a Comment to a View](#)
- [SELECT SESSION](#)
- [USER Information Functions](#)
- [HELP SESSION](#)
- [HELP SQL](#)
- [A HELP SQL Example](#)
- [Show Commands](#)
- [SHOW Table command for Table DDL](#)
- [SHOW View command for View Create Statement](#)
- [SHOW Macro command for Macro Create Statement](#)
- [SHOW Trigger command for Trigger Create Statement](#)

Determining the Release of your Teradata System

```
SELECT * FROM DBC.DBCINFO;
```

InfoKey	InfoData
RELEASE	13.00.00.12
VERSION	13.00.00.12
LANGUAGE SUPPORT MODE	Standard

The above query pulls information from the Data Dictionary in USER DBC. Some companies don't allow users to see this information, but if you have the access rights then you can run the above query.

Basic HELP Commands

HELP DATABASE <database-name> ;	Displays the names of all the tables (T), views (V), macros (M), and triggers (G) stored in a database and user-written table comments
HELP USER <user-name> ;	Displays the names of all the tables (T), views (V), macros (M), and triggers (G) stored in a user area and user-written table comments
HELP TABLE <table-name> ;	Displays the column names, type identifier, and any user-written comments on the columns within a table.
HELP VOLATILE TABLE ;	Displays the names of all Volatile temporary tables active for the user session.
HELP VIEW <view-name> ;	Displays the column names, type identifier, and any user-written comments on the columns within a view.
HELP MACRO <macro-name> ;	Displays the characteristics of parameters passed to it at execution time.
HELP PROCEDURE <procedure-name> ;	Displays the characteristics of parameters passed to it at execution time.
HELP TRIGGER <trigger-name> ;	Displays details created for a trigger, like action time and sequence.
HELP COLUMN <table-name>.* ; HELP COLUMN <view-name>.* ; HELP COLUMN <table-name>.<column-name>, .. ;	Displays detail data describing the column level characteristics.

Other HELP Commands

HELP INDEX <table-name> ;	Displays the indexes and their characteristics like unique or non-unique and the column or columns involved in the index. This data is used by the Optimizer to create a plan for SQL.
HELP STATISTICS <table-name> ;	Displays values associated with the data demographics collected on the table. This data is used by the Optimizer to create a plan for SQL.
HELP CONSTRAINT <table-name>. <constraint-name> ;	Displays the checks to be made on the data when it is inserted or updated and the columns are involved.
HELP SESSION;	Displays the user name, account name, logon date and time, current database name, collation code set and character set being used, transaction semantics, time zone and character set data.
HELP 'SQL';	Displays a list of available SQL commands and functions.
HELP 'SQL <command>';	Displays the basic syntax and options for the actual SQL command inserted in place of the <command> .
HELP 'SPL';	Displays a list of available SPL commands.
HELP 'SPL <command>'; >, .. ;	Displays the basic syntax and options for the actual SPL command inserted in place of the <command> .

The above chart shows HELP commands for information on database tables and sessions, as well as SQL and SPL commands:

HELP DATABASE

HELP DATABASE SQL_Class ;

Table/View/Macro name	Kind	Comment
Tstamp_Macro	M	?
Subscribers	T	?
Student_Table	T	?
Student_Course_Table	T	?
Stats_Table	T	?
Services	T	?
Sales_Table	T	?
Providers	T	?
Order_Table	T	?
Names_View	V	?
Job_Table	T	?
Hierarchy_Join_Index	I	?
Employee_Table	T	?

A complete list of KIND

- A Aggregate function
- B Combined aggregate/analytical function
- E External stored procedure
- F Function
- G Trigger
- H Method
- I Join index
- J Journal table
- M Macro
- N Hash index
- P Procedure
- Q Queue table
- R Table function
- S Ordered analytical function
- T Table
- U UDT
- V View
- X Authorization

Not all columns in the HELP Database SQL_Class were shown, just the important ones.
The HELP DATABASE command will show you the objects in your database.

HELP USER

HELP USER DBC;

Table/View/Macro name	Kind
AccessLog	V
AccessLogV	V
AccessRights	T
AccLogRule	M
AccLogRules	V
AccLogRulesV	V
AccLogRuleTbl	T
AccLogRuleTbl_TD12	T
AccLogTbl	T
AccLogTbl_TD12	T
AccLogTbl_V2R6	T
AccountInfo	V
AccountInfoV	V

A complete list of KIND

- A Aggregate function
- B Combined aggregate and analytical function
- E External stored procedure
- F Function
- G Trigger
- H Method
- I [Join index](#)
- J Journal table
- M Macro
- N Hash index
- P Procedure
- Q Queue table
- R Table function
- S Ordered analytical function
- T Table
- U UDT
- V View
- X Authorization

Not all columns in the HELP USER were shown, just the important ones. The HELP USER command will show you the objects in a USER. USER is a keyword!

HELP TABLE

HELP Table SQL_Class.Employee_Table ;

Column Name	Type	Comment	Nullable	Format	Title	MaxLength
Employee_No	I	?	Y	-(10)9	?	4
Dept_No	I2	?	Y	-(5)9	?	2
Last_Name	CF	?	Y	X(20)	?	20
First_Name	CV	?	Y	X(12)	?	12
Salary	D	?	Y	-----.99	?	4



- I = Integer
- I2 = Smallint
- CF = Character Fixed
- CV = Character Variable
- D = Decimal

Not all columns in the HELP TABLE were shown, just the important ones. The HELP TABLE command will show you information about a table.

Adding a Comment to a Table

```
COMMENT ON TABLE SQL_Class.Stats_Table 'This table holds Stats' ;
```

```
Help Database SQL_Class ;
```

Table/View/Macro name	Kind	Comment
Tstamp_Macro	M	?
Subscribers	T	?
Student_Table	T	?
Student_Course_Table	T	?
Stats_Table	T	This table holds Stats
Services	T	?
Sales_Table	T	?
Providers	T	?
Order_Table	T	?
Names_View	V	?

The above syntax will place a comment on the table.

Adding a Comment to a View

```
COMMENT ON View SQL_VIEWS.Employee_V 'No Salary is shown';
```

```
Help Database SQL_VIEWS ;
```

Table/View/Macro name	Kind	Comment
Addresses_v	V	?
Claims_v	V	?
Course_v	V	?
Customer_v	V	?
Department_v	V	?
Employee_v	V	No Salary is shown
Emp_Job_v	V	?
Hierarchy_v	V	?
Job_v	V	?
Names_v	V	?

The above syntax will place a comment on the View.

SELECT SESSION

```
SELECT Session;
```

Session
8692

The SELECT Session command will show you the SESSION Number you received when you logged on to Teradata. The Parsing Engine assigned to manage your session tracks you by this session number.

USER Information Functions

```
SELECT Account  
      ,Database  
      ,Session  
      ,USER
```

Account	Database	Session	USER
DBC	SQL_CLASS	8838	DBC

The Teradata RDBMS (Relational DataBase Management System) has incorporated into it functions that provide data regarding a user who has performed a logon connection to the system. The following functions make that data available to a user for display or storage. Notice the keyword USER.

HELP SESSION

Help Session;

User Name	Account Name	Logon Date	Logon Time	Current Database	Collation	Char Set	Transaction Semantics	Current Dateform	Session Time Zone
DBC	DBC	12/06/17	15:55:39	SQL_CLASS	ASCII	ASCII	Teradata	IntegerDate	00:00

The HELP Session command will show information about your SESSION. Not all columns were shown above, just the most important ones.

HELP SQL

Help 'SQL';

DBS SQL Commands

ABORT
BEGIN TRANSACTION
COMMIT
CREATE INDEX
CREATE USER
DELETE

ALTER TABLE
CHECKPOINT
COMMENT
CREATE MACRO
CREATE VIEW
DELETE DATABASE

BEGIN LOGGING
COLLECT STATISTICS
CREATE DATABASE
CREATE TABLE
DATABASE
DELETE USER

DBS SQL Functions

ABS
CHARACTERS
COUNT
CSUM
FORMAT
HASHBKAMP

ADD_MONTHS
CAST
CORR
EXP
INDEX
HASHBUCKET

AVERAGE
CHAR2HEXINT
COVAR_POP
EXTRACT
HASHAMP
HASHROW

The HELP 'SQL' will show you a list of SQL Commands and another list of Functions supported by Teradata. Not all commands or functions are shown above.

A HELP SQL Example

```
Help 'SQL CSUM';
```

Syntax

`CSUM(value_expression, sort_expression_list)`

Computes a running or cumulative total of a column value.

The HELP 'SQL CSUM' will show you the syntax for the CSUM command. This HELP command will work for each and every SQL Statement.

Show Commands

SHOW TABLE <table-name> ;	Displays the CREATE TABLE statement needed to create this table.
SHOW VIEW <view-name> ;	Displays the CREATE VIEW statement needed to create this view.
SHOW MACRO <macro-name> ;	Displays the CREATE MACRO statement needed to create this macro.
SHOW TRIGGER <trigger-name> ;	Displays the CREATE TRIGGER statement needed to create this trigger.
SHOW PROCEDURE <procedure-name> ;	Displays the CREATE PROCEDURE statement needed to create this stored procedure.
SHOW <SQL-statement> ;, .. ;	Displays the CREATE TABLE statements for all tables/views referenced by the SQL statement .

SHOW Table command for Table DDL

```
SHOW Table SQL_Class.Employee_Table ;
```

```
CREATE SET TABLE SQL_CLASS.Employee_Table ,NO FALBACK ,
NO BEFORE JOURNAL,
NO AFTER JOURNAL,
CHECKSUM = DEFAULT
(
Employee_No INTEGER,
Dept_No SMALLINT,
Last_Name CHAR(20) CHARACTER SET LATIN NOT CASESPECIFIC,
First_Name VARCHAR(12) CHARACTER SET LATIN NOT CASESPECIFIC,
Salary DECIMAL(8,2))
UNIQUE PRIMARY INDEX ( Employee_No )
INDEX ( Last_Name )
INDEX ( Dept_No );
```

The above syntax will show the Table DDL (Data Definition Language). It is the table CREATE Statement plus any additional changes, such as adding an Index.

SHOW View command for View Create Statement

```
SHOW View SQL_VIEWS.Employee_v ;
```

```
CREATE VIEW SQL_VIEWS.Employee_v (Emp_No, Lname, Fname, Sal, Dept) AS  
SELECT Employee_No,  
       Last_Name,  
       First_Name,  
       Salary,  
       Dept_No  
FROM SQL_CLASS.Employee_Table;
```

The above syntax will show the View's CREATE Statement.

SHOW Macro command for Macro Create Statement

```
SHOW MACRO MY_Mac ;
```

```
CREATE MACRO MJL01.MY_Mac (INPARM1 INTEGER, INPARM2 CHAR(10))
AS
(SELECT DEPT, DAY_OF_WEEK, AVG(SAL)
FROM SYS_CALENDAR.CALENDAR SC, MYTABLE
WHERE CALENDAR_DATE = :INPARM2 (DATE, FORMAT 'YYYYMMDD')
AND DEPT = :INPARM1
GROUP BY 1,2; );
```

The above syntax will show the Macro's CREATE Statement.

SHOW Trigger command for Trigger Create Statement

```
SHOW TRIGGER AVG_SAL_T ;
```

```
CREATE TRIGGER MJL.AVG_SAL_T
AFTER UPDATE OF (SALARY) ON MJL.EMPLOYEE
REFERENCING OLD AS OLDROW
NEW AS NEWROW
FOR EACH ROW
WHEN (NEWROW.SALARY >
(SELECT AVG(BUDGET) * .10 (DECIMAL(10,2))
 FROM MJL01.DEPARTMENT ) )
(INSERT INTO MJL01.GREATER_10_PERCENT
(EMP_NUM ,SAL_DATE ,OLDSAL ,NEWSAL ,PERC_OF_BUDGET )
VALUES (NEWROW.EMP_NBR ,CURRENT_DATE ,OLDROW.SALARY
,NEWROW.SALARY); );
```

The above syntax will show the Trigger's CREATE Statement.

Chapter 17

Aggregation Function

“Teradata climbed the Aggregate Mountain and delivered
a better way to Sum It.”

Tera-Tom Coffing

Table of Contents Chapter 17 - Aggregation Function

- [Quiz – You calculate the Answer Set in your own Mind](#)
- [Answer – You calculate the Answer Set in your own Mind](#)
- [The 3 Rules of Aggregation](#)
- [There are Five Aggregates](#)
- [Quiz – How many rows come back?](#)
- [Troubleshooting Aggregates](#)
- [GROUP BY when Aggregates and Normal Columns Mix](#)
- [GROUP BY Delivers one row per Group](#)
- [GROUP BY Dept No or GROUP BY 1 the same thing](#)
- [Limiting Rows and Improving Performance with WHERE](#)
- [WHERE Clause in Aggregation limits unneeded Calculations](#)
- [Keyword HAVING tests Aggregates after they are Totaled](#)
- [Keyword HAVING is like an Extra WHERE Clause for Totals](#)
- [Three types of Advanced Grouping](#)
- [GROUP BY Grouping Sets](#)
- [GROUP BY Rollup](#)
- [GROUP BY Rollup Result Set](#)
- [GROUP BY Cube](#)
- [GROUP BY CUBE Result Set](#)
- [Testing Your Knowledge](#)
- [Final Answer to Test Your Knowledge on Aggregates](#)

Quiz – You calculate the Answer Set in your own Mind

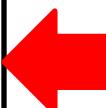
Aggregation_Table

Employee_No	Salary
423400	100000.00
423401	100000.00
423402	NULL

```
SELECT AVG(Salary) as "AVG"  
      ,Count(Salary) as SalCnt  
      ,Count(*)      as RowCnt  
FROM Aggregation_Table ;
```

AVG SalCnt RowCnt

Please fill in
the values you
think will be
in the Answer.



What would the result set be from the above query? The next slide shows answers!

Answer – You calculate the Answer Set in your own Mind

Aggregation_Table

Employee_No	Salary
423400	100000.00
423401	100000.00
423402	NULL

```
SELECT AVG(Salary) as "AVG"  
      ,Count(Salary) as SalCnt  
      ,Count(*)      as RowCnt  
FROM Aggregation_Table ;
```

AVG	SalCnt	RowCnt
100000.00	2	3

Here are
all the
Correct
answers

Here are your answers!

The 3 Rules of Aggregation

Aggregation_Table

Employee_No	Salary
423400	100000.00
423401	100000.00
423402	NULL

```
SELECT AVG(Salary)
      ,Count(Salary)
      ,Count(*)
FROM   Aggregation_Table;
```

- 1) Aggregates Ignore **Null** Values.
- 2) Aggregates WANT to come back in **one** row.
- 3) You CAN'T mix Aggregates with **normal columns** unless you use a **GROUP BY**.

AVG(Salary) = \$100000.00

Count(Salary) = 2

Count(*) = 3

There are Five Aggregates

Aggregation_Table	
<u>Employee_No</u>	<u>Salary</u>
423400	100000.00
423401	100000.00
423402	NULL

There are **FIVE AGGREGATES** which are the following:

MIN – The Minimum Value.

MAX – The Maximum Value.

AVG – The Average of the Column Values.

SUM – The Sum Total of the Column Values.

COUNT – The Count of the Column Values.

Quiz – How many rows come back?

Employee_Table

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

Query 

```
SELECT MIN (Salary)
      ,MAX (Salary)
      ,SUM (Salary)
      ,AVG (Salary)
      ,Count(*)
FROM Employee_Table ;
```

How many rows will the above query produce in the result set? The answer is one.

Troubleshooting Aggregates

Employee_Table

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

SELECT Dept_No ←
,MIN (Salary)
,MAX (Salary)
,SUM (Salary)
,AVG (Salary)
,Count(*)
FROM Employee_Table ;

NON-Aggregate

How many rows will the above query produce in the result set? This query errors!

GROUP BY when Aggregates and Normal Columns Mix

Employee_Table

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

```
SELECT Dept_No  
      ,MIN (Salary)  
      ,MAX (Salary)  
      ,SUM (Salary)  
      ,AVG (Salary)  
      ,COUNT(*)  
FROM Employee_Table  
GROUP BY Dept_No  
Order by 1 ;
```

Group By
Needed

NON-Aggregate

GROUP BY Delivers one row per Group

```
SELECT Dept_No ,MIN (Salary)  
      ,MAX (Salary)  
      ,SUM (Salary)  
      ,AVG (Salary)  
      ,Count(*)  
FROM Employee_Table  
GROUP BY Dept_No  
Order by 1;
```

Group By
Needed

NON-Aggregate

Dept_No	Min(Salary)	Max(Salary)	Sum(Salary)	AVG(Salary)	Count(*)
?	32800.50	32800.50	32800.50	32800.50	1
10	64300.00	64300.00	64300.00	64300.00	1
100	48850.00	48850.00	48850.00	48850.00	1
200	41888.88	48000.00	89888.88	44944.44	2
300	40200.00	40200.00	40200.00	40200.00	1
400	36000.00	54500.00	145000.00	48333.33	3

Group By Dept_No command allow for the Aggregates to be calculated per Dept_No.

GROUP BY Dept_No or GROUP BY 1 the same thing

```
SELECT Dept_No  
      ,MIN (Salary)  
      ,MAX (Salary)  
      ,SUM (Salary)  
      ,AVG (Salary)  
      ,Count(*)  
FROM   Employee_Table  
GROUP BY Dept_No  
ORDER BY Dept_No;
```

Both Queries
are the same
to the system.

```
SELECT Dept_No  
      ,MIN (Salary)  
      ,MAX (Salary)  
      ,SUM (Salary)  
      ,AVG (Salary)  
      ,Count(*)  
FROM   Employee_Table  
GROUP BY 1  
ORDER BY 1;
```

Dept_No	Min(Salary)	Max(Salary)	Sum(Salary)	AVG(Salary)	Count(*)
?	32800.50	32800.50	32800.50	32800.50	1
10	64300.00	64300.00	64300.00	64300.00	1
100	48850.00	48850.00	48850.00	48850.00	1
200	41888.88	48000.00	89888.88	44944.44	2
300	40200.00	40200.00	40200.00	40200.00	1
400	36000.00	54500.00	145000.00	48333.33	3

Both queries above produce the same result. The GROUP BY allows you to either name the column or use the number in the SELECT list, just like the ORDER BY.

Limiting Rows and Improving Performance with WHERE

Employee_Table

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

```
SELECT Dept_No, MIN (Salary), MAX (Salary), SUM (Salary)
      , AVG (Salary) , COUNT(*)
FROM Employee_Table
WHERE Dept_No IN (200, 400)
GROUP BY Dept_No
Order by 1 ;
```

WHERE Clause acts
as a filter before any
Calculations are done

Will Dept_No 300 be calculated? Of course you know it will...NOT!

WHERE Clause in Aggregation limits unneeded Calculations

```
SELECT Dept_No, MIN (Salary), MAX (Salary), SUM (Salary)  
      , AVG (Salary) , COUNT(*)  
FROM   Employee_Table  
WHERE Dept_No IN (200, 400) ← WHERE Clause acts  
GROUP BY Dept_No           as a filter before any  
Order by 1 ;                Calculations are done
```

Dept_No	Min(Salary)	Max(Salary)	Sum(Salary)	AVG(Salary)	Count(*)
200	41888.88	48000.00	89888.88	44944.44	2
400	36000.00	54500.00	145000.00	48333.33	3

The system eliminates reading any other Dept_No's other than 200 and 400. This means that only Dept_No's of 200 and 400 will come off the disk to be calculated.

Keyword HAVING tests Aggregates after they are Totaled

```
SELECT Dept_No, MIN (Salary), MAX (Salary), SUM (Salary)  
      , AVG (Salary) , COUNT(*)  
FROM Employee_Table  
WHERE Dept_No in (200, 400)  
GROUP BY Dept_No  
HAVING Count(*) > 2 ;
```

HAVING Clause acts as a filter on all Aggregates after they are totaled.

Previous Answer Set

Dept_No	Min(Salary)	Max(Salary)	Sum(Salary)	Avg(Salary)	Count(*)
200	41888.88	48000.00	89888.88	44944.44	2
400	36000.00	54500.00	145000.00	48333.33	3

NEW Answer Set
???????????????

Can you calculate what the new Answer Set will be after the HAVING Clause is implemented.

The HAVING Clause only works on Aggregate Totals. The WHERE filters rows to be excluded from calculation, but the HAVING filters the Aggregate totals after the calculations, thus eliminating certain Aggregate totals.

Keyword HAVING is like an Extra WHERE Clause for Totals

```
SELECT Dept_No, MIN (Salary), MAX (Salary), SUM (Salary)  
      , AVG (Salary) , COUNT(*)  
FROM Employee_Table  
WHERE Dept_No in (200, 400)  
GROUP BY Dept_No  
HAVING Count(*) > 2 ;
```

HAVING Clause acts as a filter on all Aggregates after they are totaled.

Previous Answer Set

Dept_No	Min(Salary)	Max(Salary)	Sum(Salary)	AVG(Salary)	Count(*)
200	41888.88	48000.00	89888.88	44944.44	2
400	36000.00	54500.00	145000.00	48333.33	3

NEW Answer Set

Dept_No	Min(Salary)	Max(Salary)	Sum(Salary)	AVG(Salary)	Count(*)
400	36000.00	54500.00	145000.00	48333.33	3

The HAVING Clause only works on Aggregate Totals and only Count(*) > 2 can return.

Three types of Advanced Grouping

Sales_Table		
Product_ID	Sale_Date	Daily_Sales
1000	2000-09-28	48850.40
2000	2000-09-28	41888.88
3000	2000-09-28	61301.77
1000	2000-09-29	54500.22
2000	2000-09-29	48000.00
3000	2000-09-29	34509.13
1000	2000-09-30	36000.07
2000	2000-09-30	49850.03
3000	2000-09-30	43868.86
1000	2000-10-01	40200.43
2000	2000-10-01	54850.29
3000	2000-10-01	28000.00
1000	2000-10-02	32800.50
2000	2000-10-02	36021.93
3000	2000-10-02	19678.94

Not all rows
in this table
are displayed

GROUP BY **Grouping Sets**
GROUP BY **Rollup**
GROUP BY **Cube**

GROUP BY Grouping Sets

```
SELECT Product_ID  
      ,EXTRACT (MONTH FROM Sale_Date) AS MTH  
      ,EXTRACT (YEAR FROM Sale_Date) AS YR  
      ,SUM(Daily_Sales) AS SUM_Daily_Sales  
FROM Sales_Table  
GROUP BY GROUPING SETS(Product_ID, MTH, YR)  
ORDER BY Product_ID Desc, MTH Desc, YR Desc;
```

<u>Product ID</u>	<u>MTH</u>	<u>YR</u>	<u>SUM Daily Sales</u>	
3000	?	?	224587.82	Product 3000 sales
2000	?	?	306611.81	Product 2000 sales
1000	?	?	331204.72	Product 1000 sales
?	10	?	443634.99	October sales all years
?	09	?	418769.36	Sept. sales all years
?	?	2000	862404.35	Grand Total

GROUP BY GROUPING Sets above will show you what your Daily_Sales were for each Product_ID, for each month, and for each year.

GROUP BY Rollup

```
SELECT Product_ID  
      ,EXTRACT (MONTH FROM Sale_Date) AS MTH  
      ,EXTRACT (YEAR FROM Sale_Date) AS YR  
      ,SUM(Daily_Sales) AS SUM_Daily_Sales  
FROM Sales_Table  
GROUP BY Rollup (Product_ID, MTH, YR)  
ORDER BY Product_ID Desc, MTH Desc, YR Desc;
```

<u>Product_ID</u>	<u>MTH</u>	<u>YR</u>	<u>SUM_Daily_Sales</u>	
3000	10	2000	84908.06	October, 2000 sales
3000	10	?	84908.06	October sales all years
3000	09	2000	139679.76	September, 2000 sales
3000	09	?	139679.76	October sales all years
3000	?	?	224587.82	All Product 3000 sales
2000	10	2000	166872.90	
2000	10	?	166872.90	
2000	09	2000	139738.91	
				Grand Total at end

Not all 16 rows in this answer set could be displayed

GROUP BY ROLLUP displays what the Daily_Sales were for each Product_ID, for each distinct month, for each month per year and for each year, plus a grand total.

GROUP BY Rollup Result Set

<u>Product_ID</u>	<u>MTH</u>	<u>YR</u>	<u>SUM_Daily_Sales</u>	
3000	10	2000	84908.06	Prod 3000 October, 2000 sales
3000	10	?	84908.06	Prod 3000 October sales all years
3000	9	2000	139679.76	Prod 3000 September, 2000 sales
3000	9	?	139679.76	Prod 3000 September sales all years
3000	?	?	224587.82	All Product 3000 sales
2000	10	2000	166872.90	Prod 2000 October, 2000 sales
2000	10	?	166872.90	Prod 2000 October sales all years
2000	9	2000	139738.91	Prod 2000 September, 2000 sales
2000	9	?	139738.91	Prod 2000 September sales all years
2000	?	?	306611.81	All Product 2000 sales
1000	10	2000	191854.03	Prod 1000 October, 2000 sales
1000	10	?	191854.03	Prod 1000 October sales all years
1000	9	2000	139350.69	Prod 1000 September, 2000 sales
1000	9	?	139350.69	Prod 1000 September sales all years
1000	?	?	331204.72	All Product 1000 sales
?	?	?	862404.35	Grand Total at end

This is the full result set from the previous GROUP BY **ROLLUP** query.

GROUP BY Cube

```
SELECT Product_ID  
      ,EXTRACT (MONTH FROM Sale_Date) AS MTH  
      ,EXTRACT (YEAR FROM Sale_Date) AS YR  
      ,SUM(Daily_Sales) AS SUM_Daily_Sales  
  FROM Sales_Table  
 GROUP BY CUBE (Product_ID, MTH, YR)  
 ORDER BY Product_ID Desc, MTH Desc, YR Desc;
```

Product_ID	MTH	YR	SUM_Daily_Sales
3000	10	2000	84908.06
3000	10	?	84908.06
3000	9	2000	139679.76
3000	9	?	139679.76
3000	?	2000	224587.82
3000	?	?	224587.82
2000	10	2000	166872.90
2000	10	?	166872.90

Not all 24 rows in this answer set could be displayed

GROUP BY ROLLUP displays Daily_Sales were for each Product_ID, for each distinct month, for each month per year and for each year, plus a grand total.

GROUP BY CUBE Result Set

Product_ID	MTH	YR	SUM_Daily_Sales	
3000	10	2000	84908.06	Prod 3000 October, 2000 sales
3000	10	?	84908.06	Prod 3000 October sales all years
3000	9	2000	139679.76	Prod 3000 September, 2000 sales
3000	9	?	139679.76	Prod 3000 September sales all years
3000	?	2000	224587.82	Prod 3000 sales for the year 2000
3000	?	?	224587.82	Prod 3000 sales for all years
2000	10	2000	166872.90	Prod 2000 October, 2000 sales
2000	10	?	166872.90	Prod 2000 October sales all years
2000	9	2000	139738.91	Prod 2000 September, 2000 sales
2000	9	?	139738.91	Prod 2000 September sales all years
2000	?	2000	306611.81	Prod 2000 sales for the year 2000
2000	?	?	306611.81	Prod 2000 sales for all years
1000	10	2000	191854.03	Prod 1000 October, 2000 sales
1000	10	?	191854.03	Prod 1000 October sales all years
1000	9	2000	139350.69	Prod 1000 September, 2000 sales
1000	9	?	139350.69	Prod 1000 September sales all years
1000	?	2000	331204.72	Prod 1000 sales for the year 2000
1000	?	?	331204.72	Prod 1000 sales for all years
?	10	2000	443634.99	Total Sales for October, 2000
?	10	?	443634.99	Total Sales for all October dates
?	9	2000	418769.36	Total Sales for September, 2000
?	9	?	418769.36	Total Sales for all September dates
?	?	2000	862404.35	Total Sales for the year 2000
?	?	?	862404.35	Total Sales for all years totaled

Testing Your Knowledge

Employee_Table

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

First of all, SELECT the AVERAGE Salary from the Employee_Table.

Testing Your Knowledge

Employee_Table

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

Now , SELECT the AVERAGE Salary and the SUM of the Salary from the Employee_Table.

Testing Your Knowledge

Employee_Table

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

Now , SELECT the AVERAGE Salary and the SUM of the Salary from the Employee_Table but PER DEPARTMENT(Dept_No).

Testing Your Knowledge

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

After that, SELECT the AVERAGE Salary and the SUM of the Salary from the Employee_Table but PER DEPARTMENT(Dept_No). However, I only want to see the people from Department 200, 300, 400.

Testing Your Knowledge

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

After that, SELECT the AVERAGE Salary and the SUM of the Salary from the Employee_Table but PER DEPARTMENT(Dept_No). However, I only want to see Department 200, 300, 400 which has an AVERAGE Salary of over 43,000.

Final Answer to Test Your Knowledge on Aggregates

Answer Set		
<u>Dept_No</u>	<u>AVG(Salary)</u>	<u>Sum(Salary)</u>
200	44944.44	89888.88
400	48316.67	144950.00

```
Select Dept_No, AVG(Salary), SUM(Salary)  
From Employee_Table  
Where Dept_No IN (200, 300, 400)  
Group By Dept_No  
Having AVG(Salary) > 43000
```

This should be your final answer set. The query under it should be approximately what you wrote to attain such an answer set. How'd you do?

Chapter 18

Join Functions

“When spider webs unite they can tie up a lion”

- African Proverb

Table of Contents Chapter 18 - Join Functions

- [A two-table join using Non-ANSI Syntax](#)
- [A two-table join using Non-ANSI Syntax with Table Alias](#)
- [Aliases and Fully Qualifying Columns](#)
- [A two-table join using ANSI Syntax](#)
- [Both Queries have the same Results and Performance](#)
- [Quiz – Can You Finish the Join Syntax?](#)
- [Answer to Quiz – Can You Finish the Join Syntax?](#)
- [Quiz – Can You Find the Error?](#)
- [Answer to Quiz – Can You Find the Error?](#)
- [Quiz – Which rows from both tables Won’t Return?](#)
- [Answer to Quiz – Which rows from both tables Won’t Return?](#)
- [LEFT OUTER JOIN](#)
- [LEFT OUTER JOIN EXAMPLE](#)
- [RIGHT OUTER JOIN](#)
- [RIGHT OUTER JOIN EXAMPLE](#)
- [FULL OUTER JOIN](#)
- [FULL OUTER JOIN EXAMPLE](#)
- [INNER JOIN with Additional AND Clause](#)
- [ANSI INNER JOIN with Additional AND Clause](#)
- [ANSI INNER JOIN with Additional WHERE Clause](#)
- [OUTER JOIN with Additional AND Clause](#)
- [OUTER JOIN with Additional WHERE Clause](#)
- [OUTER JOIN with Additional AND Clause](#)
- [OUTER JOIN with Additional AND Clause Example](#)
- [Quiz – Why is this Considered an INNER JOIN?](#)
- [The DREADED Product Join](#)
- [The DREADED Product Join](#)
- [The Horrifying Cartesian Product Join](#)
- [The ANSI Cartesian Join will ERROR](#)
- [Quiz – Do these Joins Return the Same Answer Set?](#)

Continued on next page

Table of Contents Chapter 18 - Join Functions Continued

- [Answer – Do these Joins Return the Same Answer Set?](#)
- [The CROSS JOIN](#)
- [The CROSS JOIN Answer Set](#)
- [The Self Join](#)
- [The Self Join with ANSI Syntax](#)
- [Quiz – Will both queries bring back the same Answer Set?](#)
- [Answer – Will both queries bring back the same Answer Set?](#)
- [Quiz – Will both queries bring back the same Answer Set?](#)
- [Answer – Will both queries bring back the same Answer Set?](#)
- [How would you Join these two tables?](#)
- [How would you Join these two tables? You Can't....Yet!](#)
- [An Associative Table is a Bridge that Joins Two Tables](#)
- [Quiz – Can you Write the 3-Table Join?](#)
- [Answer to Quiz – Can you Write the 3-Table Join?](#)
- [Quiz – Can you Write the 3-Table Join to ANSI Syntax?](#)
- [Answer – Can you Write the 3-Table Join to ANSI Syntax?](#)
- [Quiz – Can you Place the ON Clauses at the End?](#)
- [Answer – Can you Place the ON Clauses at the End?](#)
- [The 5-Table Join – Logical Insurance Model](#)
- [The Nexus Query Chameleon](#)
- [The 5-Table Join ANSI SQL Created by Nexus](#)
- [The 5-Table Join With ON Clauses at END](#)
- [The Join Tab of Nexus](#)
- [The COLUMNS Tab of Nexus](#)

A two-table join using Non-ANSI Syntax

Customer_Table		Order_Table		
Customer_Number	Customer_Name	Order_Number	Customer_Number	Order_Total
11111111	Billy's Best Choice	123456	11111111	12347.53
31313131	Acme Products	123512	11111111	8005.91
31323134	ACE Consulting	123552	31323134	5111.47
57896883	XYZ Plumbing	123585	87323456	15231.62
87323456	Databases N-U	123777	57896883	23454.84

```
SELECT Customer_Table.Customer_Number, Customer_Name,  
       Order_Number, Order_Total  
FROM   Customer_Table,  
       Order_Table  
WHERE  Customer_Table.Customer_Number = Order_Table.Customer_Number ;
```

A Join combines columns on the report from more than one table. The example above joins the Customer_Table and the Order_Table together. The most complicated part of any join is the JOIN CONDITION. The JOIN CONDITION means what Column from each table is a match. In this case Customer_Number is a match that establishes the relationship so this join will happen on matching Customer_Number columns.

A two-table join using Non-ANSI Syntax with Table Alias

Customer_Table		Order_Table		
Customer_Number	Customer_Name	Order_Number	Customer_Number	Order_Total
11111111	Billy's Best Choice	123456	11111111	12347.53
31313131	Acme Products	123512	11111111	8005.91
31323134	ACE Consulting	123552	31323134	5111.47
57896883	XYZ Plumbing	123585	87323456	15231.62
87323456	Databases N-U	123777	57896883	23454.84

```
SELECT      Cust.Customer_Number, Customer_Name,  
            Order_Number, Order_Total  
FROM        Customer_Table as Cust,  
            Order_Table as ORD  
WHERE       Cust.Customer_Number = Ord.Customer_Number ;
```

A Join combines columns on the report from more than one table. The example above joins the Customer_Table and the Order_Table together. The most complicated part of any join is the JOIN CONDITION. The JOIN CONDITION means what Column from each table is a match. In this case Customer_Number is a match that establishes the relationship.

Aliases and Fully Qualifying Columns

Customer_Table		Order_Table		
Customer_Number	Customer_Name	Order_Number	Customer_Number	Order_Total
11111111	Billy's Best Choice	123456	11111111	12347.53
31313131	Acme Products	123512	11111111	8005.91
31323134	ACE Consulting	123552	31323134	5111.47
57896883	XYZ Plumbing	123585	87323456	15231.62
87323456	Databases N-U	123777	57896883	23454.84

```
SELECT Cust.Customer_Number,  
       Customer_Name, Order_Number,  
       Order_Total  
FROM   Customer_Table as Cust,  
       Order_Table as ORD  
WHERE  Cust.Customer_Number  
       = Ord.Customer_Number ;
```

The query is annotated with three colored arrows pointing from the right towards specific parts of the code:

- A red arrow points to the first occurrence of `Cust.Customer_Number`, with the text "Fully Qualified with CUST" above it.
- A green arrow points to the alias `CUST` in `Customer_Table as Cust`, with the text "CUST is now Table ALIAS" above it.
- A blue arrow points to the alias `ORD` in `Order_Table as ORD`, with the text "ORD in now Table ALIAS" above it.

Customer_Number is a column in both the Customer and Order Tables.
CUST.Customer_Number fully qualifies the column to specifically state we want the Customer_Number from the Customer_Table. That is why we ALIASED the table names, so we could fully qualify any columns in both tables, or we error!

A two-table join using ANSI Syntax

Customer_Table		Order_Table		
Customer_Number	Customer_Name	Order_Number	Customer_Number	Order_Total
11111111	Billy's Best Choice	123456	11111111	12347.53
31313131	Acme Products	123512	11111111	8005.91
31323134	ACE Consulting	123552	31323134	5111.47
57896883	XYZ Plumbing	123585	87323456	15231.62
87323456	Databases N-U	123777	57896883	23454.84

```
SELECT Cust.Customer_Number,  
       Customer_Name,  
       Order_Number,  
       Order_Total  
  FROM Customer_Table as Cust  
INNER JOIN  
       Order_Table as ORD  
  ON Cust.Customer_Number  
    = Ord.Customer_Number ;
```

On instead
of WHERE

INNER JOIN Keyword
Replaces the comma

This is the same join as the previous slide except it is using ANSI syntax. Both will return the same rows with the same performance. Rows are joined when the Customer_Number matches on both tables, but non-matches won't return.

Both Queries have the same Results and Performance

Customer_Table		Order_Table		
Customer_Number	Customer_Name	Order_Number	Customer_Number	Order_Total
11111111	Billy's Best Choice	123456	11111111	12347.53
31313131	Acme Products	123512	11111111	8005.91
31323134	ACE Consulting	123552	31323134	5111.47
57896883	XYZ Plumbing	123585	87323456	15231.62
87323456	Databases N-U	123777	57896883	23454.84

/* Traditional Syntax */

```
SELECT Cust.Customer_Number,  
       Customer_Name,  
       Order_Number,  
       Order_Total  
FROM   Customer_Table as Cust,  
       Order_Table as ORD  
WHERE  Cust.Customer_Number  
       = Ord.Customer_Number ;
```

/* ANSI Syntax */

```
SELECT Cust.Customer_Number,  
       Customer_Name,  
       Order_Number,  
       Order_Total  
FROM   Customer_Table as Cust  
INNER JOIN  
       Order_Table as ORD  
ON     Cust.Customer_Number  
       = Ord.Customer_Number ;
```

Both of these syntax techniques bring back the same result set and have the same performance. The INNER JOIN is considered ANSI. Which one does Outer Joins?

Quiz – Can You Finish the Join Syntax?

Employee_Table					Department_Table	
Employee_No	Dept_No	Last_Name	First_Name	Salary	Dept_No	Department_Name
1232578	100	Chambers	Mandee	48850.00	100	Marketing
1256349	400	Harrison	Herbert	54500.00	200	Research and Dev
2341218	400	Reilly	William	36000.00	300	Sales
2312225	300	Larkins	Lorraine	40200.00	400	Customer Support
2000000	?	Jones	Squiggy	32800.50	500	Human Resources
1000234	10	Smythe	Richard	32800.00		
1121334	400	Strickling	Cletus	54500.00		
1324657	200	Coffing	Billy	41888.88		
1333454	200	Smith	John	48000.00		

```
SELECT First_Name, Last_Name,  
       Department_Name  
FROM   Employee_Table as E  
INNER JOIN  
       Department_Table as D  
ON
```

Finish this join by placing the missing SQL in the proper place!

Answer to Quiz – Can You Finish the Join Syntax?

Employee_Table					Department_Table	
Employee_No	Dept_No	Last_Name	First_Name	Salary	Dept_No	Department_Name
1232578	100	Chambers	Mandee	48850.00	100	Marketing
1256349	400	Harrison	Herbert	54500.00	200	Research and Dev
2341218	400	Reilly	William	36000.00	300	Sales
2312225	300	Larkins	Lorraine	40200.00	400	Customer Support
2000000	?	Jones	Squiggy	32800.50	500	Human Resources
1000234	10	Smythe	Richard	32800.00		
1121334	400	Strickling	Cletus	54500.00		
1324657	200	Coffing	Billy	41888.88		
1333454	200	Smith	John	48000.00		

```
SELECT First_Name, Last_Name,  
       Department_Name  
FROM   Employee_Table as E  
INNER JOIN  
       Department_Table as D  
ON      E.Dept_No = D.Dept_No ;
```



This query is ready to run.

Quiz – Can You Find the Error?

Employee_Table					Department_Table	
Employee_No	Dept_No	Last_Name	First_Name	Salary	Dept_No	Department_Name
1232578	100	Chambers	Mandee	48850.00	100	Marketing
1256349	400	Harrison	Herbert	54500.00	200	Research and Dev
2341218	400	Reilly	William	36000.00	300	Sales
2312225	300	Larkins	Lorraine	40200.00	400	Customer Support
2000000	?	Jones	Squiggy	32800.50	500	Human Resources
1000234	10	Smythe	Richard	32800.00		
1121334	400	Strickling	Cletus	54500.00		
1324657	200	Coffing	Billy	41888.88		
1333454	200	Smith	John	48000.00		

```
SELECT First_Name, Last_Name, Dept_No  
      Department_Name  
FROM   Employee_Table as E  
INNER JOIN  
        Department_Table as D  
ON     E.Dept_No = D.Dept_No ;
```

This query has an error! Can you find it?

Answer to Quiz – Can You Find the Error?

Employee_Table					Department_Table	
Employee_No	Dept_No	Last_Name	First_Name	Salary	Dept_No	Department_Name
1232578	100	Chambers	Mandee	48850.00	100	Marketing
1256349	400	Harrison	Herbert	54500.00	200	Research and Dev
2341218	400	Reilly	William	36000.00	300	Sales
2312225	300	Larkins	Lorraine	40200.00	400	Customer Support
2000000	?	Jones	Squiggy	32800.50	500	Human Resources
1000234	10	Smythe	Richard	32800.00		
1121334	400	Strickling	Cletus	54500.00		
1324657	200	Coffing	Billy	41888.88		
1333454	200	Smith	John	48000.00		

```
SELECT First_Name, Last_Name, E.Dept_No  
      Department_Name  
  FROM Employee_Table as E  
INNER JOIN  
      Department_Table as D  
  ON      E.Dept_N o = D.Dept_No ;
```



If a column in the SELECT list is in both tables you must fully qualify it.

Quiz – Which rows from both tables Won’t Return?

Employee_Table					Department_Table	
Employee_No	Dept_No	Last_Name	First_Name	Salary	Dept_No	Department_Name
1232578	100	Chambers	Mandee	48850.00	100	Marketing
1256349	400	Harrison	Herbert	54500.00	200	Research and Dev
2341218	400	Reilly	William	36000.00	300	Sales
2312225	300	Larkins	Lorraine	40200.00	400	Customer Support
2000000	?	Jones	Squiggy	32800.50	500	Human Resources
1000234	10	Smythe	Richard	32800.00		
1121334	400	Strickling	Cletus	54500.00		
1324657	200	Coffing	Billy	41888.88		
1333454	200	Smith	John	48000.00		

```
SELECT First_Name, Last_Name,  
       Department_Name  
FROM   Employee_Table as E  
INNER JOIN  
        Department_Table as D  
ON     E.Dept_N o = D.Dept_No ;
```

An Inner Join returns matching rows, but did you know an Outer Join returns both matching rows and non-matching rows? You will understand soon!

Answer to Quiz – Which rows from both tables Won’t Return?

Employee_Table					Department_Table	
Employee_No	Dept_No	Last_Name	First_Name	Salary	Dept_No	Department_Name
1232578	100	Chambers	Mandee	48850.00	100	Marketing
1256349	400	Harrison	Herbert	54500.00	200	Research and Dev
2341218	400	Reilly	William	36000.00	300	Sales
2312225	300	Larkins	Lorraine	40200.00	400	Customer Support
2000000	?	Jones	Squiggy	32800.50	500	Human Resources
1000234	10	Smythe	Richard	32800.00		
1121334	400	Strickling	Cletus	54500.00		
1324657	200	Coffing	Billy	41888.88		
1333454	200	Smith	John	48000.00		

1

Squiggy Jones has a **NULL** Dept_No

2

Richard Smythe has an **invalid** Dept_No 10

3

No Employees work in Department 500

The bottom line is that the three rows excluded did not have a matching Dept_No.

LEFT OUTER JOIN

Employee_Table					Department_Table	
Employee_No	Dept_No	Last_Name	First_Name	Salary	Dept_No	Department_Name
1232578	100	Chambers	Mandee	48850.00	100	Marketing
1256349	400	Harrison	Herbert	54500.00	200	Research and Dev
2341218	400	Reilly	William	36000.00	300	Sales
2312225	300	Larkins	Lorraine	40200.00	400	Customer Support
2000000	?	Jones	Squiggy	32800.50	500	Human Resources
1000234	10	Smythe	Richard	32800.00		
1121334	400	Strickling	Cletus	54500.00		
1324657	200	Coffing	Billy	41888.88		
1333454	200	Smith	John	48000.00		

```
SELECT First_Name, Department_Name
FROM Employee_Table as E
LEFT OUTER JOIN
    Department_Table as D
ON E.Dept_No = D.Dept_No ;
```

1st Table
after **FROM**
is always the
LEFT Table

This is a LEFT OUTER JOIN. That means that all rows from the LEFT Table will appear in the report regardless if it finds a match on the right table.

Employee_Table

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

Department_Table

Dept_No	Department_Name
100	Marketing
200	Research and Dev
300	Sales
400	Customer Support
500	Human Resources

This row is the only row that didn't return.

Left OUTER JOIN

```
SELECT First_Name, Department_Name
FROM Employee_Table as E
LEFT OUTER JOIN
    Department_Table as D
ON E.Dept_No = D.Dept_No ;
```

First_Name	Department_Name
Mandee	Marketing
Herbert	Customer Support
William	Customer Support
Lorraine	Sales
Squiggy	?
Richard	?
Cletus	Customer Support
Billy	Research and Dev
John	Research and Dev

A LEFT Outer Join Returns all rows from the LEFT Table, including all Matches. If a LEFT row can't find a match a NULL is placed on right columns not found!

RIGHT OUTER JOIN

Employee_Table					Department_Table	
Employee_No	Dept_No	Last_Name	First_Name	Salary	Dept_No	Department_Name
1232578	100	Chambers	Mandee	48850.00	100	Marketing
1256349	400	Harrison	Herbert	54500.00	200	Research and Dev
2341218	400	Reilly	William	36000.00	300	Sales
2312225	300	Larkins	Lorraine	40200.00	400	Customer Support
2000000	?	Jones	Squiggy	32800.50	500	Human Resources
1000234	10	Smythe	Richard	32800.00		
1121334	400	Strickling	Cletus	54500.00		
1324657	200	Coffing	Billy	41888.88		
1333454	200	Smith	John	48000.00		

```
SELECT First_Name, Department_Name  
FROM Employee_Table as E  
RIGHT OUTER JOIN  
        Department_Table as D  
ON E.Dept_No = D.Dept_No ;
```

2nd Table
after FROM
Is always the
RIGHT Table

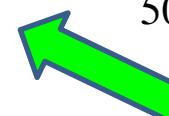
This is a RIGHT OUTER JOIN. That means that all rows from the RIGHT Table will appear in the report regardless if it finds a match with the LEFT Table.

Employee_Table

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

Department_Table

Dept_No	Department_Name
100	Marketing
200	Research and Dev
300	Sales
400	Customer Support
500	Human Resources



Rows colored in red did not return. Why?

RIGHT OUTER JOIN

```
SELECT First_Name, Department_Name
FROM Employee_Table as E
RIGHT OUTER JOIN
    Department_Table as D
ON E.Dept_No = D.Dept_No ;
```

First_Name	Department_Name
Mandee	Marketing
Herbert	Customer Support
William	Customer Support
Lorraine	Sales
Cletus	Customer Support
Billy	Research and Dev
John	Research and Dev
?	Human Resources

All rows from the Right Table were returned with matches and Dept_No 500 didn't have a match so the system put a NULL Value for Left Column values.

FULL OUTER JOIN

Employee_Table					Department_Table	
Employee_No	Dept_No	Last_Name	First_Name	Salary	Dept_No	Department_Name
1232578	100	Chambers	Mandee	48850.00	100	Marketing
1256349	400	Harrison	Herbert	54500.00	200	Research and Dev
2341218	400	Reilly	William	36000.00	300	Sales
2312225	300	Larkins	Lorraine	40200.00	400	Customer Support
2000000	?	Jones	Squiggy	32800.50	500	Human Resources
1000234	10	Smythe	Richard	32800.00		
1121334	400	Strickling	Cletus	54500.00		
1324657	200	Coffing	Billy	41888.88		
1333454	200	Smith	John	48000.00		

```
SELECT First_Name, Department_Name
FROM Employee_Table as E
FULL OUTER JOIN 
    Department_Table as D
ON E.Dept_No = D.Dept_No ;
```

Full Outer will
Return all rows
from both tables!

The is a FULL OUTER JOIN. That means that all rows from both the RIGHT and LEFT Table will appear in the report regardless if it finds a match.

Employee_Table

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

Department_Table

Dept_No	Department_Name
100	Marketing
200	Research and Dev
300	Sales
400	Customer Support
500	Human Resources

FULL OUTER JOIN

```
SELECT First_Name, Department_Name
FROM Employee_Table as E
FULL OUTER JOIN
    Department_Table as D
ON E.Dept_No = D.Dept_No ;
```

First_Name	Department_Name
Mandee	Marketing
Herbert	Customer Support
William	Customer Support
Lorraine	Sales
Squiggy	?
Richard	?
Cletus	Customer Support
Billy	Research and Dev
John	Research and Dev
	Human Resources



The FULL Outer Join Returns all rows from both Tables. NULLs show the flaws!

INNER JOIN with Additional AND Clause

Employee_Table					Department_Table	
Employee_No	Dept_No	Last_Name	First_Name	Salary	Dept_No	Department_Name
1232578	100	Chambers	Mandee	48850.00	100	Marketing
1256349	400	Harrison	Herbert	54500.00	200	Research and Dev
2341218	400	Reilly	William	36000.00	300	Sales
2312225	300	Larkins	Lorraine	40200.00	400	Customer Support
2000000	?	Jones	Squiggy	32800.50	500	Human Resources
1000234	10	Smythe	Richard	32800.00		
1121334	400	Strickling	Cletus	54500.00		
1324657	200	Coffing	Billy	41888.88		
1333454	200	Smith	John	48000.00		

```
SELECT First_Name, Last_Name, Department_Name  
FROM Employee_Table as E,  
      Department_Table as D  
WHERE E.Dept_No = D.Dept_No  
AND Department_Name like 'Marke%';
```



The additional AND is performed first in order to eliminate unwanted data so the join is less intensive than joining everything first and then eliminating.

ANSI INNER JOIN with Additional AND Clause

Employee_Table					Department_Table	
Employee_No	Dept_No	Last_Name	First_Name	Salary	Dept_No	Department_Name
1232578	100	Chambers	Mandee	48850.00	100	Marketing
1256349	400	Harrison	Herbert	54500.00	200	Research and Dev
2341218	400	Reilly	William	36000.00	300	Sales
2312225	300	Larkins	Lorraine	40200.00	400	Customer Support
2000000	?	Jones	Squiggy	32800.50	500	Human Resources
1000234	10	Smythe	Richard	32800.00		
1121334	400	Strickling	Cletus	54500.00		
1324657	200	Coffing	Billy	41888.88		
1333454	200	Smith	John	48000.00		

```
SELECT First_Name, Last_Name, Department_Name
FROM Employee_Table as E
INNER JOIN
    Department_Table as D
ON E.Dept_No = D.Dept_No
AND Department_Name like 'Marke%';
```



The additional AND is performed first in order to eliminate unwanted data so the join is less intensive than joining everything first and then eliminating after.

ANSI INNER JOIN with Additional WHERE Clause

Employee_Table					Department_Table	
Employee_No	Dept_No	Last_Name	First_Name	Salary	Dept_No	Department_Name
1232578	100	Chambers	Mandee	48850.00	100	Marketing
1256349	400	Harrison	Herbert	54500.00	200	Research and Dev
2341218	400	Reilly	William	36000.00	300	Sales
2312225	300	Larkins	Lorraine	40200.00	400	Customer Support
2000000	?	Jones	Squiggy	32800.50	500	Human Resources
1000234	10	Smythe	Richard	32800.00		
1121334	400	Strickling	Cletus	54500.00		
1324657	200	Coffing	Billy	41888.88		
1333454	200	Smith	John	48000.00		

```
SELECT First_Name, Last_Name, Department_Name
FROM Employee_Table as E
INNER JOIN
    Department_Table as D
ON E.Dept_No = D.Dept_No
WHERE Department_Name like 'Marke%';
```



The additional WHERE is performed first in order to eliminate unwanted data so the join is less intensive than joining everything first and then eliminating.

OUTER JOIN with Additional WHERE Clause

Employee_Table					Department_Table	
Employee_No	Dept_No	Last_Name	First_Name	Salary	Dept_No	Department_Name
1232578	100	Chambers	Mandee	48850.00	100	Marketing
1256349	400	Harrison	Herbert	54500.00	200	Research and Dev
2341218	400	Reilly	William	36000.00	300	Sales
2312225	300	Larkins	Lorraine	40200.00	400	Customer Support
2000000	?	Jones	Squiggy	32800.50	500	Human Resources
1000234	10	Smythe	Richard	32800.00		
1121334	400	Strickling	Cletus	54500.00		
1324657	200	Coffing	Billy	41888.88		
1333454	200	Smith	John	48000.00		

```
SELECT First_Name, Last_Name,  
       Department_Name  
FROM   Employee_Table  as E  
LEFT OUTER JOIN  
       Department_Table as D  
ON   E.Dept_No = D.Dept_No  
WHERE Dept_No = 100 ;
```

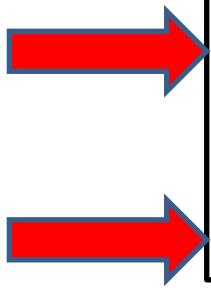
First_Name	Department_Name
Mandee	Marketing

Only Mandee Chambers
is in Dept_No 100

The additional WHERE is performed last on Outer Joins. All rows will be joined first and then the additional WHERE clause filters after the join takes place.

OUTER JOIN with Additional AND Clause

Employee_Table					Department_Table	
Employee_No	Dept_No	Last_Name	First_Name	Salary	Dept_No	Department_Name
1232578	100	Chambers	Mandee	48850.00	100	Marketing
1256349	400	Harrison	Herbert	54500.00	200	Research and Dev
2341218	400	Reilly	William	36000.00	300	Sales
2312225	300	Larkins	Lorraine	40200.00	400	Customer Support
2000000	?	Jones	Squiggy	32800.50	500	Human Resources
1000234	10	Smythe	Richard	32800.00		
1121334	400	Strickling	Cletus	54500.00		
1324657	200	Coffing	Billy	41888.88		
1333454	200	Smith	John	48000.00		



```
SELECT First_Name, Department_Name
FROM Employee_Table as E
LEFT OUTER JOIN
    Department_Table as D
ON E.Dept_No = D.Dept_No
AND E.Dept_No = 100 ;
```

The additional AND is performed in conjunction with the ON statement on Outer Joins. All rows will be evaluated with the ON clause and the AND combined.

Employee_Table

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

Department_Table

<u>Dept_No</u>	<u>Department_Name</u>
100	Marketing
200	Research and Dev
300	Sales
400	Customer Support
500	Human Resources

OUTER Join with additional AND

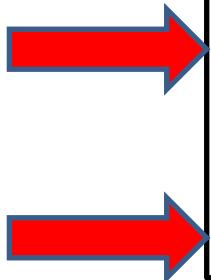
```
SELECT First_Name, Department_Name
FROM Employee_Table as E
LEFT OUTER JOIN
    Department_Table as D
ON E.Dept_No = D.Dept_No
AND E.Dept_No = 100 ;
```

<u>First Name</u>	<u>Department Name</u>
Mandee	Marketing
Herbert	?
William	?
Lorraine	?
Squiggy	?
Richard	?
Cletus	?
Billy	?
John	?

The additional AND is performed in conjunction with the ON statement on Outer Joins. This can surprise you. Only Mandee is in Dept_No 100!

Quiz – Why is this Considered an INNER JOIN?

Employee_Table					Department_Table	
Employee_No	Dept_No	Last_Name	First_Name	Salary	Dept_No	Department_Name
1232578	100	Chambers	Mandee	48850.00	100	Marketing
1256349	400	Harrison	Herbert	54500.00	200	Research and Dev
2341218	400	Reilly	William	36000.00	300	Sales
2312225	300	Larkins	Lorraine	40200.00	400	Customer Support
2000000	?	Jones	Squiggy	32800.50	500	Human Resources
1000234	10	Smythe	Richard	32800.00		
1121334	400	Strickling	Cletus	54500.00		
1324657	200	Coffing	Billy	41888.88		
1333454	200	Smith	John	48000.00		



```
SELECT First_Name, Department_Name
FROM Employee_Table as E
LEFT OUTER JOIN
    Department_Table as D
ON E.Dept_No = D.Dept_No
AND Department_Name like 'Marke%';
```

This is considered an INNER JOIN because we are doing a LEFT OUTER JOIN on the Employee_Table and then filtering with the AND for a column in the right table!

The DREADED Product Join

Employee_Table					Department_Table	
Employee_No	Dept_No	Last_Name	First_Name	Salary	Dept_No	Department_Name
1232578	100	Chambers	Mandee	48850.00	100	Marketing
1256349	400	Harrison	Herbert	54500.00	200	Research and Dev
2341218	400	Reilly	William	36000.00	300	Sales
2312225	300	Larkins	Lorraine	40200.00	400	Customer Support
2000000	?	Jones	Squiggy	32800.50	500	Human Resources
1000234	10	Smythe	Richard	32800.00		
1121334	400	Strickling	Cletus	54500.00		
1324657	200	Coffing	Billy	41888.88		
1333454	200	Smith	John	48000.00		

No Join
Condition
Linking the
Two Tables!



```
SELECT First_Name, Last_Name, Department_Name
FROM Employee_Table as E,
     Department_Table as D
WHERE Department_Name like '%m%'
Order by 1, 2, 3;
```

This query becomes a Product Join because it does not possess any JOIN Conditions (Join Keys). Every row from one table is compared to every row of the other table and quite often the data is not what you intended to get back.

The DREADED Product Join

No Join
Condition
Linking the
Two Tables!

```
SELECT First_Name, Last_Name, Department_Name  
FROM Employee_Table as E,  
      Department_Table as D  
WHERE Department_Name like '%m%'  
Order by 1, 2, 3;
```

<u>First Name</u>	<u>Last Name</u>	<u>Department Name</u>
Billy	Coffing	Customer Support
Billy	Coffing	Human Resources
Billy	Coffing	Marketing
Cletus	Strickling	Customer Support
Cletus	Strickling	Human Resources
Cletus	Strickling	Marketing
Herbert	Harrison	Customer Support
Herbert	Harrison	Human Resources
Herbert	Harrison	Marketing

Not all
returned rows
shown here

27 Rows came
back. 9 employees
with each working
in 3 different
departments
simultaneously!
Impressive!

WRONG!

How can **Billy Coffing** work in **3** different departments?

A Product Join is often a mistake! 3 Department rows had an 'm' in their name so these were joined to every employee, and the information is worthless.

The Horrifying Cartesian Product Join

Employee_Table					Department_Table	
Employee_No	Dept_No	Last_Name	First_Name	Salary	Dept_No	Department_Name
1232578	100	Chambers	Mandee	48850.00	100	Marketing
1256349	400	Harrison	Herbert	54500.00	200	Research and Dev
2341218	400	Reilly	William	36000.00	300	Sales
2312225	300	Larkins	Lorraine	40200.00	400	Customer Support
2000000	?	Jones	Squiggy	32800.50	500	Human Resources
1000234	10	Smythe	Richard	32800.00		
1121334	400	Strickling	Cletus	54500.00		
1324657	200	Coffing	Billy	41888.88		
1333454	200	Smith	John	48000.00		

No WHERE
Clause in
the join!

SELECT First_Name, Last_Name, Department_Name
FROM Employee_Table as E,
Department_Table as D

This joins every row from one table to every row of another table.

9 rows multiplied by 5 rows = 45 rows of complete nonsense!

A Cartesian Product Join is a big mistake.

The ANSI Cartesian Join will ERROR

Employee_Table					Department_Table	
Employee_No	Dept_No	Last_Name	First_Name	Salary	Dept_No	Department_Name
1232578	100	Chambers	Mandee	48850.00	100	Marketing
1256349	400	Harrison	Herbert	54500.00	200	Research and Dev
2341218	400	Reilly	William	36000.00	300	Sales
2312225	300	Larkins	Lorraine	40200.00	400	Customer Support
2000000	?	Jones	Squiggy	32800.50	500	Human Resources
1000234	10	Smythe	Richard	32800.00		
1121334	400	Strickling	Cletus	54500.00		
1324657	200	Coffing	Billy	41888.88		
1333454	200	Smith	John	48000.00		

```
SELECT First_Name, Last_Name, Department_Name  
FROM Employee_Table as E  
INNER JOIN  
Department_Table as D
```

No ON
Clause in
the join!



This causes an error. ANSI won't let this run unless a join condition is present.

Quiz – Do these Joins Return the Same Answer Set?

Employee_Table					Department_Table	
Employee_No	Dept_No	Last_Name	First_Name	Salary	Dept_No	Department_Name
1232578	100	Chambers	Mandee	48850.00	100	Marketing
1256349	400	Harrison	Herbert	54500.00	200	Research and Dev
2341218	400	Reilly	William	36000.00	300	Sales
2312225	300	Larkins	Lorraine	40200.00	400	Customer Support
2000000	?	Jones	Squiggy	32800.50	500	Human Resources
1000234	10	Smythe	Richard	32800.00		
1121334	400	Strickling	Cletus	54500.00		
1324657	200	Coffing	Billy	41888.88		
1333454	200	Smith	John	48000.00		

/* Query 1 */

```
SELECT First_Name,  
       Department_Name  
FROM Employee_Table  
INNER JOIN  
       Department_Table ;
```

/* Query 2 */

```
SELECT First_Name,  
       Department_Name  
FROM Employee_Table,  
       Department_Table ;
```

Do these two queries produce the same result?

Answer – Do these Joins Return the Same Answer Set?

Employee_Table					Department_Table	
Employee_No	Dept_No	Last_Name	First_Name	Salary	Dept_No	Department_Name
1232578	100	Chambers	Mandee	48850.00	100	Marketing
1256349	400	Harrison	Herbert	54500.00	200	Research and Dev
2341218	400	Reilly	William	36000.00	300	Sales
2312225	300	Larkins	Lorraine	40200.00	400	Customer Support
2000000	?	Jones	Squiggy	32800.50	500	Human Resources
1000234	10	Smythe	Richard	32800.00		
1121334	400	Strickling	Cletus	54500.00		
1324657	200	Coffing	Billy	41888.88		
1333454	200	Smith	John	48000.00		

/* Query 1 */

```
SELECT First_Name,  
       Department_Name  
FROM Employee_Table  
INNER JOIN  
       Department_Table ;
```

/* Query 2 */

```
SELECT First_Name,  
       Department_Name  
FROM Employee_Table,  
       Department_Table ;
```

Do these two queries produce the same result? No, Query 1 Errors due to ANSI syntax and no ON Clause, but Query 2 Product Joins to bring back junk!

The CROSS JOIN

Customer_Table		Order_Table		
Customer_Number	Customer_Name	Order_Number	Customer_Number	Order_Total
11111111	Billy's Best Choice	123456	11111111	12347.53
31313131	Acme Products	123512	11111111	8005.91
31323134	ACE Consulting	123552	31323134	5111.47
57896883	XYZ Plumbing	123585	87323456	15231.62
87323456	Databases N-U	123777	57896883	23454.84

A Cross Join
is the ANSI
equivalent to
a Product Join

SELECT Customer_Name,
Order_Number
FROM Customer_Table
CROSS JOIN
Order_Table
WHERE Order_Number = 123456
ORDER BY 1;

Only a **WHERE**
will work. **ON**
Will NOT!

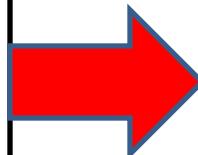
This query becomes a Product Join because a Cross Join is an ANSI Product Join. It will compare every row from the Customer_Table to Order_Number 123456 in the Order_Table. Check out the Answer Set on the next page.

The CROSS JOIN Answer Set

Customer_Table		Order_Table		
Customer_Number	Customer_Name	Order_Number	Customer_Number	Order_Total
11111111	Billy's Best Choice	123456	11111111	12347.53
31313131	Acme Products	123512	11111111	8005.91
31323134	ACE Consulting	123552	31323134	5111.47
57896883	XYZ Plumbing	123585	87323456	15231.62
87323456	Databases N-U	123777	57896883	23454.84

```
SELECT Customer_Name,  
       Order_Number  
FROM   Customer_Table  
CROSS JOIN  
       Order_Table  
WHERE  Order_Number = 123456  
ORDER BY 1;
```

Answer
Set



Customer_Name	Order_Number
Billy's Best Choice	123456
Acme Products	123456
ACE Consulting	123456
XYZ Plumbing	123456
Databases N-U	123456

This Cross Join produces information that just isn't worth anything quite often!

The Self Join

Employee_Table2

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

Notice we
Added a
column
which
Will tell us
If someone
is a
manager

```
SELECT Mgrs.Dept_No, Mgrs.Last_Name as MgrName, Mgrs.Salary as MgrSal,  
Emps.Last_Name as EmpName, Emps.Salary as Empsal  
FROM Employee_Table2 as Emps,  
Employee_Table2 as Mgrs  
WHERE Emps.Dept_No = Mgrs.Dept_No  
AND Mgrs.Mgr = 'Y'  
AND Emps.Salary > Mgrs.Salary ;
```

Which Workers
make a bigger
Salary than
their Manager?

A Self Join gives itself 2 different Aliases, which is then seen as two different tables.

The Self Join with ANSI Syntax

Employee_Table2

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

Notice we
Added a
column
which
Will tell us
If someone
is a
manager

```
SELECT Mgrs.Dept_No, Mgrs.Last_Name as MgrName, Mgrs.Salary as MgrSal,  
Emps.Last_Name as EmpName, Emps.Salary as Empsal  
FROM Employee_Table2 as Emps  
INNER JOIN  
    Employee_Table2 as Mgrs  
ON Emps.Dept_No = Mgrs.Dept_No  
WHERE Mgrs.Mgr = 'Y'  
AND Emps.Salary > Mgrs.Salary ;
```

Which Workers
make a bigger
Salary than
their Manager?

A Self Join gives itself 2 different Aliases, which is then seen as two different tables.

Quiz – Will both queries bring back the same Answer Set?

Customer_Table		Order_Table		
Customer_Number	Customer_Name	Order_Number	Customer_Number	Order_Total
11111111	Billy's Best Choice	123456	11111111	12347.53
31313131	Acme Products	123512	11111111	8005.91
31323134	ACE Consulting	123552	31323134	5111.47
57896883	XYZ Plumbing	123585	87323456	15231.62
87323456	Databases N-U	123777	57896883	23454.84

```
SELECT Customer_Name,  
       Order_Number,  
       Order_Total  
  FROM Customer_Table as Cust  
INNER JOIN Order_Table as ORD  
  ON      Cust.Customer_Number  
        =  Ord.Customer_Number  
WHERE  
Customer_Name like 'Billy%'  
ORDER BY 1;
```

```
SELECT Customer_Name,  
       Order_Number,  
       Order_Total  
  FROM Customer_Table as Cust  
INNER JOIN Order_Table as ORD  
  ON      Cust.Customer_Number  
        =  Ord.Customer_Number  
AND  
Customer_Name like 'Billy%'  
ORDER BY 1;
```

Will both queries bring back the same result set?

Answer – Will both queries bring back the same Answer Set?

Customer_Table		Order_Table		
Customer_Number	Customer_Name	Order_Number	Customer_Number	Order_Total
11111111	Billy's Best Choice	123456	11111111	12347.53
31313131	Acme Products	123512	11111111	8005.91
31323134	ACE Consulting	123552	31323134	5111.47
57896883	XYZ Plumbing	123585	87323456	15231.62
87323456	Databases N-U	123777	57896883	23454.84

```
SELECT Customer_Name,  
       Order_Number,  
       Order_Total  
FROM Customer_Table as Cust  
INNER JOIN Order_Table as ORD  
ON      Cust.Customer_Number  
        = Ord.Customer_Number  
WHERE  
Customer_Name like 'Billy%'  
ORDER BY 1;
```

```
SELECT Customer_Name,  
       Order_Number,  
       Order_Total  
FROM Customer_Table as Cust  
INNER JOIN Order_Table as ORD  
ON      Cust.Customer_Number  
        = Ord.Customer_Number  
AND  
Customer_Name like 'Billy%'  
ORDER BY 1;
```

Will both queries bring back the same result set? Yes! Because they're both inner joins.

Quiz – Will both queries bring back the same Answer Set?

Customer_Table		Order_Table		
Customer_Number	Customer_Name	Order_Number	Customer_Number	Order_Total
11111111	Billy's Best Choice	123456	11111111	12347.53
31313131	Acme Products	123512	11111111	8005.91
31323134	ACE Consulting	123552	31323134	5111.47
57896883	XYZ Plumbing	123585	87323456	15231.62
87323456	Databases N-U	123777	57896883	23454.84

```
SELECT Customer_Name,  
       Order_Number,  
       Order_Total  
FROM Customer_Table as Cust  
LEFT OUTER JOIN  
       Order_Table as ORD  
ON      Cust.Customer_Number  
          = Ord.Customer_Number  
WHERE  
Customer_Name like 'Billy%'  
ORDER BY 1;
```

```
SELECT Customer_Name,  
       Order_Number,  
       Order_Total  
FROM Customer_Table as Cust  
LEFT OUTER JOIN  
       Order_Table as ORD  
ON      Cust.Customer_Number  
          = Ord.Customer_Number  
AND  
Customer_Name like 'Billy%'  
ORDER BY 1;
```

Will both queries bring back the same result set?

Answer – Will both queries bring back the same Answer Set?

Customer_Table		Order_Table		
Customer_Number	Customer_Name	Order_Number	Customer_Number	Order_Total
11111111	Billy's Best Choice	123456	11111111	12347.53
31313131	Acme Products	123512	11111111	8005.91
31323134	ACE Consulting	123552	31323134	5111.47
57896883	XYZ Plumbing	123585	87323456	15231.62
87323456	Databases N-U	123777	57896883	23454.84

```
SELECT Customer_Name,  
       Order_Number,  
       Order_Total  
  FROM Customer_Table as Cust  
LEFT OUTER JOIN  
       Order_Table as ORD  
  ON    Cust.Customer_Number  
       =  Ord.Customer_Number  
WHERE  
Customer_Name like 'Billy%'  
ORDER BY 1;
```

```
SELECT Customer_Name,  
       Order_Number,  
       Order_Total  
  FROM Customer_Table as Cust  
LEFT OUTER JOIN  
       Order_Table as ORD  
  ON    Cust.Customer_Number  
       =  Ord.Customer_Number  
AND  
Customer_Name like 'Billy%'  
ORDER BY 1;
```

Will both queries bring back the same result set? **NO!** The **WHERE** is performed last.

How would you Join these two tables?

Course_Table				
Course_ID	Course_Name	Credits	Seats	
100	Database Concepts	3	50	
200	Introduction to SQL	3	20	
210	Advanced SQL	3	22	
220	V2R3 SQL Features	2	25	
300	Physical Database Design	4	20	
400	Database Administration	4	16	

Student_Table					
Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt	
423400	Larkins	Michael	FR	0.00	
231222	Wilson	Susie	SO	3.80	
280023	McRoberts	Richard	JR	1.90	
322133	Bond	Jimmy	JR	3.95	
125634	Hanson	Henry	FR	2.88	
333450	Smith	Andy	SO	2.00	
324652	Delaney	Danny	SR	3.35	
260000	Johnson	Stanley	?	?	
234121	Thomas	Wendy	FR	4.00	
123250	Phillips	Martin	SR	3.00	

Looking at the columns above which will allow these two tables to join?

How would you Join these two tables? You Can't....Yet!

Course_Table				
Course_ID	Course_Name	Credits	Seats	
100	Database Concepts	3	50	
200	Introduction to SQL	3	20	
210	Advanced SQL	3	22	
220	V2R3 SQL Features	2	25	
300	Physical Database Design	4	20	
400	Database Administration	4	16	

Student_Table					
Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt	
423400	Larkins	Michael	FR	0.00	
231222	Wilson	Susie	SO	3.80	
280023	McRoberts	Richard	JR	1.90	
322133	Bond	Jimmy	JR	3.95	
125634	Hanson	Henry	FR	2.88	
333450	Smith	Andy	SO	2.00	
324652	Delaney	Danny	SR	3.35	
260000	Johnson	Stanley	?	?	
234121	Thomas	Wendy	FR	4.00	
123250	Phillips	Martin	SR	3.00	

Get ready for the Associative Table!

An Associative Table is a Bridge that Joins Two Tables

Associative
Table


Student_Course_Table

Student_ID Course_ID

280023	210
231222	210
125634	100
231222	220
125634	200
322133	220
125634	220
322133	300
324652	200
333450	500
260000	400
333450	400
234121	100
123250	100

Course_Table

Course_ID	Course_Name	Credits	Seats
100	Database Concepts	3	50
200	Introduction to SQL	3	20
210	Advanced SQL	3	22
220	V2R3 SQL Features	2	25
300	Physical Database Design	4	20
400	Database Administration	4	16

Student_Table

Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
423400	Larkins	Michael	FR	0.00
231222	Wilson	Susie	SO	3.80
280023	McRoberts	Richard	JR	1.90
322133	Bond	Jimmy	JR	3.95
125634	Hanson	Henry	FR	2.88
333450	Smith	Andy	SO	2.00
324652	Delaney	Danny	SR	3.35
260000	Johnson	Stanley	?	?
234121	Thomas	Wendy	FR	4.00
123250	Phillips	Martin	SR	3.00

The Associative Table is a bridge between the Course_Table and Student_Table.

Quiz – Can you Write the 3-Table Join?

Associative

Table



Student_Course_Table

Student_ID Course_ID

280023	210
231222	210
125634	100
231222	220
125634	200
322133	220
125634	220
322133	300
324652	200
333450	500
260000	400
333450	400
234121	100
123250	100

Course_Table

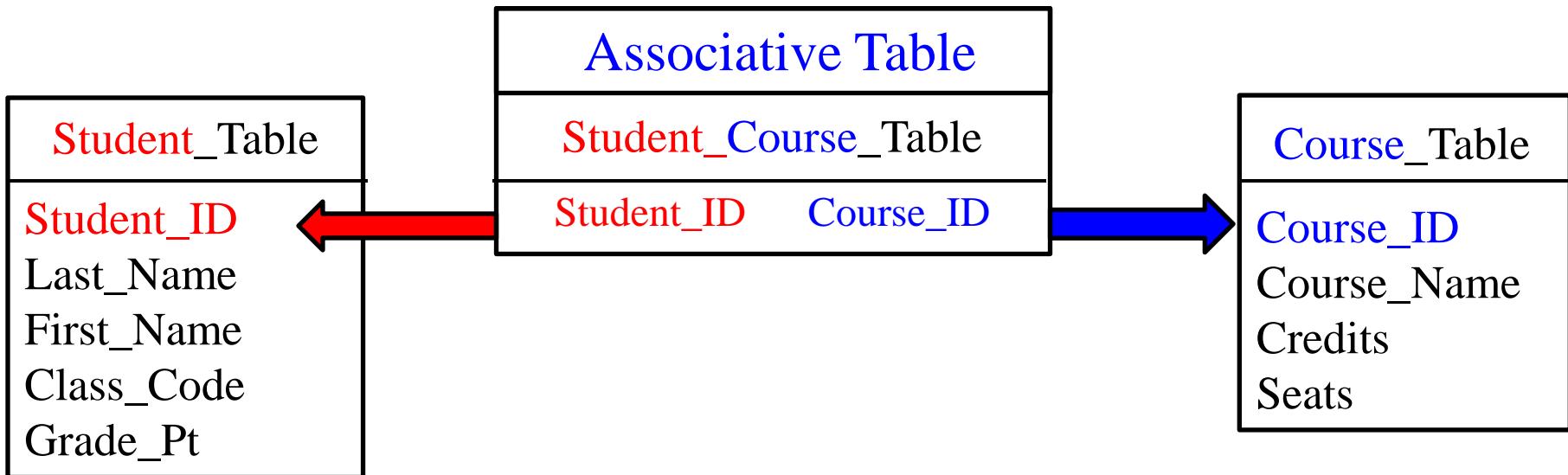
Course_ID	Course_Name	Credits	Seats
100	Database Concepts	3	50
200	Introduction to SQL	3	20
210	Advanced SQL	3	22
220	V2R3 SQL Features	2	25
300	Physical Database Design	4	20
400	Database Administration	4	16

Student_Table

Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
423400	Larkins	Michael	FR	0.00
231222	Wilson	Susie	SO	3.80
280023	McRoberts	Richard	JR	1.90
322133	Bond	Jimmy	JR	3.95
125634	Hanson	Henry	FR	2.88
333450	Smith	Andy	SO	2.00
324652	Delaney	Danny	SR	3.35
260000	Johnson	Stanley	?	?
234121	Thomas	Wendy	FR	4.00
123250	Phillips	Martin	SR	3.00

SELECT ALL Columns from the Course_Table and Student_Table and Join them.

Answer to Quiz – Can you Write the 3-Table Join?

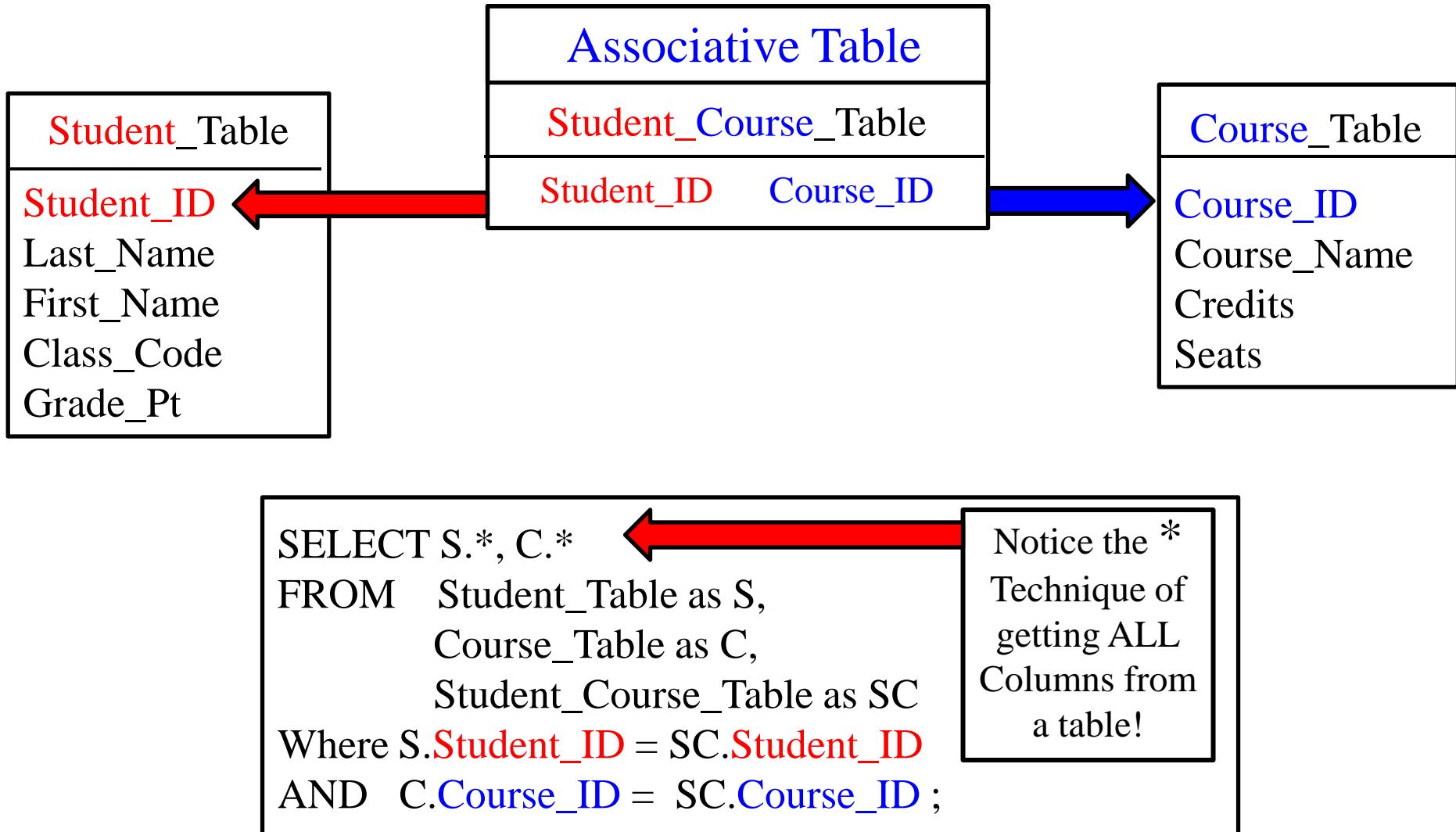


```
SELECT S.*, C.*  
FROM Student_Table as S,  
      Course_Table as C,  
      Student_Course_Table as SC  
Where S.Student_ID = SC.Student_ID  
AND   C.Course_ID = SC.Course_ID ;
```

Notice the *
Technique of
getting ALL
Columns from
a table!

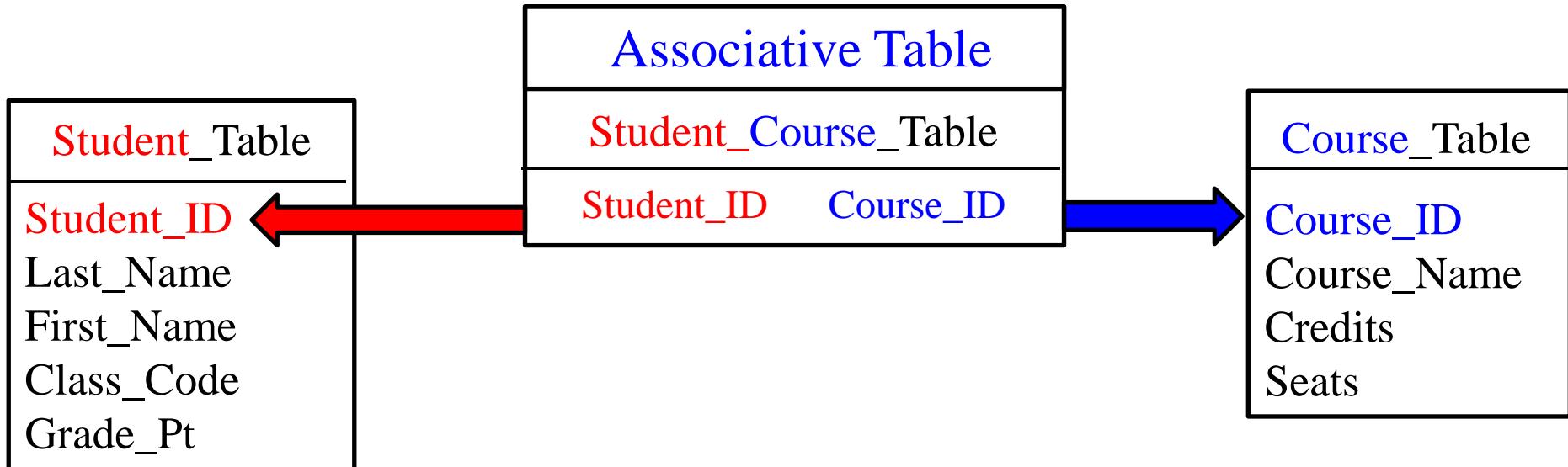
The Associative Table is a bridge between the Course_Table and Student_Table.

Quiz – Can you Write the 3-Table Join to ANSI Syntax?



Please re-write the above query using ANSI Syntax.

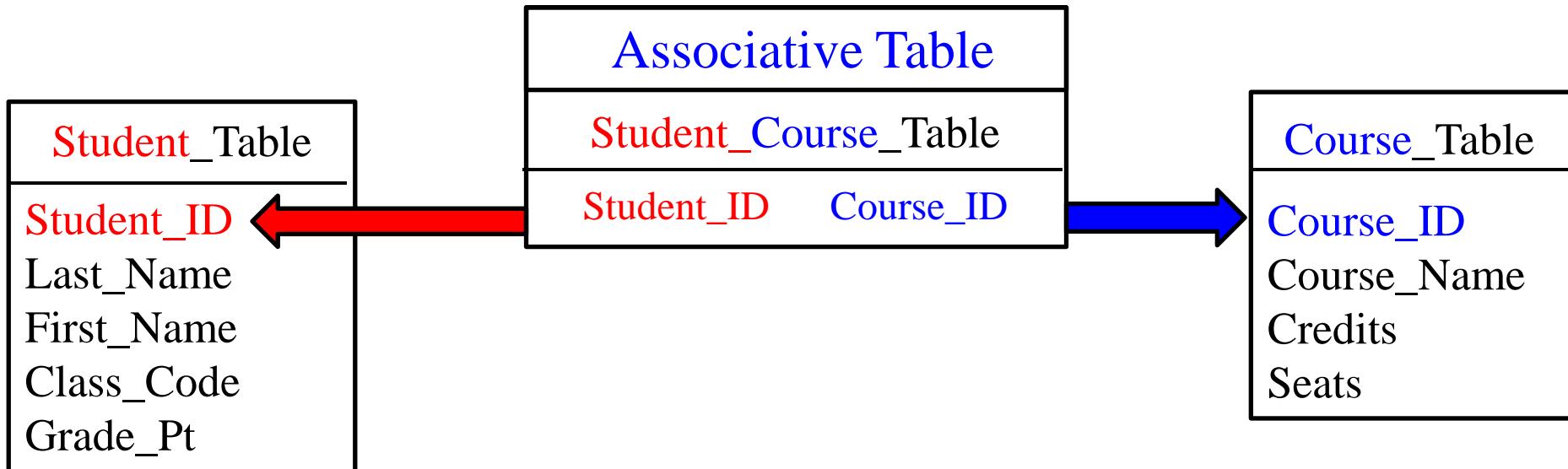
Answer – Can you Write the 3-Table Join to ANSI Syntax?



```
Select S.*, C.*  
From Student_Table as S  
INNER JOIN  
    Student_Course_Table as SC  
ON          S.Student_ID = SC.Student_ID  
INNER JOIN  
    Course_Table as C  
ON          C.Course_ID = SC.Course_ID;
```

The above query has been written using ANSI Syntax.

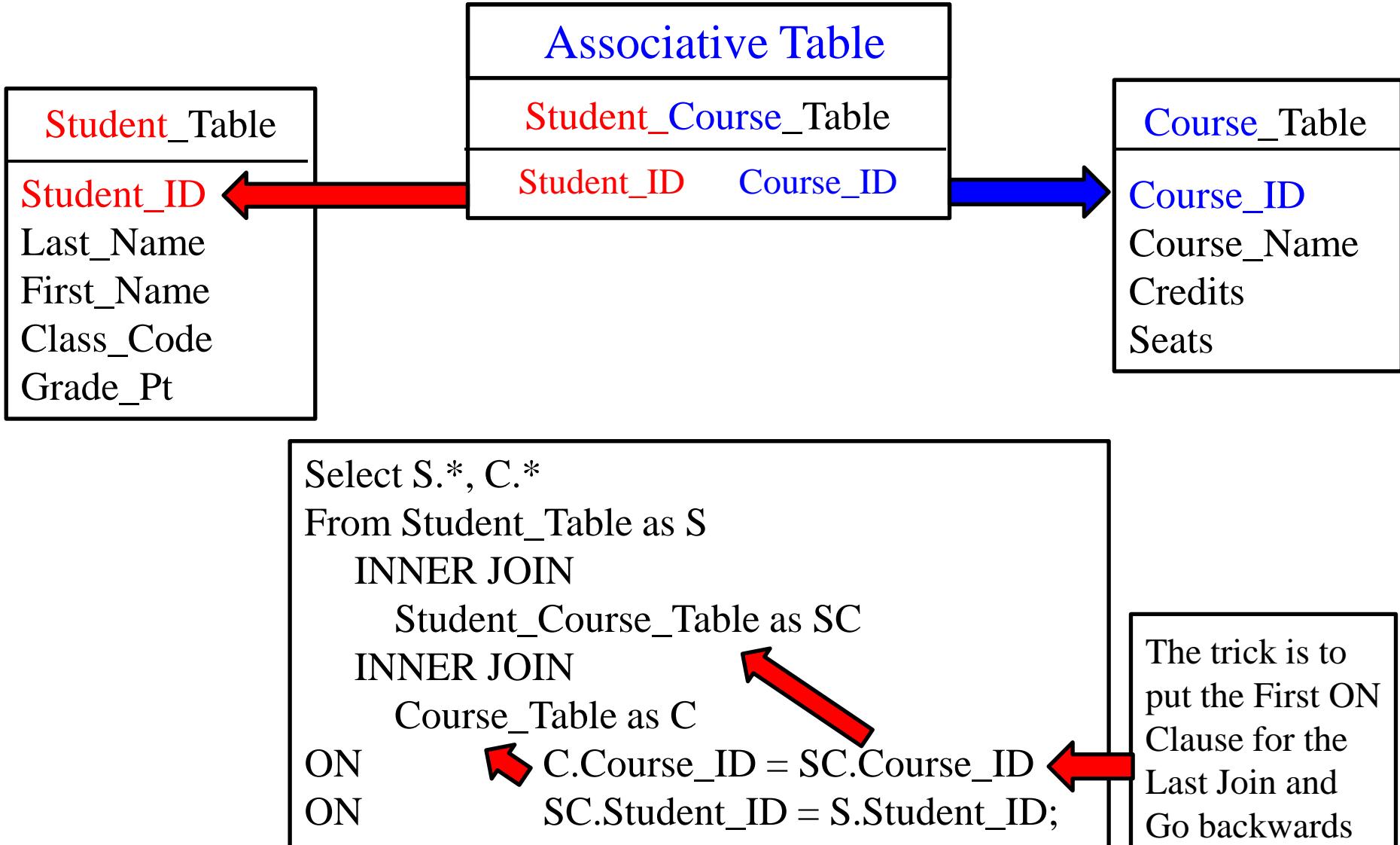
Quiz – Can you Place the ON Clauses at the End?



```
Select S.*, C.*  
From Student_Table as S  
INNER JOIN  
    Student_Course_Table as SC  
ON          S.Student_ID = SC.Student_ID  
INNER JOIN  
    Course_Table as C  
ON          C.Course_ID = SC.Course_ID;
```

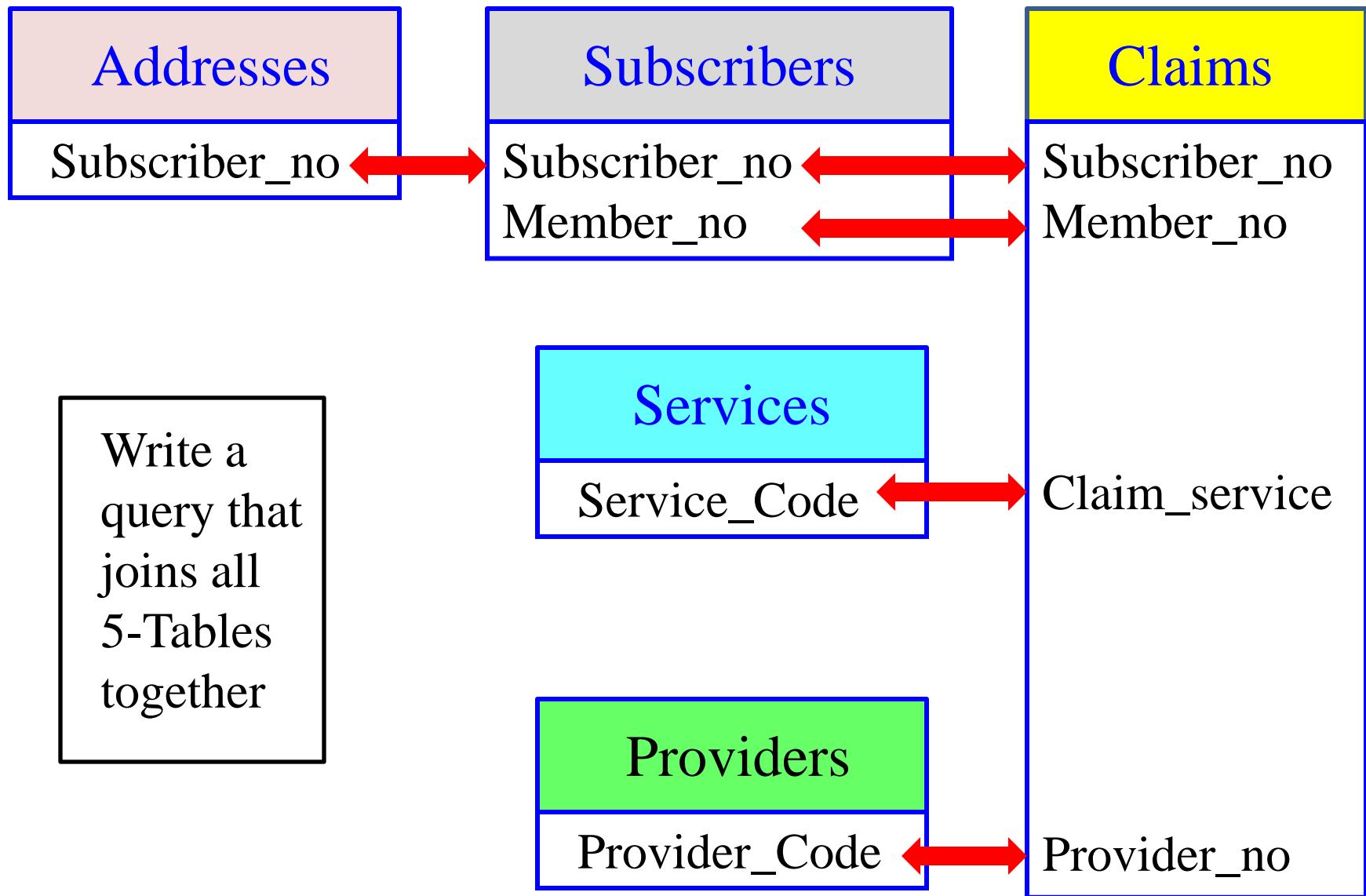
Please re-write the above query and place both ON Clauses at the end.

Answer – Can you Place the ON Clauses at the End?

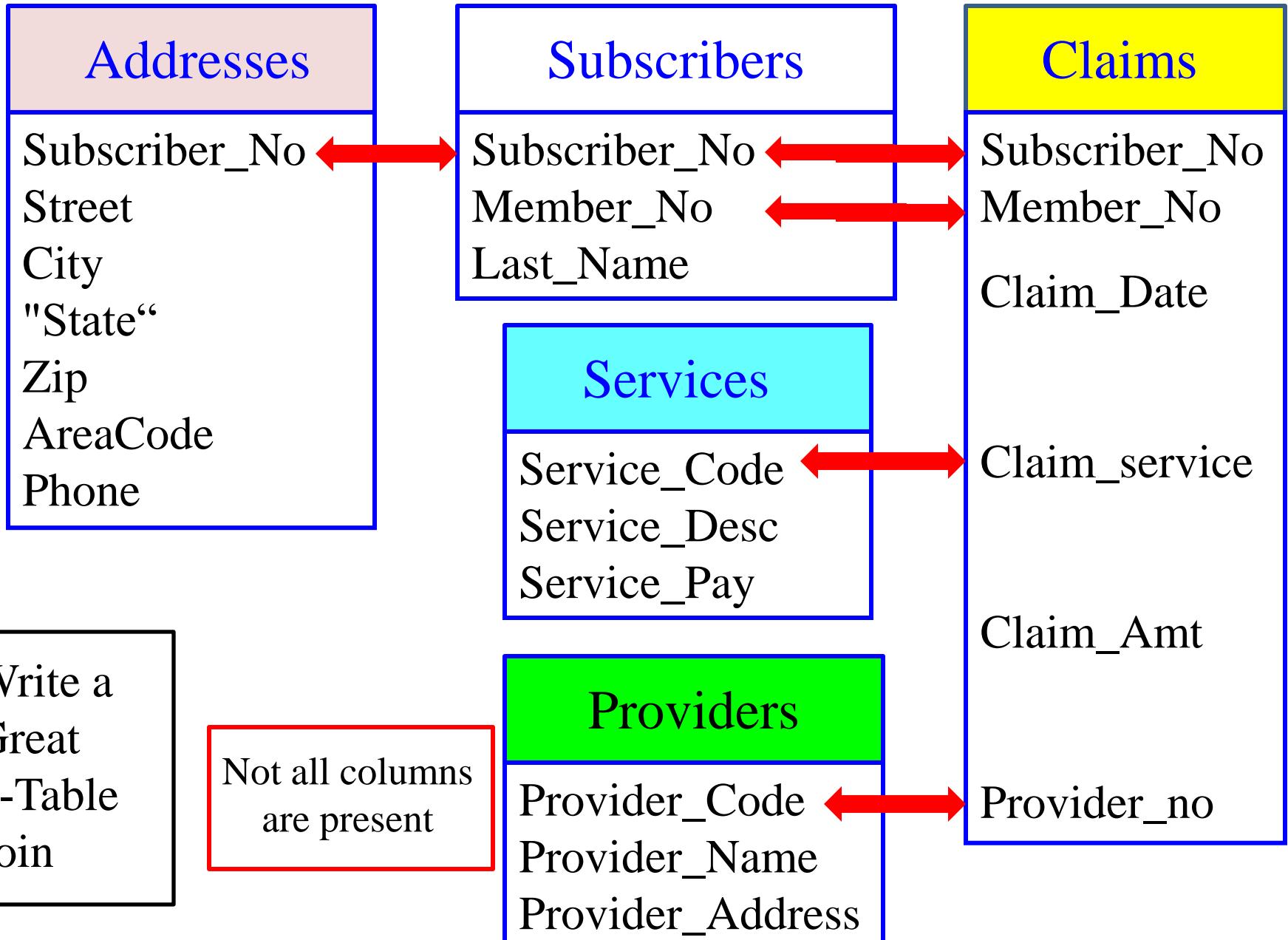


This is tricky. The only way it works is to place the ON clauses backwards. The first ON Clause represents the last INNER JOIN, and then moves backwards.

The 5-Table Join – Logical Insurance Model



The 5-Table Join – Logical Insurance Model



The Nexus Query Chameleon

View Answer Sets Send SQL to Nexus

Objects Columns Sorting Joins Where SQL Answer Set

CLAIMS

Select *

Claim_Id integer

Claim_Date decimal(9,2)

Subscriber_No integer

Member_No smallint

Claim_Amt decimal(12,2)

Provider_No smallint

Claim_Service smallint

Add Join...

SUBSCRIBERS

Select *

Last_Name varchar(20)

First_Name varchar(20)

Gender char(1)

SSN integer

Member_No smallint

Subscriber_No integer

Add Join...

ADDRESSES

Select *

Street varchar(30)

City varchar(20)

State char(2)

Zip integer

AreaCode smallint

Phone integer

Subscriber_No integer

Add Join...

PROVIDERS

Select *

Provider_Code smallint

Provider_Name varchar(30)

P_Address varchar(30)

P_City varchar(20)

P_State char(2)

P_Zip integer

P_Error_Rate decimal(4,4)

Add Join...

SERVICES

Select *

Service_Code smallint

Service_Desc varchar(30)

Service_Pay decimal(7,2)

Add Join...

Nexus knows your logical Model.

This query was built with
3 clicks of the mouse.

The SQL is on the next page

The 5-Table Join ANSI SQL Created by Nexus

SELECT

 cla1.Claim_Id, cla1.Claim_Date, cla1.Subscriber_No, cla1.Member_No,
 cla1.Claim_Amt, cla1.Provider_No, cla1.Claim_Service,
 sub1.Last_Name, sub1.First_Name, sub1.Gender,
 sub1.SSN, sub1.Member_No, sub1.Subscriber_No,
 add1.Street, add1.City, add1.State, add1.Zip, add1.AreaCode,
 add1.Phone, add1.Subscriber_No,
 pro1.Provider_Code, pro1.Provider_Name, pro1.P_Address,
 pro1.P_City, pro1.P_State, pro1.P_Zip, pro1.P_Error_Rate,
 ser1.Service_Code, ser1.Service_Desc, ser1.Service_Pay

FROM SQL_CLASS.CLAIMS AS cla1

INNER JOIN SQL_CLASS.SUBSCRIBERS AS sub1

 ON cla1.Subscriber_No = sub1.Subscriber_No
 AND cla1.Member_No = sub1.Member_No

INNER JOIN SQL_CLASS.ADDRESSES AS add1

 ON sub1.Subscriber_No = add1.Subscriber_No

INNER JOIN SQL_CLASS.PROVIDERS AS pro1

 ON cla1.Provider_No = pro1.Provider_Code

INNER JOIN SQL_CLASS.SERVICES AS ser1

 ON cla1.Claim_Service = ser1.Service_Code;

The 5-Table Join With ON Clauses at END

The screenshot shows the Nexus Query Chameleon application window. The menu bar includes File, Edit, View, Query, Tools, Windows, Web, Help. The toolbar contains various icons for database management tasks. The system dropdown shows 'Teradata' selected, and the database dropdown shows 'sql_class'. The main area has a 'Systems' tree view on the left with nodes for DB2, Greenplum, Netezza, Oracle, Sandbox, SQL_SERVER, and Teradata, which further expands to show tables like Addresses, Claims, Course_table, etc. A central query editor window titled 'Query1*' displays the following SQL code:

```
SELECT      cla1.*, sub1.*, add1.* , pro1.* , ser1.*  
FROM        SQL_CLASS.ADDRESSES AS add1  
INNER JOIN SQL_Class.SUBSCRIBERS AS sub1  
INNER JOIN SQL_CLASS.PROVIDERS AS pro1  
INNER JOIN SQL_CLASS.SERVICES AS ser1  
INNER JOIN SQL_CLASS.CLAIMS AS cla1  
          ON cla1.Claim_Service = ser1.Service_Code  
          ON cla1.Provider_No = pro1.Provider_Code  
          ON cla1.Subscriber_No = sub1.Subscriber_No  
          AND cla1.Member_No = sub1.Member_No  
          ON sub1.Subscriber_No = add1.Subscriber_No;
```

Below the query editor is a results viewer with tabs for 'Messages' and 'Results 1'. The results table has columns: Claim_Id, Claim_Date, Claim_Service, Subscriber_No, Member_No, Claim_Amt, Provider_No, Subscriber_No1, and Member_No1. The data is as follows:

	Claim_Id	Claim_Date	Claim_Service	Subscriber_No	Member_No	Claim_Amt	Provider_No	Subscriber_No1	Member_No1
1	3305444	05/12/2003	111	5555555	2	235.22	3	5555555	
2	3306333	06/28/2003	111	3333333	1	200.50	3	3333333	
3	3402222	02/28/2004	555	2222222	1	550.99	3	2222222	
4	3402222	02/28/2004	999	2222222	1	405.99	3	2222222	
5	3308333	08/01/2003	111	3333333	2	200.00	3	3333333	
6	3306333	06/28/2003	444	3333333	1	1430.66	3	3333333	
7	3306333	06/28/2003	122	3333333	1	201.50	3	3333333	
8	3308333	08/01/2003	122	3333333	2	200.00	3	3333333	
9	1403555	03/01/2004	222	5555555	2	31235.22	3	5555555	

The Join Tab of Nexus

Join Type	Join Objects	
INNER	SQL_CLASS.CLAIMS cla1 JOINS TO SQL_CLASS.SUBSCRIBERS sub1 ON cla1.Subscriber_No = sub1.Subscriber_No AND cla1.Member_No = sub1.Member_No	CLAIMS INNER JOIN SUBSCRIBERS ON cla1.Subscriber_No = sub1.Subscriber_No AND cla1.Member_No = sub1.Member_No
INNER	SQL_CLASS.SUBSCRIBERS sub1 JOINS TO SQL_CLASS.ADDRESSES add1 ON sub1.Subscriber_No = add1.Subscriber_No	INNER JOIN ADDRESSES ON sub1.Subscriber_No = add1.Subscriber_No
INNER	SQL_CLASS.CLAIMS cla1 JOINS TO SQL_CLASS.PROVIDERS pro1 ON cla1.Provider_No = pro1.Provider_Code	INNER JOIN PROVIDERS ON cla1.Provider_No = pro1.Provider_Code
INNER	SQL_CLASS.CLAIMS cla1 JOINS TO SQL_CLASS.SERVICES ser1 ON cla1.Claim_Service = ser1.Service_Code	INNER JOIN SERVICES ON cla1.Claim_Service = ser1.Service_Code

```

SELECT
    cla1.*, sub1.*, add1.*, pro1.*, ser1.*
FROM SQL_CLASS.CLAIMS AS cla1
INNER JOIN SQL_CLASS.SUBSCRIBERS AS sub1
    ON cla1.Subscriber_No = sub1.Subscriber_No
    AND cla1.Member_No = sub1.Member_No
INNER JOIN SQL_CLASS.ADDRESSES AS add1
    ON sub1.Subscriber_No = add1.Subscriber_No
INNER JOIN SQL_CLASS.PROVIDERS AS pro1
    ON cla1.Provider_No = pro1.Provider_Code
INNER JOIN SQL_CLASS.SERVICES AS ser1
    ON cla1.Claim_Service = ser1.Service_Code;

```

The COLUMNS Tab of Nexus

Join Builder (BETA)

View Answer Sets Send SQL to Nexus

Objects Columns Sorting Joins Where SQL Answer Set

OLAP

Drag and drop objects here to remove them from the report.

Columns

Claim_Id	Claim_Date	Subscriber_No	Member_No	Claim_Amt	Provider_No	Claim_Service	Last_Name	First_Name		
Gender	SSN	Member_No	Subscriber_No	Street	City	State	Zip	AreaCode	Phone	Subscriber_N
Provider_Code	Provider_Name	P_Address	P_City	P_State	P_Zip	P_Error_Rate	Service_Code	Service_Desc		
Service_Pay										

SELECT SQL

SELECT

cla1.Claim_Id,	cla1.Claim_Date,	cla1.Subscriber_No,	cla1.Member_No,
cla1.Claim_Amt,	cla1.Provider_No,	cla1.Claim_Service,	
sub1.Last_Name,	sub1.First_Name,	sub1.Gender,	sub1.SSN,
sub1.Member_No,	sub1.Subscriber_No,		
add1.Street,	add1.City,	add1.State,	add1.Zip,
add1.AreaCode,	add1.Phone,	add1.Subscriber_No,	
pro1.Provider_Code,	pro1.Provider_Name,	pro1.P_Address,	pro1.P_City,
pro1.P_State,	pro1.P_Zip,	pro1.P_Error_Rate,	
ser1.Service_Code,	ser1.Service_Desc,	ser1.Service_Pay	

Chapter 19

Date Functions

“An inch of time cannot be bought with an inch of gold.”

- Chinese Proverb

Table of Contents Chapter 19 - Date Functions

- [Dates are stored Internally as INTEGERS from a Formula](#)
- [Date, Time, and Timestamp Keywords](#)
- [Displaying Dates for INTEGERDATE and ANSIDATE](#)
- [DATEFORM](#)
- [Changing the DATEFORM in Client Utilities such as BTEQ](#)
- [Date, Time, and Timestamp Recap](#)
- [Timestamp Differences](#)
- [Troubleshooting Timestamp](#)
- [Add or Subtract Days from a date](#)
- [A Summary of Math Operations on Dates](#)
- [Using a Math Operation to find your Age in Years](#)
- [Find What Day of the week you were Born](#)
- [The ADD_MONTHS Command](#)
- [Using the ADD_MONTHS Command to Add 1-Year](#)
- [Using the ADD_MONTHS Command to Add 5-Years](#)
- [The EXTRACT Command](#)
- [EXTRACT from DATES and TIME](#)
- [CURRENT_DATE and EXTRACT or Current Date and Math](#)
- [CAST the Date of January 1, 2011 and the Year 1800](#)
- [The System Calendar](#)
- [How to really use the Sys Calendar.Calendar](#)
- [Storing Dates Internally](#)
- [Storing Time Internally](#)
- [Storing TIME With TIME ZONE Internally](#)

Continued on next page

Table of Contents Chapter 19 - Date Functions Continued

- [Storing Timestamp Internally](#)
- [Storing Timestamp with TIME ZONE Internally](#)
- [Storing Date, Time, Timestamp with Zone Internally](#)
- [Time Zones](#)
- [Setting Time Zones](#)
- [Seeing your Time Zone](#)
- [Creating a Sample Table for Time Zone Examples](#)
- [Inserting Rows in the Sample Table for Time Zone Examples](#)
- [Selecting the Data from our Time Zone Table](#)
- [Normalizing our Time Zone Table with a CAST](#)
- [Intervals for Date, Time and Timestamp](#)
- [Interval Data Types and the Bytes to Store Them](#)
- [The Basics of a Simple Interval](#)
- [Troubleshooting The Basics of a Simple Interval](#)
- [Interval Arithmetic Results](#)
- [A Date Interval Example](#)
- [A Time Interval Example](#)
- [A - DATE Interval Example](#)
- [A Complex Time Interval Example using CAST](#)
- [A Complex Time Interval Example using CAST](#)
- [The OVERLAPS Command](#)
- [An OVERLAPS Example that Returns No Rows](#)
- [The OVERLAPS Command using TIME](#)
- [The OVERLAPS Command using a NULL Value](#)

Dates are stored Internally as INTEGERS from a Formula

INTEGERDATE = ((Year – 1900) * 10000) + (Month * 100) + Day

/* Example – Tom’s Birthday January 10, 1959 */

$$\begin{aligned}\text{INTEGERDATE} &= ((1959 - 1900) = 59 && \text{Year Portion} \\ &\quad * 10000) = 590000 \\ &\quad + (\text{Month} * 100) = 590100 && \text{Month Portion} \\ &\quad + \text{Day} = 590110 && \text{Day Portion}\end{aligned}$$

/* Example – Tom’s Birthday January 10, 1999 */

990110

/* Example – Tom’s Birthday January 10, 2000 */

1000110

The reason the Smart Calendar works so well is that it stores EVERY date in Teradata as something known as an INTEGERDATE.

Date, Time, and Current_Timestamp Keywords

```
SELECT Date AS "Date"  
      ,Current_Date AS ANSI_Date  
      ,Time AS "Time"  
      ,Current_Time AS ANSI_Time  
      ,Current_Timestamp(6) AS ANSI_Timestamp
```

Answer Set

<u>Date</u>	<u>ANSI Date</u>	<u>Time</u>	<u>ANSI Time</u>	<u>ANSI Timestamp</u>
2011/03/22	2011/03/22	10:34:44	10:34:44	2011/03/22 10:34:44.123456 -04:00

There's **no** keyword **Timestamp**, but only ANSI's **Current_Timestamp**

Above are the keywords you can utilize to get the date, time, or timestamp. These are reserved words that the system will deliver to you when requested.

Displaying Dates for INTEGERDATE and ANSIDATE

SELECT Date ,Current_Date	AS "Date" AS ANSI_Date
------------------------------	---------------------------

INTEGERDATE (YY/MM/DD)

June 30, 2012

Date	ANSI_Date
12/06/30	12/06/30

ANSIDATE (YYYY-MM-DD)

June 30, 2012

Date	ANSI_Date
2012-06-30	2012-06-30

NEXUS Query Chameleon MM-DD-YYYY

Date	ANSI_Date
06-30-2012	06-30-2012

Teradata in release V2R3 defaulted to a display of YY/MM/DD. This is called the INTEGERDATE. This can be changed to ANSIDATE, which is YYYY-MM-DD for a specific session or by Default if the DBA changes the DATEFORM in DBS Control. This has nothing to do with how the date is stored internally. It has to do with the display of dates when using any ODBC tool or load utility. Above are some examples.

DATEFORM

DATEFORM Controls the default display of dates.

DATEFORM display choices are either INTEGERDATE or ANSIDATE.

INTEGERDATE is (YY/MM/DD) and ANSIDATE is (YYYY-MM-DD).

DATEFORM is the expected format for import and export of dates in Load Utilities.

Can be over-ridden by USER or within a Session at any time.

The Default can be changed by the DBA by changing the DATEFORM in DBSControl.

INTEGERDATE (YY/MM/DD)

June 30, 2012

Date	ANSI_Date
12/06/30	12/06/30

ANSIDATE (YYYY-MM-DD)

June 30, 2012

Date	ANSI_Date
2012-06-30	2012-06-30

Teradata in release V2R3 defaulted to a display of YY/MM/DD. This is called the INTEGERDATE. This can be changed to ANSIDATE, which is YYYY-MM-DD for a specific session or by Default if the DBA changes the DATEFORM in DBS Control. This has nothing to do with how the date is stored internally. It has to do with the display of dates when using any ODBC tool or load utility.

Changing the DATEFORM in Client Utilities such as BTEQ

Enter your logon or BTEQ Command:

.logon localtd/dbc

Password: *****

Logon successfully completed

BTEQ – Enter your DBC/SQL request or BTEQ command:

SELECT DATE;

Date

12/06/30

← INTEGERDATE is the Default

BTEQ – Enter your DBC/SQL request or BTEQ command:

SET Session DATEFORM = ANSIDATE;

← Changing the DATEFORM
for this BTEQ session.

SELECT DATE;

Notice the
Word 'Current_Date'

Current Date

2012-06-30

← ANSIDATE is the Display Form

Date, Time, and Timestamp Recap

```
SELECT Date          AS "Date"  
      ,Current_Date  AS ANSI_Date
```

INTEGERDATE (YY/MM/DD)

June 30, 2012

Date	ANSI_Date
12/06/30	12/06/30

ANSIDATE (YYYY-MM-DD)

June 30, 2012

Date	ANSI_Date
2012-06-30	2012-06-30

Dates are converted to an **integer** through a formula before being **stored**.

Dates are **displayed** by **default** as **INTEGERDATE YY/MM/DD**.

The DBA can set up the system to display as **ANSIDATE YYYY-MM-DD**.

Keywords **Date** or **Current_Date** will return the date automatically.

Time, **Current_Time** and **Current_Timestamp** are keywords.

The **Nexus** Query Chameleon displays dates as **MM-DD-YYYY**.

Timestamp Differences

```
SELECT Current_Timestamp(0) AS Col1  
      ,Current_Timestamp(6) AS Col2
```

Answer Set

Col1	Col2
2011/03/22 10:34:44	2011/03/22 10:34:44.123456

 Date  Space  Time Milliseconds

A timestamp has the date separated by a space and the time. In our second example we have asked for 6 milliseconds.

Troubleshooting Timestamp

```
SELECT Timestamp(0) AS Col1  
      , Timestamp(6) AS Col2
```

Error

There is Date and Current_Date (both work).

There is Time and Current_Time (both work).

There is **NO** Timestamp, but only **Current_Timestamp!**

There is **NO** Timestamp KEYWORD, but only ANSI's Current_Timestamp!

Add or Subtract Days from a date

```
SELECT Order_Date  
      ,Order_Date + 60 as "Due Date"  
      ,Order_Total  
      , "Due date" -10  as Discount  
      ,Order_Total *.98 (FORMAT '$$$$$,$$$$.99', Title 'Discounted')  
FROM   Order_Table  
ORDER BY 1 ;
```

Order_Date	Due Date	Order_Total	Discount	Discounted
05/04/1998	07/03/1998	12347.53	06/23/1998	12100.57
01/01/1999	03/02/1999	8005.91	02/20/1999	7845.79
09/09/1999	11/08/1999	23454.84	10/29/1999	22985.74
10/01/1999	11/30/1999	5111.47	11/20/1999	5009.24
10/10/1999	12/09/1999	15231.62	11/29/1999	14926.98

When you add or subtract from a Date you are adding/subtracting Days

Because Dates are stored internally on disk as integers it makes it easy to add days to the calendar. In the query above we are adding 60 days to the Order_Date.

A Summary of Math Operations on Dates

- ① DATE - DATE = Interval (days between dates)
- ② DATE + or - Integer = Date

Let's find the number of days Tera-Tom has been alive since his last birthday.

```
SELECT (1120110(date)) - (590110 (date)) (Title 'Tera-Tom''s Age In Days');
```

Tera-Tom's Age In Days

19358

Below is the same exact query, but with a clearer example of the dates.

```
SELECT ('2012-01-10' (date)) - ('1959-01-10' (date)) (Title 'Tera-Tom''s Age In Days');
```

Tera-Tom's Age In Days

19358

A DATE – DATE is an interval of days between dates. A DATE + or – Integer = Date. Both queries above perform the same function, but the top query uses the internal date functions and the query on the bottom does dates the traditional way.

Using a Math Operation to find your Age in Years

- ① DATE – DATE = Interval (days between dates)
- ② DATE + or - Integer = Date

Let's find the number of **days** Tera-Tom has been alive since his last birthday.

```
SELECT (1120110(date)) - (590110 (date)) (Title 'Tera-Tom''s Age In Days');
```

Tera-Tom's Age In Days

19358

Let's find the number of **years** Tera-Tom has been alive since his last birthday.

```
SELECT ((1120110(date)) - (590110 (date))) / 365 (Title 'Tera-Tom's Age In Years');
```

Tera-Tom's Age In Years

53

A DATE – DATE is an interval of days between dates. A DATE + or – Integer = Date. Both queries above perform a Date function, but the top query brings back Tom's age in days and the bottom query brings back Tom's age in years..

Find What Day of the week you were Born

Let's find the actual day of the week Tera-Tom was born

SEL 'Tera-Tom was born on day ' || ((590110(date)) - (101(date))) MOD 7 (TITLE '');

Tera-Tom was born on day 5

This will produce
No Title

Result	Day of the Week
0	Monday
1	Tuesday
2	Wednesday
3	Thursday
4	Friday
5	Saturday
6	Sunday

This chart can be used
In conjunction with the
above SQL

The above subtraction results in the number of days between the two dates. Then, the MOD 7 divides by 7 to get rid of the number of weeks and results in the remainder. A MOD 7 can only result in values 0 thru 6 (always 1 less than the MOD operator). Since January 1, 1900 (101(date)) is a Monday, Tom was born on a Saturday.

The ADD_MONTHS Command

Order_Table

Order_Number	Customer_Number	Order_Date	Order_Total
123456	11111111	12347.53	1998/05/04
123512	11111111	8005.91	1999/01/01
123552	31323134	5111.47	1999/10/01
123585	87323456	15231.62	1999/10/10
123777	57896883	23454.84	1999/09/09

```
SELECT Order_Date  
      ,Add_Months(Order_Date,2) as "Due Date"  
      ,Order_Total  
FROM   Order_Table ORDER BY 1 ;
```

Order_Date	Due Date	Order_Total
05/04/1998	07/04/1998	12347.53
01/01/1999	03/01/1999	8005.91
09/09/1999	11/09/1999	23454.84
10/01/1999	12/01/1999	5111.47
10/10/1999	12/10/1999	15231.62

This is the Add_Months Command. What you can do with it is add a month or many months your columns date. Can you convert this to one year?

Using the ADD_MONTHS Command to Add 1 Year

Order_Table			
Order_Number	Customer_Number	Order_Date	Order_Total
123456	11111111	12347.53	1998/05/04
123512	11111111	8005.91	1999/01/01
123552	31323134	5111.47	1999/10/01
123585	87323456	15231.62	1999/10/10
123777	57896883	23454.84	1999/09/09

```
SELECT Order_Date  
      ,Add_Months(Order_Date,12) as "Due Date"  
      ,Order_Total  
FROM   Order_Table  
ORDER BY 1 ;
```



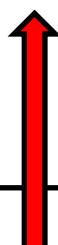
There is **no** Add_Year command, so put in **12** months for 1-year

The Add_Months command adds months to any date. Above we used a great technique that would give us 1 year. Can you give me **5 years**?

Using the ADD_MONTHS Command to Add 5 Years

Order_Table			
Order_Number	Customer_Number	Order_Date	Order_Total
123456	11111111	12347.53	1998/05/04
123512	11111111	8005.91	1999/01/01
123552	31323134	5111.47	1999/10/01
123585	87323456	15231.62	1999/10/10
123777	57896883	23454.84	1999/09/09

```
SELECT Order_Date  
      ,Add_Months(Order_Date,12 * 5) as "Due Date"  
      ,Order_Total  
FROM   Order_Table  
ORDER BY 1 ;
```



In this example we multiplied **12 months** times **5** for a total of **5 years!**

Above you see a great technique for adding multiple years to a date. Can you now SELECT only the orders in September?

The EXTRACT Command

Order_Table			
<u>Order_Number</u>	<u>Customer_Number</u>	<u>Order_Date</u>	<u>Order_Total</u>
123456	11111111	12347.53	1998/05/04
123512	11111111	8005.91	1999/01/01
123552	31323134	5111.47	1999/10/01
123585	87323456	15231.62	1999/10/10
123777	57896883	23454.84	1999/09/09

```
SELECT Order_Date  
      ,Add_Months (Order_Date,12 * 5) as "Due Date"  
      ,Order_Total  
FROM   Order_Table  
WHERE  EXTRACT(Month from Order_Date) = 09  
ORDER BY 1 ;
```

The EXTRACT command **extracts** portions of Date, Time, and Timestamp.

This is the Extract command. It extracts a portion of the date and it can be used in the SELECT list or the WHERE Clause, or the ORDER BY Clause!

EXTRACT from DATES and TIME

```
SELECT Current_Date  
      ,EXTRACT(Year from Current_Date) as Yr  
      ,EXTRACT(Month from Current_Date) as Mo  
      ,EXTRACT(Day from Current_Date) as Da  
      ,Current_Time  
      ,EXTRACT(Hour from Current_Time) as Hr  
      ,EXTRACT(Minute from Current_Time) as Mn  
      ,EXTRACT(Second from Current_Time) as Sc  
      ,EXTRACT(TIMEZONE_HOUR from Current_Time) as Th  
      ,EXTRACT(TimeZONE_MINUTE from Current_Time) as Tm
```

Answer Set

Current Date	Yr	Mo	Da	Current Time (0)	Hr	Mn	Sc	Th	Tm
2011/03/22	2011	03	22	20:01:14 20	1	14	0	0	0

Just like the Add_Months, the EXTRACT Command is a **Temporal Function** or a Time-Based Function.

CURRENT_DATE and EXTRACT or Current_Date and Math

```
SELECT Current_Date  
      ,EXTRACT(Year from Current_Date) as Yr  
      ,EXTRACT(Month from Current_Date) as Mo  
      ,EXTRACT(Day from Current_Date) as Da  
      ,Current_Date / 10000 +1900 as YrMath  
      ,(Current_Date / 100) Mod 100 as MoMath  
      ,Current_Date Mod 100 as DayMath ;
```

Math can be used to extract portions of a Date!

Answer Set

<u>Current Date</u>	<u>Yr</u>	<u>Mo</u>	<u>Day</u>	<u>YrMath</u>	<u>MoMath</u>	<u>DayMath</u>
2011/03/22	2011	03	22	2011	03	22

The Extract Temporal Function can be used to extract a portion of a date. As you can see, Basic Arithmetic accomplishes the same thing.

CAST the Date of January 1, 2011 and the Year 1800

```
SELECT
    cast('2011-01-01' as date)    as ANSI_Literal
    ,cast(1110101 as date)        as INTEGER_Literal
    ,cast('11-01-01' as date)     as YY_Literal
    ,cast(Date '2011-01-01' as Integer) as Dates_Stored
    ,cast(Date '1800-01-01' as Integer) as Dates_1800s
```

Answer Set

ANSI_Literal	INTEGER_Literal	YY_Literal	Dates_Stored	Dates_1800s
01/01/2011	01/01/2011	01/01/1911	111010	-999899

The Convert And Store (CAST) command is used to give columns a different data type temporarily for the life of the query. Notice our dates and how they're stored.

The System Calendar

Teradata systems have a **table** called **Caldates**.

Caldates has only one column in it called **Cdates**.

Cdates is a date column that contains a row for each date starting from January 1, **1900** to December 31, **2100**.

No user can access the table Caldates directly.

Views in the **Sys_Calendar** database accesses Caldates.

A **view** called **Calendar** is how USERS work with the calendar.

Users use **Sys_Calendar.Calendar** for advanced dates.

In every Teradata system, there is something known as a **System Calendar** (or as Teradata calls it **Sys_Calendar.Calendar**). Get ready for AWESOME!

```
SELECT * FROM Sys_Calendar.Calendar  
WHERE Calendar_Date = '1959-01-10';
```

Birthday of
Tera-Tom

Calendar_Date = 01/10/1959'

day_of_week = 7 (Sunday = 1)

day_of_month = 10

day_of_year = 10

day_of_Calendar = 21559 (since Jan 1, 1900)

weekday_of_month = 2

week_of_month = 1 (0 for partial week for any month not starting with Sunday)

week_of_year = 1

week_of_calendar = 3079 (since Jan 1, 1900)

month_of_quarter = 1

month_of_year = 1

month_of_calendar = 709 (since Jan 1, 1900)

quarter_of_year = 1

quarter_of_calender = 237 (since Jan 1, 1900)

year_of_calendar = 1959

Tera-Tom was born on a Saturday! It was the first full week of the month, the first full week of the year and it was the first quarter of the year!

How to really use the Sys_Calendar.Calendar

```
SELECT O.*  
FROM Order_Table as O  
INNER JOIN  
    Sys_Calendar.Calendar  
ON Order_Date = Calendar_Date ←   
AND Quarter_of_Year = 4  
AND Day_of_Week = 6  
AND Week_of_Month = 0;
```

Join a date column
with the
Calendar_Date

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

We just brought back all Orders from the Order_Table that were purchased on a **Friday** in the **4th Quarter**, during the **1st partial week**. This means **no Sunday seen yet** for that month.

Above is the perfect example of how you can utilize the Sys_Calendar.Calendar to join to any date field and then expand your search options.

Storing Dates Internally

```
CREATE SET TABLE TIMEZONE_table ,FALLBACK ,
    NO BEFORE JOURNAL,
    NO AFTER JOURNAL ,
CHECKSUM = DEFAULT
    (Date_col          Date,
     TIME_col           TIME(6),
     TIMETIMEZONE_col TIME(6) WITH TIME ZONE,
     TIMESTAMP_col     TIMESTAMP(6),
     TIMEZONE_col      TIMESTAMP(6) WITH TIME ZONE)
UNIQUE PRIMARY INDEX ( TIMEZONE_col );
```

DATE ‘1999-01-10’ is stored as 990110

DATE ‘2000-01-10’ is stored as 1000110

4 bytes store Date_col internally because dates are considered a 4-byte integer.

Storing Time Internally

```
CREATE SET TABLE TIMEZONE_table ,FALLBACK ,
    NO BEFORE JOURNAL,
    NO AFTER JOURNAL,
    CHECKSUM = DEFAULT
    (Date_col                  Date,
     TIME_col                  TIME(6),
     TIMETIMEZONE_col          TIME(6) WITH TIME ZONE,
     TIMESTAMP_col              TIMESTAMP(6),
     TIMEZONE_col              TIMESTAMP(6) WITH TIME ZONE)
UNIQUE PRIMARY INDEX ( TIMEZONE_col );
```

Time(n) stored as HHMMSS.nnnnnn

It takes 6 bytes to store Time_col internally.

Storing TIME With TIME ZONE Internally

```
CREATE SET TABLE TIMEZONE_table ,FALLBACK ,
    NO BEFORE JOURNAL,
    NO AFTER JOURNAL ,
CHECKSUM = DEFAULT
    (Date_col           Date,
     TIME_col          TIME(6),
     TIMETIMEZONE_col TIME(6) WITH TIME ZONE,
     TIMESTAMP_col    TIMESTAMP(6),
     TIMEZONE_col     TIMESTAMP(6) WITH TIME ZONE)
UNIQUE PRIMARY INDEX ( TIMEZONE_col );
```

Time(n) WITH ZONE stored as HHMMSS.nnnnnn+HHMM

It takes 8 bytes to store TimeTimezone_col internally.

Storing Timestamp Internally

```
CREATE SET TABLE TIMEZONE_table ,FALLBACK ,
    NO BEFORE JOURNAL,
    NO AFTER JOURNAL ,
    CHECKSUM = DEFAULT
        (Date_col                  Date,
         TIME_col                  TIME(6),
         TIMETIMEZONE_col          TIME(6) WITH TIME ZONE,
         TIMESTAMP_col           TIMESTAMP(6),
         TIMEZONE_col              TIMESTAMP(6) WITH TIME ZONE)
UNIQUE PRIMARY INDEX ( TIMEZONE_col );
```

TimeStamp(n) stored as YYMMDDHHMMSS.nnnnnn

It takes **10 bytes** to store TimeStamp_col internally.

Storing Timestamp with TIME ZONE Internally

```
CREATE SET TABLE TIMEZONE_table ,FALLBACK ,
    NO BEFORE JOURNAL,
    NO AFTER JOURNAL ,
CHECKSUM = DEFAULT
    (Date_col                  Date,
     TIME_col                  TIME(6),
     TIMETIMEZONE_col          TIME(6) WITH TIME ZONE,
     TIMESTAMP_col              TIMESTAMP(6),
     TIMEZONE_col              TIMESTAMP(6) WITH TIME ZONE)
UNIQUE PRIMARY INDEX ( TIMEZONE_col );
```

TimeStamp(n) With Zone stored as
YYMMDDHHMMSS.nnnnnn+HHMM

It will take 12 bytes to store Timezone_col internally.

Storing Date, Time, and Timestamp with Zone Internally

```
CREATE SET TABLE TIMEZONE_table ,FALLBACK ,
    NO BEFORE JOURNAL,
    NO AFTER JOURNAL ,
    CHECKSUM = DEFAULT
    (Date_col           Date,
     TIME_col           TIME(6),
     TIMETIMEZONE_col  TIME(6) WITH TIME ZONE,
     TIMESTAMP_col      TIMESTAMP(6),
     TIMEZONE_col       TIMESTAMP(6) WITH TIME ZONE)
UNIQUE PRIMARY INDEX ( TIMEZONE_col );
```

Date	Stored Internally	4 Bytes
Time(n)	Stored Internally	6 Bytes
Time(n) With Zone	Stored Internally	8 Bytes
Timestamp(n)	Stored Internally	10 Bytes
Timestamp(n) with zone	Stored Internally	12 Bytes

Each data type increase its internal storage by 2 bytes.

Time Zones

A time zone relative to [London](#) (UTC) might be:

LA-----	Miami-----	Frankfurt-----	Hong Kong
+8:00	+05:00	00:00	-08:00

A time zone relative to [New York](#) (EST) might be:

LA-----	Miami-----	Frankfurt-----	Hong Kong
+3:00	00:00	-05:00	-13:00

Time zones are set either at the system level (DBS Control), the user level (when user is created or modified), or at the session level as an override.

Teradata has the ability to access and store both the hours and the minutes reflecting the difference between the user's time zone and the system time zone. From a World perspective, this difference is normally the number of hours between a specific location on Earth and the United Kingdom location that was historically called Greenwich Mean Time (GMT). Since the Greenwich observatory has been "decommissioned," the new reference to this same time zone is called Universal Time Coordinate (UTC).

Setting Time Zones

A Time Zone should be established for the system and every user in each different time zone.

Setting the **system default** time zone is done by the DBA in the **DBSControl record**:

```
MODIFY GENERAL 16 = x /* Hours, n= -12 to 13 */
MODIFY GENERAL 17 = x /* Minutes, n = -59 to 59 */
```

Setting a **User's** time zone requires choosing either **LOCAL**, **NULL**, or an **explicit value**:

```
CREATE USER Tera-Tom
TIME ZONE = LOCAL /* use system level */
              = NULL /* no default, set to system or session level at logon */
              = '16:00' /* explicit setting */
              = '-06:30' /* explicit setting */
```

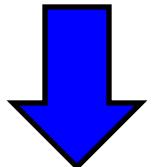
Setting a **Session's** time zone:

```
SET TIME ZONE LOCAL; /* use system level */
SET TIME ZONE USER; /* use user level */
SET TIME ZONE INTERVAL '08:00' HOUR TO MINUTE; /* explicit setting */
```

A Teradata session can modify the time zone without requiring a logoff and logon.

Seeing your Time Zone

Help Session ;



User Name	Account Name	Logon Date	Logon Time	Current Database	Collation	Char Set	Transaction Semantics	Current Dateform	Session Time Zone
DBC	DBC	12/06/17	15:55:39	SQL_CLASS	ASCII	ASCII	Teradata	IntegerDate	00:00

Not all output
is displayed
above from the
HELP Session

A user's time zone is now part of the information maintained by Teradata. The settings can be seen in the extended information available in the HELP SESSION request. Teradata converts all TIME and TIMESTAMP values to Universal Time Coordinate (UTC) prior to storing them. All operations, including hashing, collation, and comparisons that act on TIME and TIMESTAMP values are performed using their UTC forms. This will allow users to CAST the information to their local times.

Creating a Sample Table for Time Zone Examples

```
CREATE TABLE Tstamp_Test  
(  
    TS_Zone CHAR(3)  
    ,TS_with_Zone TIMESTAMP(6) WITH TIME ZONE  
    ,TS_Without_Zone TIMESTAMP(6)  
)  
UNIQUE PRIMARY INDEX ( TS_Zone );
```

Not all output
is displayed
above from the
HELP Session

A user's time zone is now part of the information maintained by Teradata. The settings can be seen in the extended information available in the HELP SESSION request.

Inserting Rows in the Sample Table for Time Zone Examples

Enter your logon or BTEQ Command:

.logon localtd/dbc

Password: *****

Logon successfully completed

BTEQ – Enter your DBC/SQL request or BTEQ command:

SET TIME ZONE INTERVAL ‘05:00’ HOUR TO MINUTE :

SET TIME ZONE INTERVAL - '03:00' HOUR TO MINUTE :

SET TIME ZONE INTERVAL - '11:00' HOUR TO MINUTE :

Selecting the Data from our Time Zone Table

```
SELECT * FROM Tstamp_Test ;
```

TS_Zone	TS_With_Zone	TS_Without_Zone
UTC	2000-10-01 08:12:00.000000+05:00	2000-10-01 08:12:00.000000
EST	2000-10-01 08:12:00.000000+00:00	2000-10-01 08:12:00.000000
PST	2000-10-01 08:12:00.000000-03:00	2000-10-01 08:12:00.000000
HKT	2000-10-01 08:12:00.000000-11:00	2000-10-01 08:12:00.000000



Notice the
Accompanying
Time Zone Offsets

Our Insert statements were done at 08:12:00 exactly. Notice the Time Zone offsets in the column TS_With_Zone and how they're not there for the column TS_Without_Zone. Teradata converts all TIME and TIMESTAMP values to Universal Time Coordinate (UTC) prior to storing them. All operations, including hashing, collation, and comparisons that act on TIME and TIMESTAMP values are performed using their UTC forms. This will allow users to CAST the information to their local times.

Normalizing our Time Zone Table with a CAST

```
SELECT TS_Zone, TS_with_Zone  
      ,CAST(TS_with_Zone AS TIMESTAMP(6)) AS T_Normal  
FROM Tstamp_Test ORDER BY 3 ;
```

TS_Zone	TS_with_Zone	T_Normal
UTC	2000-10-01 08:12:00.000000+05:00	2000-10-01 03:12:00.000000
EST	2000-10-01 08:12:00.000000+00:00	2000-10-01 08:12:00.000000
PST	2000-10-01 08:12:00.000000-03:00	2000-10-01 11:12:00.000000
HKT	2000-10-01 08:12:00.000000-11:00	2000-10-01 19:12:00.000000



The System is on EST Time. The New Times are Normalized to the time zone of the System!

Notice that the Time Zone value was added to or subtracted from the time portion of the time stamp to adjust it to a perspective of the same time zone. As a result, at that moment, it has normalized the different Times Zones in respect to the system time.

As an illustration, when the transaction occurred at 8:12 AM locally in the PST Time Zone, it was already 11:12 AM in EST, the location of the system. The times in the columns have been **normalized** in respect to the time zone of the **system**.

Intervals for Date, Time and Timestamp

Interval Chart

Simple Intervals	More involved Intervals
YEAR	DAY TO HOUR
MONTH	DAY TO MINUTE
DAY	DAY TO SECOND
HOUR	HOUR TO MINUTE
MINUTE	HOUR TO SECOND
SECOND	MINUTE TO SECOND

To make Teradata SQL more ANSI compliant and compatible with other RDBMS SQL, Teradata has added INTERVAL processing. Intervals are used to perform DATE, TIME and TIMESTAMP arithmetic and conversion.

Although Teradata allowed arithmetic on DATE and TIME, it was not performed in accordance to ANSI standards and therefore, an extension instead of a standard. With INTERVAL being a standard instead of an extension, more SQL can be ported directly from an ANSI compliant database to Teradata without conversion.

Interval Data Types and the Bytes to Store Them

Interval Chart

Bytes	Data Type	Comments
2	INTERVAL YEAR	
4	INTERVAL YEAR TO MONTH	
2	INTERVAL MONTH	
2	INTERVAL MONTH TO DAY	
2	INTERVAL DAY	
8	INTERVAL DAY TO MINUTE	
10/12	INTERVAL DAY TO SECOND	10 for 32-bit systems; 12 for 64-bit
2	INTERVAL HOUR 2	
4	INTERVAL HOUR TO MINUTE 4	
8	INTERVAL HOUR TO SECOND 8	
2	INTERVAL MINUTE 2	
6/8	INTERVAL MINUTE TO SECOND	6 for 32-bit systems; 8 for 64-bit
6/8	INTERVAL SECOND	6 for 32-bit systems; 8 for 64-bit

The Basics of a Simple Interval

```
SELECT Current_Date as Our_Date  
      ,Current_Date + Interval '1' Day      as Plus_1_Day  
      ,Current_Date + Interval '3' Month    as Plus_3_Months  
      ,Current_Date + Interval '5' Year     as Plus_5_Years
```

Our_Date	Plus_1_Day	Plus_3_Months	Plus_5_Years
06/18/2012	06/19/2012	09/18/2012	06/18/2017

In the example SQL above we take a simple date and add 1 day, 3 months and 5 years. Notice that our current_date is 06/18/2012 and that our intervals come out perfectly.

Troubleshooting The Basics of a Simple Interval

```
SELECT Date '2012-01-29' as Our_Date  
      ,Date '2012-01-29' + INTERVAL '1' Month as Leap_Year
```

Our_Date	Leap_Year
01/29/2012	02/29/2012

```
SELECT Date '2011-01-29' as Our_Date  
      ,Date '2011-01-29' + INTERVAL '1' Month as Leap_Year
```

Error – Invalid Date

The first example works because we added 1 month to the date ‘2012-01-29’ and we got ‘2012-02-29’. Because this was a leap year there actually is a date of February 29, 2012. The next example is the real point. We have a date of ‘2011-01-29’ and we add 1-month to that, but there is no February 29th in 2011 so the query fails.

Interval Arithmetic Results

DATE and TIME arithmetic Results using intervals:

DATE	- DATE	= Interval
TIME	- TIME	= Interval
TIMESTAMP	- TIMESTAMP	= Interval
DATE	- or + Interval	= DATE
TIME	- or + Interval	= TIME
TIMESTAMP	- or + Interval	= TIMESTAMP
Interval	- or + Interval	= Interval

To use DATE and TIME arithmetic, it is important to keep in mind the results of various operations. The above chart is your Interval guide.

A Date Interval Example

```
SELECT (DATE '1999-10-01' - DATE '1988-10-01') DAY AS Actual_Days ;
```

ERROR – Interval Field Overflow

The **Error** occurred because the default for all intervals is **2** digits.

```
SELECT (DATE '1999-10-01' - DATE '1988-10-01') DAY(4) AS Actual_Days ;
```

Makes the output 4 digits

Actual_Days

4017

The default for all intervals is 2 digits. We received an overflow error because the Actual_Days is 4017. The second example works because we declared the output to be 4 digits (the maximum for intervals).

A Time Interval Example

Makes the output 3 digits

```
SELECT (TIME '12:45:01' - TIME '10:10:01') HOUR AS Actual_Hours  
,(TIME '12:45:01' - TIME '10:10:01') MINUTE(3) AS Actual_Minutes  
,(TIME '12:45:01' - TIME '10:10:01') SECOND(4) AS Actual_Seconds  
,(TIME '12:45:01' - TIME '10:10:01') SECOND(4,4) AS Actual_Seconds4
```

Actual_Hours	Actual_Minutes	Actual_Seconds	Actual_Seconds4
2	155	9300.000000	9300.0000

```
SELECT (TIME '12:45:01' - TIME '10:10:01') HOUR AS Actual_Hours  
,(TIME '12:45:01' - TIME '10:10:01') MINUTE AS Actual_Minutes  
,(TIME '12:45:01' - TIME '10:10:01') SECOND(4) AS Actual_Seconds  
,(TIME '12:45:01' - TIME '10:10:01') SECOND(4,4) AS Actual_Seconds4
```

ERROR – Interval Field Overflow

The default for all intervals is 2 digits, but notice in the top example we put in 3 digits for Minute, 4 digits for Second and 4,4 digits for the Actual_Seconds4. If we had not we would have received an overflow error as in the bottom example.

A - DATE Interval Example

```
SELECT Current_Date,  
       INTERVAL - '2' YEAR + CURRENT_DATE as Two_years_Ago;
```

Date	Two_Year_Ago
06/18/2012	06/18/2010

The above Interval example uses a - '2' to go back in time.

A Complex Time Interval Example using CAST

Below is the syntax for using the CAST with a date:

```
SELECT CAST (<interval> AS INTERVAL <interval> )
FROM <table-name>;
```

The following converts an INTERVAL of 6 years and 2 months to an INTERVAL number of months:

```
SELECT CAST( (INTERVAL '6-02' YEAR TO MONTH) AS INTERVAL MONTH );
```

$$\frac{6-02}{74}$$

The CAST function (Convert And Store) is the ANSI method for converting data from one type to another. It can also be used to convert one INTERVAL to another INTERVAL representation. Although the CAST is normally used in the SELECT list, it works in the WHERE clause for comparison purposes.

A Complex Time Interval Example using CAST

This request attempts to convert 1300 months to show the number of years and months.
Why does it fail?

```
SELECT CAST(INTERVAL '1300' MONTH AS INTERVAL YEAR TO MONTH)  
(Title 'Years & Months') ;
```

ERROR

```
SELECT CAST(INTERVAL '1300' MONTH as interval YEAR(3) TO MONTH) ;
```

<u>Years & Month</u>
108-04

The top query failed because the INTERVAL result defaults to 2-digits and we have a 3-digit answer for the year portion (108). The bottom query fixes that specifying 3-digits. The biggest advantage in using the INTERVAL processing is that SQL written on another system is now compatible with Teradata.

The OVERLAPS Command

Compatibility: Teradata Extension

The syntax of the OVERLAPS is:

SELECT <literal>

 WHERE (<start-date-time>, <end-date-time>) OVERLAPS
(<start-date-time>, <end-date-time>) ;

```
SELECT 'The Dates Overlap' (TITLE '')  
WHERE (DATE '2001-01-01', DATE '2001-11-30') OVERLAPS  
      (DATE '2001-10-15', DATE '2001-12-31');
```

Answer



The Dates Overlap

When working with dates and times, sometimes it is necessary to determine whether two different ranges have common points in time. Teradata provides a Boolean function to make this test for you. It is called OVERLAPS; it evaluates true if multiple points are in common, otherwise it returns a false. The literal is returned because both date ranges have from October 15 through November 30 in common.

An OVERLAPS Example that Returns No Rows

```
SELECT 'The dates overlap' (TITLE '')
WHERE (DATE '2001-01-01', DATE '2001-11-30') OVERLAPS
      (DATE '2001-11-30', DATE '2001-12-31') ;
```

Answer
 No rows found

The above SELECT example tests two literal dates and uses the OVERLAPS to determine whether or not to display the character literal.

The literal was not selected because the ranges do not overlap. So, the common single date of November 30 does not constitute an overlap. When dates are used, 2 days must be involved and when time is used, 2 seconds must be contained in both ranges.

The OVERLAPS Command using TIME

```
SELECT 'The Times Overlap' (TITLE '')
WHERE (TIME '08:00:00', TIME '02:00:00') OVERLAPS
      (TIME '02:01:00', TIME '04:15:00') ;
```

Answer → The Times Overlap

The above SELECT example tests two literal times and uses the OVERLAPS to determine whether or not to display the character literal.

This is a tricky example and it is shown to prove a point. At first glance, it appears as if this answer is incorrect because 02:01:00 looks like it starts 1 second after the first range ends. However, the system works on a 24-hour clock when a date and time (timestamp) are not used together. Therefore, the system considers the earlier time of 2AM time as the start and the later time of 8 AM as the end of the range. Therefore, not only do they overlap, the second range is entirely contained in the first range.

The OVERLAPS Command using a NULL Value

```
SELECT 'The Times Overlap' (TITLE '')
WHERE (TIME '10:00:00', NULL) OVERLAPS (TIME '01:01:00', TIME '04:15:00')
```

Answer
 No Rows Found

The above SELECT example tests two literal dates and uses the OVERLAPS to determine whether or not to display the character literal:

When using the OVERLAPS function, there are a couple of situations to keep in mind:

1. A single point in time, i.e. the same date, does not constitute an overlap. There must be at least one second of time in common for TIME or one day when using DATE.
2. Using a NULL as one of the parameters, the other DATE or TIME constitutes a single point in time instead of a range.

Chapter 20

Format Functions

“Even I don’t wake up looking like Cindy Crawford.”

- Cindy Crawford

Table of Contents Chapter 20 – Format Functions

- [The FORMAT Command](#)
- [The Basics of the FORMAT Command](#)
- [Quiz – How will the Date Appear after Formatting](#)
- [Answer to Quiz – How will the Date Appear after Formatting](#)
- [Quiz – How will the Date Appear after Formatting](#)
- [Answer to Quiz – How will the Date Appear after Formatting](#)
- [Formatting with MMM for the Abbreviated Month](#)
- [Answer to Quiz – How will the Date Appear after Formatting](#)
- [Formatting with MMMM for the Full Month Name](#)
- [Formatting with MMMM for the Full Month](#)
- [Formatting with DDD for the Julian Day](#)
- [Formatting with DDD for the Julian Day](#)
- [Formatting with EEE or EEEE for the Day of the Week](#)
- [EEEE for the Abbreviated or Full Day of the Week](#)
- [Placing Spaces inside your Formatting Commands with a B](#)
- [Formatting Spaces with B or b](#)
- [Formatting with 9](#)
- [Formatting with 9 Results](#)
- [Troubleshooting when Formatted Data Overflows](#)
- [Troubleshooting when Formatted Data Overflows](#)
- [Formatting with X or x](#)
- [Formatting with X or x Results](#)
- [Formatting with Z](#)
- [Formatting with Z Visual](#)
- [Formatting with 9](#)

Continued on next page

Table of Contents Chapter 20 – Format Functions Continued

- [Formatting with 9 Visual](#)
- [Formatting with \\$](#)
- [Formatting with \\$ Visual](#)
- [Formatting with \\$ and Commas](#)
- [Formatting with \\$ and Commas Visual](#)
- [Formatting with \\$ and Commas and 9](#)
- [Formatting with \\$ and Commas and 9 with Zero Dollars](#)
- [A Great Formatting Example](#)
- [A Great Formatting Example for Day, Month and Year](#)
- [A Trick to get SQL Assistant to Format Data](#)
- [Using the CASESPECIFIC \(CS\) Command in Teradata Mode](#)
- [Using NOT CASESPECIFIC \(CS\) in ANSI Mode](#)
- [Using the LOWER Command](#)
- [Using the UPPER Command](#)

The FORMAT Command

```
SELECT Current_Date (FORMAT 'mm-dd-yy') ;
```

```
SELECT Current_Date (FORMAT 'mm-dd-yy') ;
```

DATE
05-07-12



Today's date has
been formatted
for a 2-digit year!

In this example, we are using it for dates. All dates in Teradata are stored in the systems as an INTERGERDATE. This allows it to be read and formatted easily.

The Basics of the FORMAT Command

```
SELECT Current_Date (FORMAT 'mm-dd-yy') ;
```



When you put 'mm', you will get the month as a two digit number.

```
SELECT Current_Date (FORMAT 'mm-dd-yy') ;
```



When you put 'dd', you are formatting the day as a two digit number.

```
SELECT Current_Date (FORMAT 'mm-dd-yy') ;
```



When you put 'yy', you are formatting the year as a two digit number.

Format the dates for appearance on the report. The actual data doesn't change.

Quiz – How will the Date Appear after Formatting

Current Date = March 20th, 2011

```
SELECT Current_Date (FORMAT 'mm-dd-yy');
```

How will the answer appear?



Answer to Quiz – How will the Date Appear after Formatting

Current Date = March 20th, 2011

```
SELECT Current_Date (FORMAT 'mm-dd-yy') ;
```

How will the answer appear?

ANSWER: 03-20-11

Quiz – How will the Date Appear after Formatting

Current Date = March 20th, 2011

```
SELECT Current_Date (FORMAT 'mm-dd-yyyy') ;
```

How will the answer appear?



Answer to Quiz – How will the Date Appear after Formatting

Current Date = March 20th, 2011

```
SELECT Current_Date (FORMAT 'mm-dd-yyyy') ;
```

How will the answer appear?

ANSWER: 03-20-2011

Formatting with MMM for the Abbreviated Month

Current Date = March 20th, 2011

SELECT Current_Date (FORMAT ‘mmm-dd-yyyy’) ;



How will the answer appear?



Answer to Quiz – How will the Date Appear after Formatting

Current Date = March 20th, 2011

SELECT Current_Date (FORMAT ‘mmm-dd-yyyy’) ;



How will the answer appear?

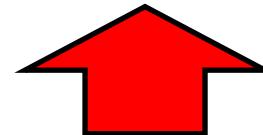
You get the first three letters of the month!

ANSWER: Mar-20-2011

Formatting with MMMM for the Full Month Name

Current Date = March 20th, 2011

```
SELECT Current_Date (FORMAT 'mmmm-dd-yyyy') ;
```



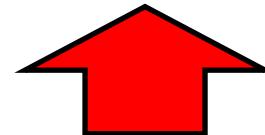
How will the answer appear?



Formatting with MMMM for the Full Month

Current Date = **March 20th, 2011**

SELECT Current_Date (FORMAT ‘mmmm-dd-yyyy’) ;



How will the answer appear?

ANSWER: March-20-2011

Formatting with DDD for the Julian Day

Current Date = March 20th, 2011

SELECT Current_Date (FORMAT ‘mm-ddd-yyyy’);



How will the answer appear?

Formatting with DDD for the Julian Day

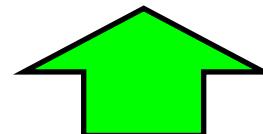
Current Date = March 20th, 2011

SELECT Current_Date (FORMAT ‘mm-ddd-yyyy’);



How will the answer appear?

ANSWER: 03-079-2011



Julian Date

Formatting with EEE or EEEE for the Day of the Week

Current Date = March 20th, 2011

```
SELECT Current_Date (FORMAT 'eee-mm-ddd-yyyy');
```

There are 3 e's in front!

```
SELECT Current_Date (FORMAT 'eeee-mm-ddd-yyyy');
```

There are 4 e's in front!

How will the answers appear?

EEEE for the Abbreviated or Full Day of the Week

Current Date = **March 20th, 2011**

SELECT Current_Date (FORMAT 'eee-mm-ddd-yyyy') ;

There are 3 e's in front!

ANSWER: **Sun-03-20-11**

SELECT Current_Date (FORMAT 'eeee-mm-ddd-yyyy') ;

There are 4 e's in front!

ANSWER: **Sunday-03-20-11**

Placing Spaces inside your Formatting Commands with a B

Current Date = March 20th, 2011

```
SELECT Current_Date (FORMAT 'eeeeBBbbMMMM');
```

How will the answer appear?



Formatting Spaces with B or b

Current Date = March 20th, 2011

```
SELECT Current_Date (FORMAT 'eeeeBBbbMMMM');
```

↑↑↑↑
4 Blank Spaces

How will the answer appear?

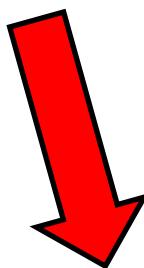


ANSWER: Sunday March

Formatting with 9

You can also format how numbers appear. Such as in the case of a phone number.

```
SELECT 5133000341 (FORMAT '999-999-9999');
```



How will the answer appear?

By putting in 999-999-9999, this is telling the system to put the literal numbers 5133000341 next to the SELECT into the formatting style.

Formatting with 9 Results

You can also format how numbers appear. Such as in the case of a phone number.

```
SELECT '5133000341' (FORMAT '999-999-9999');
```



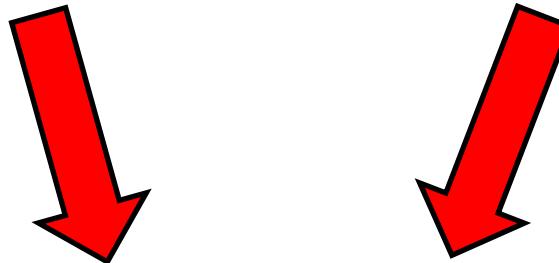
513-300-0341

By putting in 999-999-9999, this is telling the system to put the literal numbers 5133000341 next to the SELECT into the formatting style.

Troubleshooting when Formatted Data Overflows

Notice that we've taken out one of the 9s in the Format Statement

```
SELECT '5133000341' (FORMAT '99-999-9999');
```



How will the answer appear?

The FORMAT OVERFLOW is what happens when your system doesn't have enough spaces in the FORMAT to cover the number you are trying to format.

Troubleshooting when Formatted Data Overflows

Notice that we've taken out one of the 9s in the Format Statement

```
SELECT '5133000341' (FORMAT '99-999-9999');
```



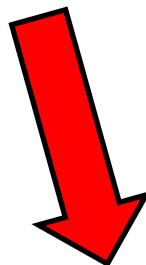
This is not an error, but something is wrong!

The FORMAT OVERFLOW is what happens when your system doesn't have enough spaces in the FORMAT to cover the number you are trying to format.

Formatting with X or x

You can also format letters and words!

SELECT ‘ABCDE’ (FORMAT ‘XxX’);



How will the answer appear?

You can also FORMAT characters. Look at this example. It doesn’t matter if the X’s are capitalized or not.

Formatting with X or x Results

You can also format letters and words!

SELECT ‘ABCDE’ (FORMAT ‘XxX’);

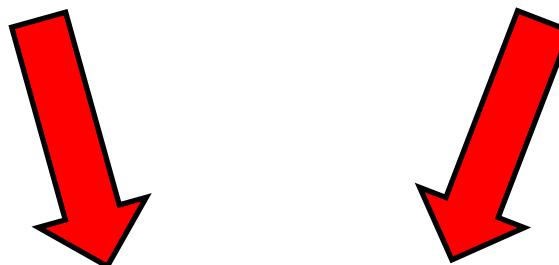


You can also FORMAT characters. Look at this example. It doesn't matter if the X's are capitalized or not.

Formatting with Z

The Z's represent potential data. This tells the system that if there is a number to put in the Z's position, then put it in. If there is not, leave it blank.

SELECT 1021.53 (FORMAT 'ZZZZZZ9.99');



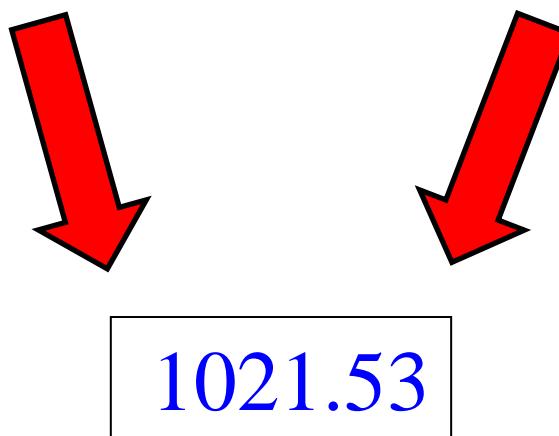
How will the answer appear?

What a '9' represents in a format statement is that if there is a number in the '9' position, then put it in. If there isn't one, then you put a blank.

Formatting with Z Visual

The Z's represent potential data. This tells the system that if there is a number to put in the Z's position, then put it in. If there is not, leave it blank.

```
SELECT 1021.53 (FORMAT 'ZZZZZZ9.99');
```

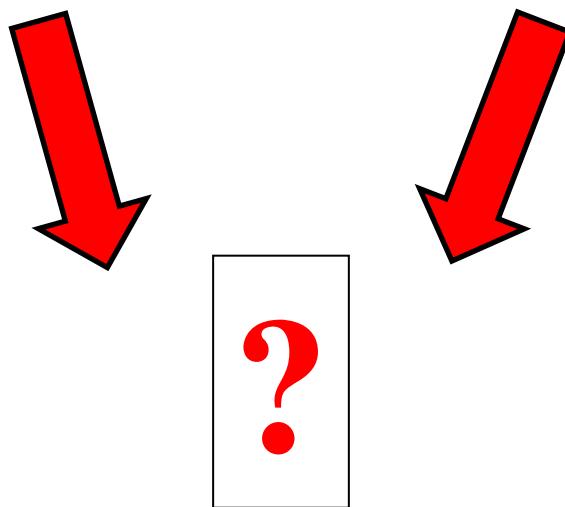


What a '9' represents in a format statement is that if there is a number in the '9' position, then put it in. If there isn't one, then you put a blank.

Formatting with 9

The 9's represent potential data. This tells the system that if there is a number to put in the 9's position, then put it in. If there is not, leave it blank.

```
SELECT 1021.53 (FORMAT '99999999.9999');
```

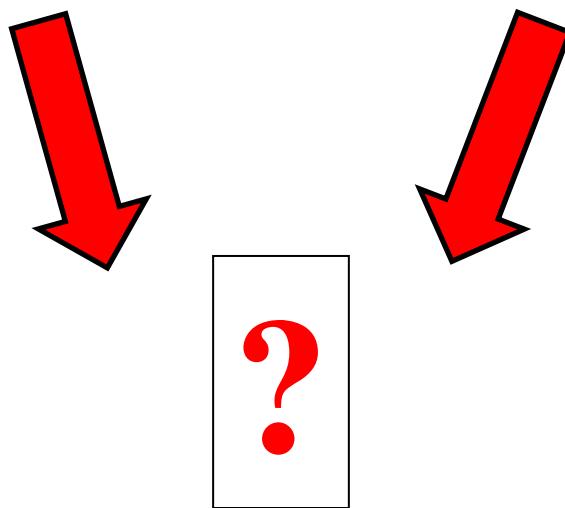


How will the answer appear?

Formatting with 9 Visual

The 9's represent potential data. This tells the system that if there is a number to put in the 9's position, then put it in. If there is not, leave it blank.

```
SELECT 1021.53 (FORMAT '99999999.9999');
```

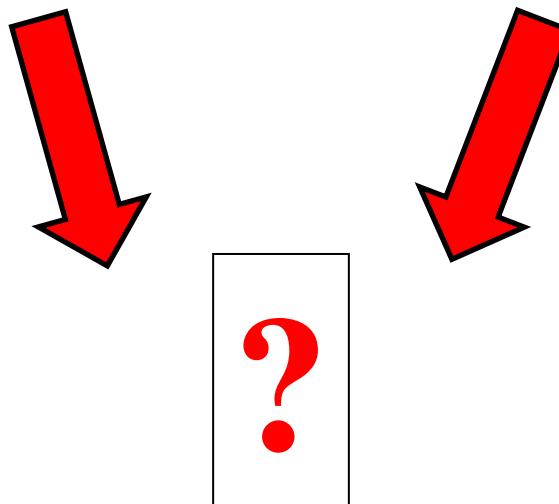


ANSWER: 00001021.5300

Formatting with \$

What the \$ allows you to do is to tell your formatting to place a \$ sign in front of the result set, but only at the beginning.

```
SELECT 1021.53 (FORMAT '$$$$$$9.99');
```

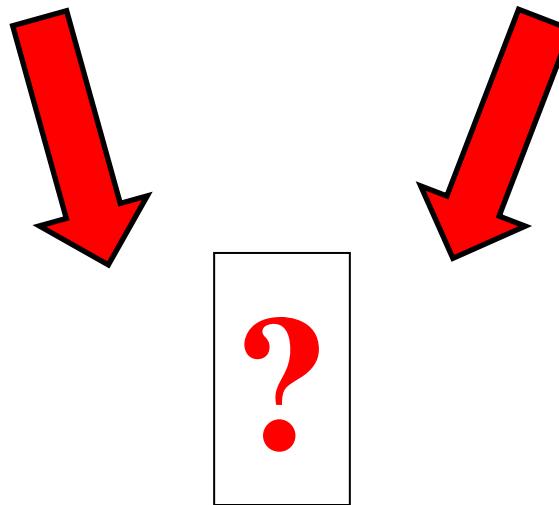


How will the answer appear?

Formatting with \$ Visual

What the \$ allows you to do is to tell your formatting to place a \$ sign in front of the result set, but only at the beginning.

```
SELECT 1021.53 (FORMAT '$$$$$$9.99');
```

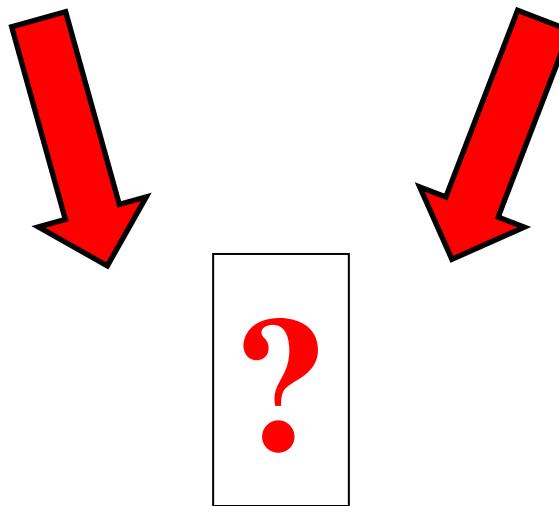


ANSWER: \$1021.53

Formatting with \$ and Commas

You can also use commas in your Formatting statements.

```
SELECT 1021.53 (FORMAT '$, $$,$$9.99');
```

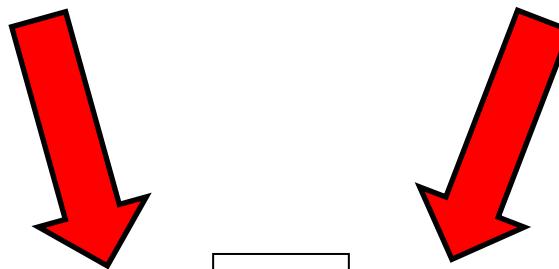


How will the answer appear?

Formatting with \$ and Commas Visual

You can also use commas in your Formatting statements.

```
SELECT 1021.53 (FORMAT '$, $$,$$9.99');
```

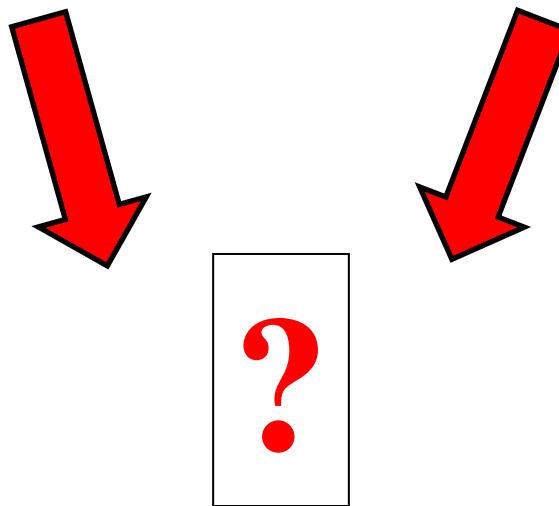


ANSWER: \$1,021.53

Formatting with \$ and Commas and 9

You can also use commas in your Formatting statements.

```
SELECT 0.53 (FORMAT '$, $$,$$9.99');
```

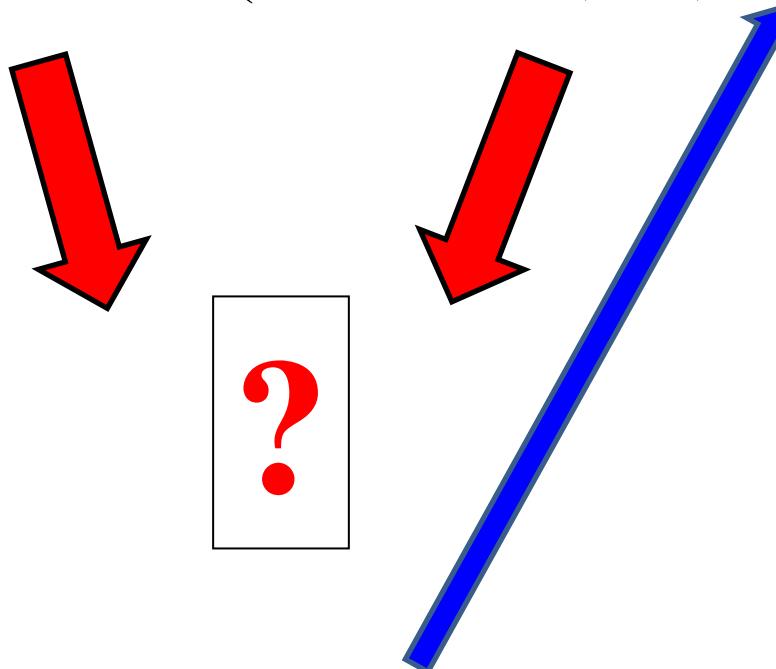


How will the answer appear?

Formatting with \$ and Commas and 9 with Zero Dollars

You can also use commas in your Formatting statements.

```
SELECT 1021.53 (FORMAT '$$$$$$9.99');
```



ANSWER: \$0.53

The '9' sees the 0 and knows to bring it back in the answer set. The floating \$ will only bring back a \$ for the first character.

A Great Formatting Example

```
SELECT 'ABCDE'      (FORMAT 'XxX') AS Fmt_Shorter  
,2014859999 (FORMAT '999-999-9999') AS Fmt_Phone  
,1021.53    (FORMAT 'ZZZZZZ9.9999') AS Z_Press  
,991001(date) (FORMAT 'Yyddd') AS Fmt_Julian  
,991001(date) (FORMAT 'eee') AS Weekday  
,991001    (FORMAT '$$$,$$$.$99') AS Fmt_Pay ;
```

Fmt_Shorter	Fmt_Phone	Z_Press	Fmt_Julian	Weekday	Fmt_Pay
ABC	201-485-9999	1021.53	99274	Fri	\$991,001.00

There are only two things that need to be watched when using the FORMAT function. First, the data type must match the formatting character used or a syntax error is returned. So, if the data is numeric, use a numeric formatting character and the same condition for character data. The other concern is configuring the format mask big enough for the largest data column. If the mask is too short, the SQL command executes, however, the output contains a series of ***** to indicate a format overflow.

A Great Formatting Example for Day, Month and Year

```
SELECT Current_Date (FORMAT 'mm-dd-yy') as Digit2_YR  
,Current_Date (FORMAT 'mm-dd-yyyy') as Digit4_YR  
,Current_Date (FORMAT 'mmm-dd-yyyy')as Mnth_Initials  
,Current_Date (FORMAT 'mmmm-dd-yy')as Mnth_Spelled  
,Current_Date (FORMAT 'mm-ddd-yyyy') as Day_Julian  
,Current_Date (FORMAT 'eeeBmm-dd-yy') as Day_of_Week  
,Current_Date (FORMAT 'eeeeBmm-dd-yy') as Day_Spelled
```

Digit2_YR	Digit4_YR	Mnth_Initials	Mnth_Spelled	Day_Julian	Day_of_Week	Day_Spelled
03-20-11	03-20-2011	Mar-20-2011	March-20-11	03-079-2011	Sun	03-20-11

All of these FORMAT requests work wonderfully if the client software is BTEQ or the Nexus Query Chameleon, but SQL Assistant has problems with formatting. After all, it is a report writer and these are report writer options. The issue is that the ODBC and SQL Assistant look at the data as data, not as a report. Since many of the formatting symbols are “characters” they cannot be numeric. Therefore, the ODBC strips off the symbols and presents the numeric data to the client software for display.

A Trick to get SQL Assistant to Format Data

```
SELECT CAST( (4859999 (FORMAT '999-9999')) AS CHAR(8) ) AS Phone  
,991001(date) (FORMAT 'yyyy.mm.dd') (CHAR(10) ) AS Cast_Date  
,CAST( (991001 (FORMAT '$$$,$$$,.99')) AS CHAR(11) ) AS Pay ;
```

Phone	Cast_Date	Pay
485-9999	1999.10.01	\$991,001.00

If a tool uses the ODBC, the FORMAT in the SELECT is ignored and the data comes back as data, not as a formatted field. This is especially noticeable with numeric data and dates.

To force tools like SQL Assistant to format the data, the software must be tricked into thinking the data is character type, which it leaves alone. This can be done using the CAST function. The Nexus Query Chameleon does not have a problem formatting.

Using the CASESPECIFIC (CS) Command in Teradata Mode

```
SELECT 'They match'      as "Do They"  
WHERE 'A'(CASESPECIFIC) = 'a'(CS) ;
```

Do They

They did NOT match using the
CASESPECIFIC Command

```
SELECT 'They match' as "Do They"  
WHERE 'A' = 'a' ;
```

Do They

They Match

They match because Teradata Mode
does not differentiate about case.

In Teradata Mode a capital 'A' is seen as the same in comparison as a little 'a'. The CASESPECIFIC command, which is abbreviated as CS will make sure the case of a letter is examined.

Using NOT CASESPECIFIC (CS) in ANSI Mode

```
SELECT 'They match' as "Do They"  
WHERE 'A'(NOT CASESPECIFIC) = 'a'(NOT CS) ;
```

Do They
They Match

They match when using the
NOT CASESPECIFIC Command

```
SELECT 'They match' as "Do They"  
WHERE 'A' = 'a' ;
```

Do They

They won't match in ANSI Mode
because ANSI distinguishes
between upper and lower case.

In ANSI Mode a capital 'A' is seen as different in comparison as a little 'a'. The NOT CASESPECIFIC command, which is abbreviated as NOT CS will make sure the case of a letter is examined and returned whether or not they have the same case.

Using the LOWER Command

```
SELECT LOWER('AbCdE') as Result1;
```

Result1

abcde

All Capitol letters are
now in LOWER Case

```
SELECT 'They match' as "Do They"  
WHERE LOWER('ABCDE') = 'abcde' ;
```

Do They

They Match

They match because we used
the LOWER command so now
they match perfectly in ANSI Mode

In ANSI Mode a capital 'A' is seen as different in comparison as a little 'a'. The LOWER command can be used to make sure the case of a column is lowered.

Using the UPPER Command

```
SELECT UPPER('AbCdE') as Result1;
```

Result1

ABCDE

All letters are now
in UPPER Case

```
SELECT 'They match' as "Do They"  
WHERE 'ABCDE' = UPPER('abcde') ;
```

Do They

They Match

They match because we used
the UPPER command so now
they match perfectly in ANSI Mode

In ANSI Mode a capital 'A' is seen as different in comparison as a little 'a'. The UPPER command can be used to make sure the case of a column is UPPERED.

Chapter 21

OLAP Functions

“The best we can do is size up the chances, calculate the risks involved, estimate our ability to deal with them, and then make our plans with confidence.”

- Henry Ford

Table of Contents Chapter 21- OLAP Functions

- [On-Line Analytical Processing \(OLAP\) or Ordered Analytics](#)
- [Cumulative Sum \(CSUM\) Command and how OLAP Works](#)
- [OLAP Commands always Sort \(ORDER BY\) in the Command](#)
- [Calculate the Cumulative Sum \(CSUM\) after Sorting the Data](#)
- [The OLAP Major Sort Key](#)
- [The OLAP Major Sort Key and the Minor Sort Key\(s\)](#)
- [Troubleshooting OLAP – My Data isn't coming back Correct](#)
- [GROUP BY in Teradata OLAP Syntax Resets on the Group](#)
- [CSUM the Number 1 to get a Sequential Number](#)
- [A Single GROUP BY Resets each OLAP with Teradata Syntax](#)
- [A Better Choice – The ANSI Version of CSUM](#)
- [The ANSI Version of CSUM – The Sort Explained](#)
- [The ANSI CSUM – Rows Unbounded Preceding Explained](#)
- [The ANSI CSUM – Making Sense of the Data](#)
- [The ANSI CSUM – Making Even More Sense of the Data](#)
- [The ANSI CSUM – The Major and Minor Sort Key\(s\)](#)
- [The ANSI CSUM – Getting a Sequential Number](#)
- [Troubleshooting The ANSI OLAP on a GROUP BY](#)
- [The ANSI OLAP – Reset with a PARTITION BY Statement](#)
- [PARTITION BY only Resets a Single OLAP not ALL of them](#)
- [The Moving Average \(MAVG\) and Moving Window](#)
- [How the Moving Average is Calculated](#)
- [How the Sort works for Moving Average \(MAVG\)](#)
- [GROUP BY in the Moving Average does a Reset](#)
- [Quiz – Can you make the Advanced Calculation in your mind?](#)
- [Answer to Quiz for the Advanced Calculation in your mind?](#)
- [Quiz – Write that Teradata Moving Average in ANSI Syntax](#)

Continued on next page

Table of Contents Chapter 21- OLAP Functions continued

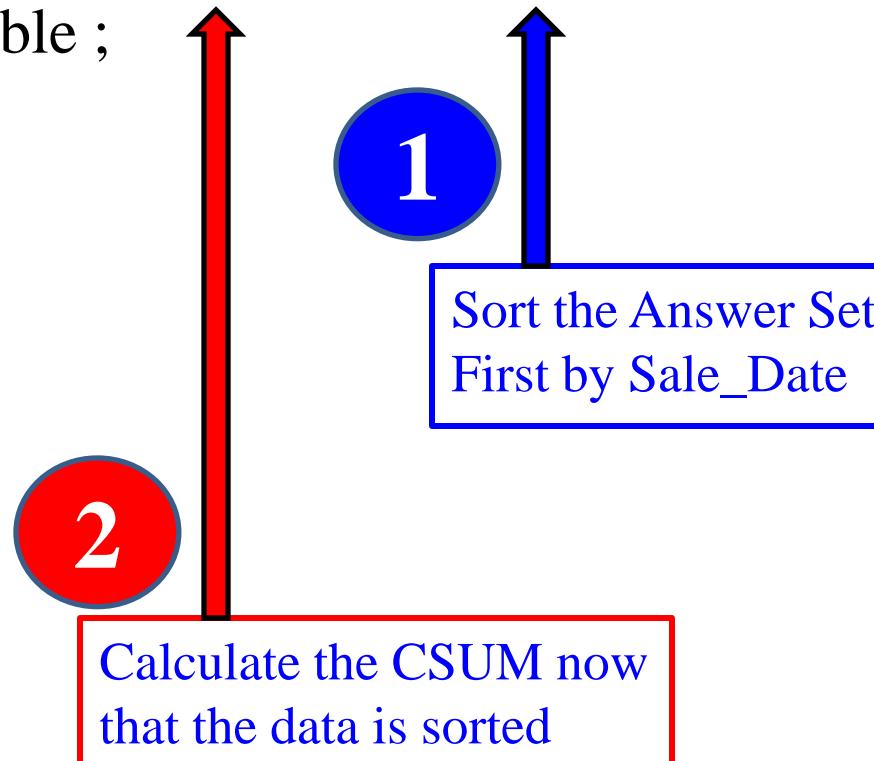
- [Both the Teradata Moving Average and ANSI Version](#)
- [The ANSI Moving Window is Current Row and Preceding](#)
- [How ANSI Moving Average Handles the Sort](#)
- [Quiz – How is that Total Calculated](#)
- [Answer to Quiz – How is that Total Calculated?](#)
- [Quiz – How is that 4th Row Calculated?](#)
- [Answer to Quiz – How is that 4th Row Calculated?](#)
- [Moving Average every 3-rows Vs a Continuous Average](#)
- [Partition By Resets an ANSI OLAP](#)
- [The Moving Difference \(MDIFF\)](#)
- [Moving Difference \(MDIFF\) Visual](#)
- [Moving Difference using ANSI Syntax](#)
- [Moving Difference using ANSI Syntax with Partition By](#)
- [Trouble Shooting the Moving Difference \(MDIFF\)](#)
- [The RANK Command](#)
- [How to get Rank to Sort in Ascending Order](#)
- [Two ways to get Rank to Sort in Ascending Order](#)
- [RANK using ANSI Syntax Defaults to Ascending Order](#)
- [Getting RANK using ANSI Syntax to Sort in DESC Order](#)
- [RANK\(\) OVER and PARTITION BY with a QUALIFY](#)
- [QUALIFY and WHERE](#)
- [Quiz – How can you simplify the QUALIFY Statement](#)
- [Answer to Quiz – Can you simplify the QUALIFY Statement](#)
- [The QUALIFY Statement without Ties](#)
- [The QUALIFY Statement with Ties](#)
- [The QUALIFY Statement with Ties Brings back Extra Rows](#)
- [Mixing Sort Order for QUALIFY Statement](#)
- [Quiz – What Caused the RANK to Reset?](#)
- [Answer to Quiz – What Caused the RANK to Reset?](#)

Continued on next page

Table of Contents Chapter 21- OLAP Functions Continued

On-Line Analytical Processing (OLAP) or Ordered Analytics

```
SELECT Product_ID  
      ,Sale_Date  
      ,Daily_Sales  
      ,CSUM(Daily_Sales, Sale_Date) AS "CSum"  
FROM Sales_Table ;
```



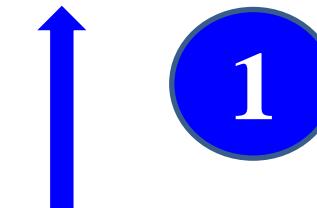
OLAP is often called Ordered Analytics because the first thing every OLAP does before any calculating is SORT all the rows. The query above sorts by Sale_Date!

Cumulative Sum (CSUM) Command and how OLAP Works

```
SELECT Product_ID , Sale_Date, Daily_Sales  
      ,CSUM(Daily_Sales, Sale_Date) AS "CSum"  
FROM Sales_Table ;
```

<u>Product ID</u>	<u>Sale Date</u>	<u>Daily Sales</u>	<u>CSUM</u>
1000	2000-09-28	48850.40	
2000	2000-09-28	41888.88	
3000	2000-09-28	61301.77	
1000	2000-09-29	54500.22	
2000	2000-09-29	48000.00	
3000	2000-09-29	34509.13	
1000	2000-09-30	36000.07	
2000	2000-09-30	49850.03	
3000	2000-09-30	43868.86	
1000	2000-10-01	40200.43	
2000	2000-10-01	54850.29	
3000	2000-10-01	28000.00	
1000	2000-10-02	32800.50	
2000	2000-10-02	36021.93	
3000	2000-10-02	19678.94	

Not all rows
are displayed in
this answer set



Sort the Answer Set
first by Sale_Date, but
**Don't do any CSUM
Calculations yet!**

OLAP always sorts first and then is in a position to calculate starting with the first sorted row and continuing to the last sorted row, thus calculating all Daily_Sales.

OLAP Commands always Sort (ORDER BY) in the Command

```
SELECT Product_ID , Sale_Date, Daily_Sales  
      ,CSUM(Daily_Sales, Sale_Date) AS "CSum"  
FROM Sales_Table ;
```

Product_ID	Sale Date	Daily_Sales	CSUM
1000	2000-09-28	48850.40	48850.40
2000	2000-09-28	41888.88	90739.28
3000	2000-09-28	61301.77	152041.05
1000	2000-09-29	54500.22	206541.27
2000	2000-09-29	48000.00	254541.27
3000	2000-09-29	34509.13	289050.40
1000	2000-09-30	36000.07	325050.47
2000	2000-09-30	49850.03	374900.50
3000	2000-09-30	43868.86	418769.36
1000	2000-10-01	40200.43	458969.79
2000	2000-10-01	54850.29	513820.08
3000	2000-10-01	28000.00	541820.08
1000	2000-10-02	32800.50	574620.58

Not all rows are displayed in this answer set

2

Calculate the CSUM starting with the first sorted row and go to the last.

Once the data is first sorted by Sale_Date then phase 2 is ready and the OLAP calculation can be performed on the sorted data. Day 1 we made 48850.40! Add the next row's Daily_Sales to get a Cumulative Sum (CSUM) to get **90739.28!**

Calculate the Cumulative Sum (CSUM) after Sorting the Data

```
SELECT Product_ID , Sale_Date, Daily_Sales,  
       CSUM(Daily_Sales, Sale_Date) AS "CSUM"  
FROM Sales_Table WHERE Product_ID BETWEEN 1000 and 2000
```

Not all rows
are displayed in
this answer set

<u>Product ID</u>	<u>Sale Date</u>	<u>Daily Sales</u>	<u>CSUM</u>
2000	2000-09-28	41888.88	41888.88
1000	2000-09-28	48850.40	90739.28
2000	2000-09-29	48000.00	138739.28
1000	2000-09-29	54500.22	193239.50
1000	2000-09-30	36000.07	229239.57
2000	2000-09-30	49850.03	279089.60
1000	2000-10-01	40200.43	319290.03
2000	2000-10-01	54850.29	374140.32
1000	2000-10-02	32800.50	406940.82
2000	2000-10-02	36021.93	442962.75
1000	2000-10-03	64300.00	507262.75

This is our first OLAP known as a CSUM. Right now, the syntax wants to see the cumulative sum of the Daily_Sales sorted by Sale_Date. The first thing the above query does before calculating is SORT all the rows on Sale_Date.

The OLAP Major Sort Key

```
SELECT Product_ID , Sale_Date, Daily_Sales,  
       CSUM(Daily_Sales, Sale_Date) AS "CSum"  
FROM Sales_Table WHERE Product_ID BETWEEN 1000 and 2000
```

<u>Product ID</u>	<u>Sale Date</u>	<u>Daily Sales</u>	<u>CSum</u>
2000	2000-09-28	41888.88	41888.88
1000	2000-09-28	48850.40	90739.28
2000	2000-09-29	48000.00	138739.28
1000	2000-09-29	54500.22	193239.50
1000	2000-09-30	36000.07	229239.57
2000	2000-09-30	49850.03	279089.60
1000	2000-10-01	40200.43	319290.03
2000	2000-10-01	54850.29	374140.32
1000	2000-10-02	32800.50	406940.82
2000	2000-10-02	36021.93	442962.75
1000	2000-10-03	64300.00	507262.75

Not all rows
are displayed in
this answer set

In a CSUM, the **second column** listed is always the **major SORT KEY**. The SORT KEY in the above query is Sale_Date. Notice again the answer set is sorted by this. After the sort has finished the CSUM is calculated starting with the first sorted row till the end.

The OLAP Major Sort Key and the Minor Sort Key(s)

```
SELECT      Product_ID , Sale_Date, Daily_Sales,  
          CSUM(Daily_Sales, Product_ID, Sale_Date) AS "CSum"  
FROM Sales_Table;
```

Major Sort Key

Minor Sort Key

Product ID	Sale Date	Daily Sales	CSUM
1000	2000-09-28	48850.40	48850.40
1000	2000-09-29	54500.22	103350.62
1000	2000-09-30	36000.07	139350.69
1000	2000-10-01	40200.43	179551.12
1000	2000-10-02	32800.50	212351.62
1000	2000-10-03	64300.00	276651.62
1000	2000-10-04	54553.10	331204.72
2000	2000-09-28	41888.88	373093.60
2000	2000-09-29	48000.00	421093.60
2000	2000-09-30	49850.03	470943.63
2000	2000-10-01	54850.29	525793.92

Not all rows
are displayed in
this answer set

Product_ID is the MAJOR sort key and Sale_Date is the MINOR Sort key above.

Troubleshooting OLAP – My Data isn't coming back Correct

```
SELECT      Product_ID , Sale_Date, Daily_Sales,  
          CSUM(Daily_Sales, Product_ID, Sale_Date)  AS "CSum"  
FROM Sales_Table WHERE Product_ID BETWEEN 1000 and 2000  
ORDER BY Daily_Sales;
```



Don't place an
ORDER BY
at the end!

Product ID	Sale Date	Daily Sales	CSUM
1000	2000-10-02	32800.50	212351.62
2000	2000-10-04	32800.50	637816.53
1000	2000-09-30	36000.07	139350.69
2000	2000-10-02	36021.93	561815.85
1000	2000-10-01	40200.43	179551.12
2000	2000-09-28	41888.88	373093.60
2000	2000-10-03	43200.18	605016.03
2000	2000-09-29	48000.00	421093.60
1000	2000-09-28	48850.40	48850.40
2000	2000-09-30	49850.03	470943.63
1000	2000-09-29	54500.22	103350.62
1000	2000-10-04	54553.10	331204.72
2000	2000-10-01	54850.29	525793.92
1000	2000-10-03	64300.00	276651.62

The first thing every OLAP does is SORT. That means you should NEVER put an ORDER BY at the end. It will mess up the ENTIRE result set.

GROUP BY in Teradata OLAP Syntax Resets on the Group

```
SELECT      Product_ID , Sale_Date, Daily_Sales,  
          CSUM(Daily_Sales, Product_ID, Sale_Date)  AS "CSum"  
FROM Sales_Table  
GROUP BY Product_ID ;
```



<u>Product_ID</u>	<u>Sale_Date</u>	<u>Daily_Sales</u>	<u>CSUM</u>
1000	2000-09-28	48850.40	48850.40
1000	2000-09-29	54500.22	103350.62
1000	2000-09-30	36000.07	139350.69
1000	2000-10-01	40200.43	179551.12
1000	2000-10-02	32800.50	212351.62
1000	2000-10-03	64300.00	276651.62
1000	2000-10-04	54553.10	331204.72
2000	2000-09-28	41888.88	41888.88
2000	2000-09-29	48000.00	89888.88
2000	2000-09-30	49850.03	139738.91
2000	2000-10-01	54850.29	194589.20

Reset now!

Not all rows displayed In answer set

The GROUP BY Statement cause the CSUM to start over (reset) on its calculating the cumulative sum of the Daily_Sales each time it runs into a NEW Product_ID.

CSUM the Number 1 to get a Sequential Number

```
SELECT Product_ID , Sale_Date, Daily_Sales,  
       CSUM(Daily_Sales, Product_ID, Sale_Date) AS "CSum",  
       CSUM(1, Product_ID, Sale_Date) as "Seq_Number"  
           ↑          FROM Sales_Table ;
```

Product_ID	Sale_Date	Daily_Sales	CSUM	Seq_Number
1000	2000-09-28	48850.40	48850.40	→ 1
1000	2000-09-29	54500.22	103350.62	→ 2
1000	2000-09-30	36000.07	139350.69	3
1000	2000-10-01	40200.43	179551.12	4
1000	2000-10-02	32800.50	212351.62	5
1000	2000-10-03	64300.00	276651.62	6
1000	2000-10-04	54553.10	331204.72	7
2000	2000-09-28	41888.88	373093.60	8
2000	2000-09-29	48000.00	421093.60	9
2000	2000-09-30	49850.03	470943.63	10

Not all rows
are displayed in
this answer set

With "Seq_Number", because you placed the number 1 in the area where it calculates, it will continuously add 1 to the answer for each row.

A Single GROUP BY Resets each OLAP with Teradata Syntax

```
SELECT Product_ID , Sale_Date, Daily_Sales,  
       CSUM(Daily_Sales, Product_ID, Sale_Date) AS "CSum",  
       CSUM(1, Product_ID, Sale_Date) as "Seq_Number"  
FROM Sales_Table GROUP BY Product_ID ;
```



Not all rows
are displayed in
this answer set

Product ID	Sale Date	Daily Sales	CSUM	Seq Number
1000	2000-09-28	48850.40	48850.40	→ 1
1000	2000-09-29	54500.22	103350.62	→ 2
1000	2000-09-30	36000.07	139350.69	3
1000	2000-10-01	40200.43	179551.12	4
1000	2000-10-02	32800.50	212351.62	5
1000	2000-10-03	64300.00	276651.62	6
1000	2000-10-04	54553.10	331204.72	7
2000	2000-09-28	41888.88	41888.88	→ 1
2000	2000-09-29	48000.00	89888.88	2
2000	2000-09-30	49850.03	139738.91	3
2000	2000-10-01	54850.29	194589.20	4

What does the GROUP BY Statement cause? Both OLAP Commands to reset!

A Better Choice – The ANSI Version of CSUM

Start on 1st row
and continue
till the end

→

```
SELECT Product_ID , Sale_Date, Daily_Sales,  
SUM(Daily_Sales) OVER (ORDER BY Sale_Date  
ROWS UNBOUNDED PRECEDING) AS SUMOVER  
FROM Sales_Table  
WHERE Product_ID BETWEEN 1000 and 2000 ;
```

Not all rows
are displayed in
this answer set

<u>Product ID</u>	<u>Sale Date</u>	<u>Daily Sales</u>	<u>SUMOVER</u>
2000	2000-09-28	41888.88	41888.88
1000	2000-09-28	48850.40	90739.28
2000	2000-09-29	48000.00	138739.28
1000	2000-09-29	54500.22	193239.50
1000	2000-09-30	36000.07	229239.57
2000	2000-09-30	49850.03	279089.60
1000	2000-10-01	40200.43	319290.03
2000	2000-10-01	54850.29	374140.32
1000	2000-10-02	32800.50	406940.82
2000	2000-10-02	36021.93	442962.75

This ANSI version of CSUM is **SUM() Over**. Right now, the syntax wants to see the sum of the **Daily_Sales** after it is first sorted by **Sale_Date**. Rows Unbounded Preceding makes this a CSUM. The ANSI Syntax seems difficult, but only at first.

The ANSI Version of CSUM – The Sort Explained

```
SELECT Product_ID , Sale_Date, Daily_Sales,  
       SUM(Daily_Sales) OVER (ORDER BY Sale_Date  
                                ROWS UNBOUNDED PRECEDING) AS SUMOVER  
FROM Sales_Table WHERE Product_ID BETWEEN 1000 and 2000 ;
```

<u>Product ID</u>	<u>Sale Date</u>	<u>Daily Sales</u>	<u>SUMOVER</u>
2000	2000-09-28	41888.88	41888.88
1000	2000-09-28	48850.40	90739.28
2000	2000-09-29	48000.00	138739.28
1000	2000-09-29	54500.22	193239.50
1000	2000-09-30	36000.07	229239.57
2000	2000-09-30	49850.03	279089.60
1000	2000-10-01	40200.43	319290.03
2000	2000-10-01	54850.29	374140.32
1000	2000-10-02	32800.50	406940.82
2000	2000-10-02	36021.93	442962.75
1000	2000-10-03	64300.00	507262.75
2000	2000-10-03	43200.18	550462.93

Not all rows
are displayed in
this answer set

The first thing the above query does before calculating is **SORT** all the rows by Sale_Date. The Sort is located right after the ORDER BY.

The ANSI CSUM – Rows Unbounded Preceding Explained

```
SELECT Product_ID , Sale_Date, Daily_Sales,  
       SUM(Daily_Sales) OVER (ORDER BY Sale_Date  
      ROWS UNBOUNDED PRECEDING) AS SUMOVER  
FROM Sales_Table WHERE Product_ID BETWEEN 1000 and 2000 ;
```

<u>Product_ID</u>	<u>Sale_Date</u>	<u>Daily_Sales</u>	<u>SUMOVER</u>
2000	2000-09-28	41888.88	41888.88
1000	2000-09-28	48850.40	90739.28
2000	2000-09-29	48000.00	138739.28
1000	2000-09-29	54500.22	193239.50
1000	2000-09-30	36000.07	229239.57
2000	2000-09-30	49850.03	279089.60
1000	2000-10-01	40200.43	319290.03
2000	2000-10-01	54850.29	374140.32
1000	2000-10-02	32800.50	406940.82
2000	2000-10-02	36021.93	442962.75
1000	2000-10-03	64300.00	507262.75

Not all rows
are displayed in
this answer set

The keywords **Rows Unbounded Preceding** determines that this is a **CSUM**. There are only a few different statements and **Rows Unbounded Preceding** is the main one. It means start calculating at the beginning row and continue calculating until the last row..

The ANSI CSUM – Making Sense of the Data

```
SELECT Product_ID , Sale_Date, Daily_Sales,  
       SUM(Daily_Sales) OVER (ORDER BY Sale_Date  
                                ROWS UNBOUNDED PRECEDING) AS SUMOVER  
FROM Sales_Table WHERE Product_ID BETWEEN 1000 and 2000 ;
```

Product_ID	Sale Date	Daily Sales	SUMOVER
2000	2000-09-28	41888.88	41888.88
1000	2000-09-28	48850.40	90739.28
2000	2000-09-29	48000.00	138739.28
1000	2000-09-29	54500.22	193239.50
1000	2000-09-30	36000.07	229239.57
2000	2000-09-30	49850.03	279089.60
1000	2000-10-01	40200.43	319290.03
2000	2000-10-01	54850.29	374140.32
1000	2000-10-02	32800.50	406940.82
2000	2000-10-02	36021.93	442962.75
1000	2000-10-03	64300.00	507262.75

Not all rows
are displayed in
this answer set

The second “SUMOVER” row is 90739.28. That is derived by the first row’s Daily_Sales (41888.88) added to the SECOND row’s Daily_Sales (48850.40).

The ANSI CSUM – Making Even More Sense of the Data

```
SELECT Product_ID , Sale_Date, Daily_Sales,  
       SUM(Daily_Sales) OVER (ORDER BY Sale_Date  
                               ROWS UNBOUNDED PRECEDING) AS SUMOVER  
FROM Sales_Table WHERE Product_ID BETWEEN 1000 and 2000 ;
```

<u>Product_ID</u>	<u>Sale Date</u>	<u>Daily_Sales</u>	<u>SUMOVER</u>
2000	2000-09-28	41888.88	41888.88
1000	2000-09-28	48850.40	90739.28
2000	2000-09-29	48000.00	138739.28
1000	2000-09-29	54500.22	193239.50
1000	2000-09-30	36000.07	229239.57
2000	2000-09-30	49850.03	279089.60
1000	2000-10-01	40200.43	319290.03
2000	2000-10-01	54850.29	374140.32
1000	2000-10-02	32800.50	406940.82
2000	2000-10-02	36021.93	442962.75
1000	2000-10-03	64300.00	507262.75

Not all rows
are displayed in
this answer set

The third “SUMOVER” row is **138739.28**. That is derived by taking the first row’s Daily_Sales (**41888.88**) and adding it to the SECOND row’s Daily_Sales (**48850.40**). Then you add that total to the THIRD row’s Daily_Sales (**48000.00**).

The ANSI CSUM – The Major and Minor Sort Key(s)

```
SELECT Product_ID , Sale_Date, Daily_Sales,  
      SUM(Daily_Sales) OVER (ORDER BY Product_ID, Sale_Date  
      ROWS UNBOUNDED PRECEDING) AS SumOVER  
FROM Sales_Table ;
```

<u>Product_ID</u>	<u>Sale_Date</u>	<u>Daily_Sales</u>	<u>SUMOVER</u>
1000	2000-09-28	48850.40	48850.40
1000	2000-09-29	54500.22	103350.62
1000	2000-09-30	36000.07	139350.69
1000	2000-10-01	40200.43	179551.12
1000	2000-10-02	32800.50	212351.62
1000	2000-10-03	64300.00	276651.62
1000	2000-10-04	54553.10	331204.72
2000	2000-09-28	41888.88	373093.60
2000	2000-09-29	48000.00	421093.60
2000	2000-09-30	49850.03	470943.63
2000	2000-10-01	54850.29	525793.92

Not all rows
are displayed in
this answer set

You can have more than one SORT KEY. In the top query, Product_ID is the MAJOR Sort and Sale_Date is the MINOR Sort.

The ANSI CSUM – Getting a Sequential Number

```
SELECT Product_ID , Sale_Date, Daily_Sales,  
       SUM(Daily_Sales) OVER (ORDER BY Product_ID, Sale_Date  
                               ROWS UNBOUNDED PRECEDING) as SUMOVER,  
       SUM(1) OVER (ORDER BY Product_ID, Sale_Date  
                    ↑          ROWS UNBOUNDED PRECEDING) AS Seq_Number  
FROM Sales_Table ;
```

Product_ID	Sale Date	Daily_Sales	SUM OVER	Seq Number
1000	2000-09-28	48850.40	48850.40	→ 1
1000	2000-09-29	54500.22	103350.62	→ 2
1000	2000-09-30	36000.07	139350.69	3
1000	2000-10-01	40200.43	179551.12	4
1000	2000-10-02	32800.50	212351.62	5
1000	2000-10-03	64300.00	276651.62	6
1000	2000-10-04	54553.10	331204.72	7
2000	2000-09-28	41888.88	373093.60	8
2000	2000-09-29	48000.00	421093.60	9
2000	2000-09-30	49850.03	470943.63	10

Not all rows
are displayed in
this answer set

With “Seq_Number”, because you placed the number 1 in the area which calculates the cumulative sum, it’ll continuously add 1 to the answer for each row.

Troubleshooting The ANSI OLAP on a GROUP BY

```
SELECT Product_ID , Sale_Date, Daily_Sales,  
      SUM(Daily_Sales) OVER (ORDER BY Sale_Date  
                             ROWS UNBOUNDED PRECEDING) AS SUMOVER  
FROM Sales_Table  
GROUP BY Product_ID ;
```

Error! Why?

Never GROUP BY in a SUM()Over or with any ANSI Syntax OLAP command. If you want to reset you use a PARTITION BY Statement, but never a GROUP BY.

The ANSI OLAP – Reset with a PARTITION BY Statement

```
SELECT Product_ID , Sale_Date, Daily_Sales,  
      SUM(Daily_Sales) OVER (PARTITION BY Product_ID  
                            ORDER BY Product_ID, Sale_Date  
                            ROWS UNBOUNDED PRECEDING) AS SumANSI  
FROM Sales_Table ;
```

Product ID	Sale Date	Daily Sales	SumANSI
1000	2000-09-28	48850.40	48850.40
1000	2000-09-29	54500.22	103350.62
1000	2000-09-30	36000.07	139350.69
1000	2000-10-01	40200.43	179551.12
1000	2000-10-02	32800.50	212351.62
1000	2000-10-03	64300.00	276651.62
1000	2000-10-04	54553.10	331204.72
2000	2000-09-28	41888.88	41888.88
2000	2000-09-29	48000.00	89888.88
2000	2000-09-30	49850.03	139738.91

Not all rows
are displayed in
this answer set

The PARTITION Statement is how you reset in ANSI. This will cause the SUMANSI to start over (reset) on its calculating for each NEW Product_ID.

PARTITION BY only Resets a Single OLAP not ALL of them

```
SELECT Product_ID , Sale_Date, Daily_Sales,  
       SUM(Daily_Sales) OVER (PARTITION BY Product_ID  
      ORDER BY Product_ID, Sale_Date  
      ROWS UNBOUNDED PRECEDING) AS Subtotal,  
       SUM(Daily_Sales) OVER (ORDER BY Product_ID, Sale_Date  
      ROWS UNBOUNDED PRECEDING) AS GRANDTotal  
FROM Sales_Table ;
```

Product_ID	Sale_Date	Daily_Sales	SubTotal	GRANDTotal
1000	2000-09-28	48850.40	48850.40	48850.40
1000	2000-09-29	54500.22	103350.62	103350.62
1000	2000-09-30	36000.07	139350.69	139350.69
1000	2000-10-01	40200.43	179551.12	179551.12
1000	2000-10-02	32800.50	212351.62	212351.62
1000	2000-10-03	64300.00	276651.62	276651.62
1000	2000-10-04	54553.10	331204.72	331204.72
2000	2000-09-28	41888.88	41888.88	373093.60
2000	2000-09-29	48000.00	89888.88	421093.60
2000	2000-09-30	49850.03	139738.91	470943.63

Above are two OLAP statements. Only one has PARTITION BY so only it resets.

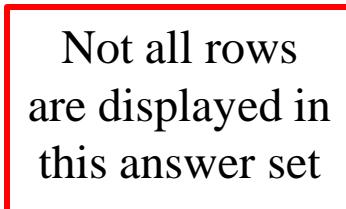
Not all rows
are displayed in
this answer set



The Moving Average (MAVG) and Moving Window

```
SELECT Product_ID , Sale_Date, Daily_Sales,  
MAVG( Daily_Sales, 3, Product_ID, Sale_Date) AS AVG3_Rows  
FROM Sales_Table
```





Product_ID	Sale_Date	Daily_Sales	AVG3_Rows
1000	2000-09-28	48850.40	48850.40
1000	2000-09-29	54500.22	51675.31
1000	2000-09-30	36000.07	46450.23
1000	2000-10-01	40200.43	43566.91
1000	2000-10-02	32800.50	36333.67
1000	2000-10-03	64300.00	45788.98
1000	2000-10-04	54553.10	50551.20
2000	2000-09-28	41888.88	53580.66
2000	2000-09-29	48000.00	48147.33
2000	2000-09-30	49850.03	46579.11

This is the **Moving Average (MAVG)**. It will calculate the average of 3 rows because that is the Moving Window. It will read the current row and TWO preceding to find the MAVG of those 3 rows. It will be sorted by Product_ID and Sale_Date first.

How the Moving Average is Calculated

```
SELECT Product_ID , Sale_Date, Daily_Sales,  
      MAVG( Daily_Sales, 3, Product_ID, Sale_Date) AS AVG3_Rows  
FROM Sales_Table
```

Not all rows
are displayed in
this answer set

<u>Product_ID</u>	<u>Sale Date</u>	<u>Daily Sales</u>	<u>AVG3 Rows</u>
1000	2000-09-28	48850.40	48850.40
1000	2000-09-29	54500.22	51675.31
1000	2000-09-30	36000.07	46450.23
1000	2000-10-01	40200.43	43566.91
1000	2000-10-02	32800.50	36333.67
1000	2000-10-03	64300.00	45788.98
1000	2000-10-04	54553.10	50551.20
2000	2000-09-28	41888.88	53580.66
2000	2000-09-29	48000.00	48147.33
2000	2000-09-30	49850.03	46579.11
2000	2000-10-01	54850.29	50900.11
2000	2000-10-02	36021.93	46907.42

With a Moving Window of 3, how is the 46450.23 amount derived in the **AVG3_Rows** column in the third row? It is the AVG of 48850.40, 54500.22 and 36000.07! The fourth row has **AVG3_Rows** equal to 43566.91. That was the average of 54500.22, 36000.07 and 40200.43. The calculation is on the current row and the two before.

How the Sort works for Moving Average (MAVG)

```
SELECT Product_ID , Sale_Date, Daily_Sales,  
      MAVG( Daily_Sales, 3, Product_ID, Sale_Date) AS AVG3_Rows  
FROM Sales_Table
```

Major and Minor
Sort keys

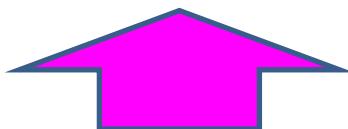
Product_ID	Sale_Date	Daily_Sales	AVG3_Rows
1000	2000-09-28	48850.40	48850.40
1000	2000-09-29	54500.22	51675.31
1000	2000-09-30	36000.07	46450.23
1000	2000-10-01	40200.43	43566.91
1000	2000-10-02	32800.50	36333.67
1000	2000-10-03	64300.00	45788.98
1000	2000-10-04	54553.10	50551.20
2000	2000-09-28	41888.88	53580.66
2000	2000-09-29	48000.00	48147.33
2000	2000-09-30	49850.03	46579.11
2000	2000-10-01	54850.29	50900.11
2000	2000-10-02	36021.93	46907.42

Not all rows
are displayed in
this answer set

The sorting is show above.

GROUP BY in the Moving Average does a Reset

```
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;
```



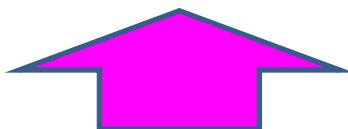
Not all rows
are displayed in
this answer set

<u>Product_ID</u>	<u>Sale Date</u>	<u>Daily_Sales</u>	<u>AVG3_Rows</u>
1000	2000-09-28	48850.40	48850.40
1000	2000-09-29	54500.22	51675.31
1000	2000-09-30	36000.07	46450.23
1000	2000-10-01	40200.43	43566.91
1000	2000-10-02	32800.50	36333.67
1000	2000-10-03	64300.00	45788.98
1000	2000-10-04	54553.10	50551.20
2000	2000-09-28	41888.88	41888.88
2000	2000-09-29	48000.00	44944.44
2000	2000-09-30	49850.03	46579.64

What does the GROUP BY Product_ID do? It causes a reset on all Product_ID breaks.

Quiz – Can you make the Advanced Calculation in your mind?

```
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;
```



Not all rows
are displayed in
this answer set

Product_ID	Sale_Date	Daily_Sales	AVG3_Rows
1000	2000-09-28	48850.40	48850.40
1000	2000-09-29	54500.22	51675.31
1000	2000-09-30	36000.07	46450.23
1000	2000-10-01	40200.43	43566.91
1000	2000-10-02	32800.50	36333.67
1000	2000-10-03	64300.00	45788.98
1000	2000-10-04	54553.10	50551.20
2000	2000-09-28	41888.88	41888.88
2000	2000-09-29	48000.00	44944.44
2000	2000-09-30	49850.03	46579.64
2000	2000-10-01	54850.29	50900.11

How is the 44944.44 derived in the 9th row of the AVG_for_3_Rows?

Answer to Quiz for the Advanced Calculation in your mind?

```
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;
```



Not all rows
are displayed in
this answer set

Product ID	Sale Date	Daily Sales	AVG3 Rows
1000	2000-09-28	48850.40	48850.40
1000	2000-09-29	54500.22	51675.31
1000	2000-09-30	36000.07	46450.23
1000	2000-10-01	40200.43	43566.91
1000	2000-10-02	32800.50	36333.67
1000	2000-10-03	64300.00	45788.98
1000	2000-10-04	54553.10	50551.20
2000	2000-09-28	41888.88	41888.88 ←→ 44944.44
2000	2000-09-29	48000.00	44944.44
2000	2000-09-30	49850.03	46579.64

AVG of **41888.88** and **48000.00**

Notice there are only two calculations although this has a moving window of 3. That is because the GROUP BY caused the MAVG to reset when Product_ID 2000 came.

Quiz – Write that Teradata Moving Average in ANSI Syntax

```
SELECT      Product_ID , Sale_Date, Daily_Sales,  
          MAVG( Daily_Sales, 3, Product_ID, Sale_Date) AS AVG3_Rows  
FROM Sales_Table ;
```

Challenge

Can you place
another **equivalent**
Moving Average
in the SQL above
using **ANSI Syntax**?

Here is a challenge that almost everyone fails. Can you do it perfectly?

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 ;
```

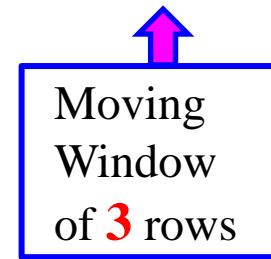
Product ID	Sale Date	Daily Sales	AVG 3	AVG 3 ANSI
1000	2000-09-28	48850.40	48850.40	48850.40
1000	2000-09-29	54500.22	51675.31	51675.31
1000	2000-09-30	36000.07	46450.23	46450.23
1000	2000-10-01	40200.43	43566.91	43566.91
1000	2000-10-02	32800.50	36333.67	36333.67
1000	2000-10-03	64300.00	45788.98	45788.98
1000	2000-10-04	54553.10	50551.20	50551.20
2000	2000-09-28	41888.88	53580.66	53580.66
2000	2000-09-29	48000.00	48147.33	48147.33
2000	2000-09-30	49850.03	46579.11	46579.11

Not all rows
are displayed in
this answer set

The MAVG and AVG(Over) commands above are equivalent. Notice the Moving Window of 3 in the Teradata syntax and that it is a 2 in the ANSI version. That is because in ANSI it is considered the Current Row and 2 preceding.

The ANSI Moving Window is Current Row and Preceding

```
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 ;
```



Calculate the
Current Row
and 2 rows
preceding

Product_ID	Sale Date	Daily Sales	AVG 3 ANSI
1000	2000-09-28	48850.40	48850.40
1000	2000-09-29	54500.22	51675.31
1000	2000-09-30	36000.07	46450.23
1000	2000-10-01	40200.43	43566.91
1000	2000-10-02	32800.50	36333.67
1000	2000-10-03	64300.00	45788.98
1000	2000-10-04	54553.10	50551.20
2000	2000-09-28	41888.88	53580.66
2000	2000-09-29	48000.00	48147.33
2000	2000-09-30	49850.03	46579.11

Not all rows
are displayed in
this answer set

The AVG () Over allows you to do is to get the moving average of a certain column.

How ANSI Moving Average Handles the Sort

```
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 ;
```

Major and Minor
Sort keys

Product_ID	Sale_Date	Daily_Sales	AVG_3_ANSI
1000	2000-09-28	48850.40	48850.40
1000	2000-09-29	54500.22	51675.31
1000	2000-09-30	36000.07	46450.23
1000	2000-10-01	40200.43	43566.91
1000	2000-10-02	32800.50	36333.67
1000	2000-10-03	64300.00	45788.98
1000	2000-10-04	54553.10	50551.20
2000	2000-09-28	41888.88	53580.66
2000	2000-09-29	48000.00	48147.33
2000	2000-09-30	49850.03	46579.11
2000	2000-10-01	54850.29	50900.11
2000	2000-10-02	36021.93	46907.42

Not all rows
are displayed in
this answer set

Much like the SUM OVER Command, the Average OVER places the sort after the ORDER BY.

Quiz – How is that 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 ;
```

<u>Product_ID</u>	<u>Sale Date</u>	<u>Daily Sales</u>	<u>AVG 3 ANSI</u>
1000	2000-09-28	48850.40	48850.40
1000	2000-09-29	54500.22	51675.31
1000	2000-09-30	36000.07	46450.23
1000	2000-10-01	40200.43	43566.91
1000	2000-10-02	32800.50	36333.67
1000	2000-10-03	64300.00	45788.98
1000	2000-10-04	54553.10	50551.20
2000	2000-09-28	41888.88	53580.66
2000	2000-09-29	48000.00	48147.33
2000	2000-09-30	49850.03	46579.11
2000	2000-10-01	54850.29	50900.11
2000	2000-10-02	36021.93	46907.42

Not all rows
are displayed in
this answer set

With a Moving Window of 3, how is the 46450.23 amount derived in the AVG_3_ANSI column in the third row?

Answer to Quiz – How is that 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 ;
```

Product ID	Sale Date	Daily Sales	AVG 3 ANSI
1000	2000-09-28	48850.40	48850.40
1000	2000-09-29	54500.22	51675.31
1000	2000-09-30	36000.07	46450.23
1000	2000-10-01	40200.43	43566.91
1000	2000-10-02	32800.50	36333.67
1000	2000-10-03	64300.00	45788.98
1000	2000-10-04	54553.10	50551.20
2000	2000-09-28	41888.88	53580.66
2000	2000-09-29	48000.00	48147.33
2000	2000-09-30	49850.03	46579.11
2000	2000-10-01	54850.29	50900.11
2000	2000-10-02	36021.93	46907.42

Not all rows
are displayed in
this answer set

AVG of **48850.40**, **54500.22**, and **36000.07**

Quiz – How is that 4th Row 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 ;
```

<u>Product ID</u>	<u>Sale Date</u>	<u>Daily Sales</u>	<u>AVG 3 ANSI</u>
1000	2000-09-28	48850.40	48850.40
1000	2000-09-29	54500.22	51675.31
1000	2000-09-30	36000.07	46450.23
1000	2000-10-01	40200.43	43566.91
1000	2000-10-02	32800.50	36333.67
1000	2000-10-03	64300.00	45788.98
1000	2000-10-04	54553.10	50551.20
2000	2000-09-28	41888.88	53580.66
2000	2000-09-29	48000.00	48147.33
2000	2000-09-30	49850.03	46579.11
2000	2000-10-01	54850.29	50900.11
2000	2000-10-02	36021.93	46907.42

Not all rows
are displayed in
this answer set

With a Moving Window of 3, how is the 43566.91 amount derived in the AVG_3_ANSI column in the fourth row?

Answer to Quiz – How is that 4th Row 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;
```

Product_ID	Sale Date	Daily Sales	AVG_3_ANSI
1000	2000-09-28	48850.40	48850.40
1000	2000-09-29	54500.22	51675.31
1000	2000-09-30	36000.07	46450.23
1000	2000-10-01	40200.43	43566.91
1000	2000-10-02	32800.50	36333.67
1000	2000-10-03	64300.00	45788.98
1000	2000-10-04	54553.10	50551.20
2000	2000-09-28	41888.88	53580.66
2000	2000-09-29	48000.00	48147.33
2000	2000-09-30	49850.03	46579.11
2000	2000-10-01	54850.29	50900.11
2000	2000-10-02	36021.93	46907.42

Not all rows
are displayed in
this answer set

AVG of **54500.22**, **36000.07**, and **40200.43**

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;
```

<u>Product_ID</u>	<u>Sale_Date</u>	<u>Daily_Sales</u>	<u>AVG3</u>	<u>Continuous</u>
1000	2000-09-28	48850.40	48850.40	48850.40
1000	2000-09-29	54500.22	51675.31	51675.31
1000	2000-09-30	36000.07	46450.23	46450.23
1000	2000-10-01	40200.43	43566.91	44887.78
1000	2000-10-02	32800.50	36333.67	42470.32
1000	2000-10-03	64300.00	45788.98	46108.60
1000	2000-10-04	54553.10	50551.20	47314.96
2000	2000-09-28	41888.88	53580.66	46636.70
2000	2000-09-29	48000.00	48147.33	46788.18

Not all rows
are displayed in
this answer set

The ROWS 2 Preceding gives the MAVG for every 3 rows. The ROWS UNBOUNDED Preceding gives the continuous MAVG.

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;
```

ANSI RESET
much Like a
GROUP BY

<u>Product_ID</u>	<u>Sale_Date</u>	<u>Daily_Sales</u>	<u>AVG3</u>	<u>Continuous</u>
1000	2000-09-28	48850.40	48850.40	48850.40
1000	2000-09-29	54500.22	51675.31	51675.31
1000	2000-09-30	36000.07	46450.23	46450.23
1000	2000-10-01	40200.43	43566.91	44887.78
1000	2000-10-02	32800.50	36333.67	42470.32
1000	2000-10-03	64300.00	45788.98	46108.60
1000	2000-10-04	54553.10	50551.20	47314.96
2000	2000-09-28	41888.88	53580.66	41888.88
2000	2000-09-29	48000.00	48147.33	44944.44

Not all rows
are displayed in
this answer set

Use a PARTITION BY Statement to Reset the ANSI OLAP.

The Moving Difference (MDIFF)

```
SELECT Product_ID , Sale_Date, Daily_Sales,  
MDIFF(Daily_Sales, 4, Product_ID, Sale_Date) as "MDiff"  
FROM Sales_Table ;
```

Moving
Difference

Not all rows
are displayed in
this answer set

Product_ID	Sale_Date	Daily_Sales	MDiff
1000	2000-09-28	48850.40	?
1000	2000-09-29	54500.22	?
1000	2000-09-30	36000.07	?
1000	2000-10-01	40200.43	?
1000	2000-10-02	32800.50	-16049.90
1000	2000-10-03	64300.00	9799.78
1000	2000-10-04	54553.10	18553.03
2000	2000-09-28	41888.88	1688.45
2000	2000-09-29	48000.00	15199.50
2000	2000-09-30	49850.03	-14449.97
2000	2000-10-01	54850.29	297.19
2000	2000-10-02	36021.93	-5866.95
2000	2000-10-03	43200.18	-4799.82
2000	2000-10-04	32800.50	-17049.53

This is the Moving Difference (MDIFF). What this does is calculate the difference between the current row and only the 4th row preceding.

Moving Difference (MDIFF) Visual

```
SELECT Product_ID , Sale_Date, Daily_Sales,  
       MDIFF(Daily_Sales, 4, Product_ID, Sale_Date) as "MDiff"  
FROM Sales_Table ;
```

<u>Product_ID</u>	<u>Sale Date</u>	<u>Daily_Sales</u>	<u>MDiff</u>
1000	2000-09-28	48850.40	?
1000	2000-09-29	54500.22	?
1000	2000-09-30	36000.07	?
1000	2000-10-01	40200.43	?
1000	2000-10-02	32800.50	-16049.90
1000	2000-10-03	64300.00	9799.78
1000	2000-10-04	54553.10	18553.03
2000	2000-09-28	41888.88	1688.45
2000	2000-09-29	48000.00	15199.50
2000	2000-09-30	49850.03	-14449.97
2000	2000-10-01	54850.29	297.19
2000	2000-10-02	36021.93	-5866.95
2000	2000-10-03	43200.18	-4799.82
2000	2000-10-04	32800.50	-17049.53

Not all rows
are displayed in
this answer set

How much more did we make for Product_ID 1000 on 2000-10-03 versus Product_ID 1000 which was 4 rows earlier on 2000-09-29?

Moving Difference using ANSI Syntax

```
SELECT Product_ID, Sale_Date, Daily_Sales,  
       Daily_Sales - SUM(Daily_Sales)  
   OVER ( ORDER BY Product_ID ASC, Sale_Date ASC  
          ROWS BETWEEN 4 PRECEDING AND 4 PRECEDING)  
          AS "MDiff_ANSI"  
  
FROM Sales_Table ;
```

Product_ID	Sale_Date	Daily_Sales	MDiff_ANSI
1000	2000-09-28	48850.40	?
1000	2000-09-29	54500.22	?
1000	2000-09-30	36000.07	?
1000	2000-10-01	40200.43	?
1000	2000-10-02	32800.50	-16049.90
1000	2000-10-03	64300.00	9799.78
1000	2000-10-04	54553.10	18553.03
2000	2000-09-28	41888.88	1688.45
2000	2000-09-29	48000.00	15199.50
2000	2000-09-30	49850.03	-14449.97
2000	2000-10-01	54850.29	297.19
2000	2000-10-02	36021.93	-5866.95

Not all rows
are displayed in
this answer set

This is how you do a MDiff using the ANSI Syntax with a moving window of 4.

Moving Difference using ANSI Syntax with Partition By

```
SELECT Product_ID, Sale_Date (Format 'yyyy-mm-dd'), Daily_Sales,  
Daily_Sales - SUM(Daily_Sales) OVER (PARTITION BY Product_ID  
ORDER BY Product_ID ASC, Sale_Date ASC  
ROWS BETWEEN 4 PRECEDING AND 4 PRECEDING)  
AS "MDiff_ANSI"
```

FROM Sales_Table;

Not all rows
are displayed in
this answer set

Product_ID	Sale_Date	Daily_Sales	MDiff_ANSI
1000	2000-09-28	48850.40	?
1000	2000-09-29	54500.22	?
1000	2000-09-30	36000.07	?
1000	2000-10-01	40200.43	?
1000	2000-10-02	32800.50	-16049.90
1000	2000-10-03	64300.00	9799.78
1000	2000-10-04	54553.10	18553.03
2000	2000-09-28	41888.88	?
2000	2000-09-29	48000.00	?
2000	2000-09-30	49850.03	?
2000	2000-10-01	54850.29	?
2000	2000-10-02	36021.93	-5866.95
2000	2000-10-03	43200.18	-4799.82
2000	2000-10-04	32800.50	-17049.53

Wow! This is how you do a MDiff using the ANSI Syntax with a PARTITION BY.

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 ;
```

Not all rows
are displayed in
this answer set



Product ID	Sale Date	Daily Sales	Compare2Rows
1000	2000-09-28	48850.40	?
1000	2000-09-29	54500.22	?
1000	2000-09-30	36000.07	?
1000	2000-10-01	40200.43	?
1000	2000-10-02	32800.50	?
1000	2000-10-03	64300.00	?
1000	2000-10-04	54553.10	?
2000	2000-09-28	41888.88	?
2000	2000-09-29	48000.00	?
2000	2000-09-30	49850.03	?
2000	2000-10-01	54850.29	?
2000	2000-10-02	36021.93	?
2000	2000-10-03	43200.18	?
2000	2000-10-04	32800.50	?

Do you notice that column Compare2Rows did not produce any data? That is because the GROUP BY Reset before it could get 7 records to find the MDIFF.

The RANK Command

```
SELECT      Product_ID ,Sale_Date , Daily_Sales,  
          RANK(Daily_Sales) AS "Rank"  
FROM Sales_Table WHERE Product_ID IN (1000, 2000);
```

<u>Product ID</u>	<u>Sale Date</u>	<u>Daily Sales</u>	<u>Rank</u>
1000	2000-10-03	64300.00	1
2000	2000-10-01	54850.29	2
1000	2000-10-04	54553.10	3
1000	2000-09-29	54500.22	4
2000	2000-09-30	49850.03	5
1000	2000-09-28	48850.40	6
2000	2000-09-29	48000.00	7
2000	2000-10-03	43200.18	8
2000	2000-09-28	41888.88	9
1000	2000-10-01	40200.43	10
2000	2000-10-02	36021.93	11
1000	2000-09-30	36000.07	12
2000	2000-10-04	32800.50	13
1000	2000-10-02	32800.50	13

This is the RANK. In this example, it will rank your Daily_Sales from greatest to least. The default for this type of RANK is to sort DESC.

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) ;
```

<u>Product_ID</u>	<u>Sale Date</u>	<u>Daily Sales</u>	<u>Rank</u>
1000	2000-10-02	32800.50	1
2000	2000-10-04	32800.50	1
1000	2000-09-30	36000.07	3
2000	2000-10-02	36021.93	4
1000	2000-10-01	40200.43	5
2000	2000-09-28	41888.88	6
2000	2000-10-03	43200.18	7
2000	2000-09-29	48000.00	8
1000	2000-09-28	48859.40	9
2000	2000-09-30	49850.03	10
1000	2000-09-29	54500.22	11
1000	2000-10-04	54553.10	12
2000	2000-10-01	54850.29	13
1000	2000-10-03	64300.00	14

This RANK query sorts in **Ascending mode**.

Two ways to get Rank to Sort in Ascending Order

```
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) ;
```

<u>Product_ID</u>	<u>Sale Date</u>	<u>Daily Sales</u>	<u>Rank1</u>	<u>Rank2</u>
1000	2000-10-02	32800.50	1	1
2000	2000-10-04	32800.50	1	1
1000	2000-09-30	36000.07	3	3
2000	2000-10-02	36021.93	4	4
1000	2000-10-01	40200.43	5	5
2000	2000-09-28	41888.88	6	6
2000	2000-10-03	43200.18	7	7
2000	2000-09-29	48000.00	8	8
1000	2000-09-28	48859.40	9	9
2000	2000-09-30	49850.03	10	10
1000	2000-09-29	54500.22	11	11
1000	2000-10-04	54553.10	12	12
2000	2000-10-01	54850.29	13	13
1000	2000-10-03	64300.00	14	14

A minus sign or keyword ASC will sort Both RANK in **Ascending mode**.

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)
```

Product_ID	Sale_Date	Daily_Sales	Rank1
2000	2000-10-04	32800.50	1
1000	2000-10-04	32800.50	1
1000	2000-09-30	65000.07	3
2000	2000-10-02	36021.93	4
1000	2000-10-01	40200.43	5
2000	2000-09-28	41888.88	6
2000	2000-10-03	43200.18	7
2000	2000-09-29	48000.00	8

Not all rows
are displayed in
this answer set

This is the RANK() OVER. It provides a rank for your queries. Notice how you do not place anything within the () after the word RANK. Default Sort is ASC.

Getting RANK using ANSI Syntax to Sort in DESC 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) ;
```

Not all rows
are displayed in
this answer set

<u>Product_ID</u>	<u>Sale Date</u>	<u>Daily Sales</u>	<u>Rank1</u>
2000	2000-09-29	48000.00	1
2000	2000-10-03	43200.00	2
2000	2000-09-28	41888.88	3
1000	2000-10-01	40200.43	4
2000	2000-10-02	36032.93	5
1000	2000-09-30	65000.07	6
1000	2000-10-04	32800.50	8
2000	2000-10-04	32800.50	8

Is the query above in ASC mode or DESC mode for sorting?

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
```

<u>Product ID</u>	<u>Sale Date</u>	<u>Daily Sales</u>	<u>Rank1</u>
1000	2000-10-03	65000.07	1
1000	2000-10-04	54553.10	2
1000	2000-09-29	54500.22	3
2000	2000-10-01	54850.29	1
2000	2000-09-30	49850.03	2
2000	2000-09-29	48000.00	3

What does the PARTITION Statement in the RANK() OVER do? It resets the rank.
The QUALIFY statement limits rows once the Rank's been calculated.

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 ;
```

<u>Product_ID</u>	<u>Sale Date</u>	<u>Daily Sales</u>	<u>Rank1</u>	<u>Rank2</u>
1000	2000-10-02	32800.50	1	1
2000	2000-10-04	32800.50	1	1
1000	2000-09-30	36000.07	3	3
2000	2000-10-02	36021.93	4	4
1000	2000-10-01	40200.43	5	5

The **WHERE** statement is performed first. It limits the rows being calculated. Then the **QUALIFY** takes the calculated rows and limits the returning rows. **QUALIFY** is to OLAP what **HAVING** is to Aggregates. Both limit after the calculations.

Quiz – How can you 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 Rank(-Daily_Sales) < 6 ;
```

<u>Product_ID</u>	<u>Sale_Date</u>	<u>Daily_Sales</u>	<u>Rank1</u>	<u>Rank2</u>
1000	2000-10-02	32800.50	1	1
2000	2000-10-04	32800.50	1	1
1000	2000-09-30	36000.07	3	3
2000	2000-10-02	36021.93	4	4
1000	2000-10-01	40200.43	5	5

How can you improve the QUALIFY Statement above for simplicity?

Answer to Quiz –Can you 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 ;
```

<u>Product_ID</u>	<u>Sale_Date</u>	<u>Daily_Sales</u>	<u>Rank1</u>	<u>Rank2</u>
1000	2000-10-02	32800.50	1	1
2000	2000-10-04	32800.50	1	1
1000	2000-09-30	36000.07	3	3
2000	2000-10-02	36021.93	4	4
1000	2000-10-01	40200.43	5	5

QUALIFY Rank2 < 6
(Use the Alias)

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 ;
```

<u>Product_ID</u>	<u>Sale_Date</u>	<u>Daily_Sales</u>	<u>Rank1</u>
1000	2000-10-03	64300.00	1
2000	2000-10-01	54850.29	2
1000	2000-10-04	54553.10	3
1000	2000-09-29	54500.22	4
2000	2000-09-30	49850.03	5

A QUALIFY < 6 will provide a result that is 5 rows. Notice there are NO ties, yet!

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 ;
```

<u>Product_ID</u>	<u>Sale Date</u>	<u>Daily Sales</u>	<u>Rank1</u>
1000	2000-10-02	32800.50	1
2000	2000-10-04	32800.50	1
1000	2000-09-30	36000.07	3
2000	2000-10-02	36021.93	4
1000	2000-10-01	40200.43	5

A QUALIFY < 6 will provide a result that is 5 rows. Notice there are Ties!

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 ;
```

<u>Product_ID</u>	<u>Sale Date</u>	<u>Daily Sales</u>	<u>Rank1</u>
1000	2000-10-02	32800.50	1
2000	2000-10-04	32800.50	1

A QUALIFY < 2 will provide more rows than 1 because of the Ties!

Mixing Sort Order for 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 ;
```

The diagram illustrates the mixed sort order. A blue arrow points from a red box labeled "DESC Mode" to the RANK clause, indicating that the ranking is done in descending order. Another red arrow points from a blue box labeled "ASC Mode" to the QUALIFY clause, indicating that only rows with a rank less than 6 are selected, which is equivalent to an ascending sort.

<u>Product ID</u>	<u>Sale Date</u>	<u>Daily Sales</u>	<u>Rank1</u>
1000	2000-10-02	32800.50	13
2000	2000-10-04	32800.50	13
1000	2000-09-30	36000.07	12
2000	2000-10-02	36021.93	11
1000	2000-10-01	40200.43	10

Look at the Rankings and the Daily_Sales. This data come out odd because Rank1 is DESC by default (using this Syntax) and the QUALIFY specifies ASC mode.

Quiz – What Caused the 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 ;
```

<u>Product_ID</u>	<u>Sale_Date</u>	<u>Daily_Sales</u>	<u>Rank1</u>
1000	2000-10-03	64300.00	1
1000	2000-10-04	54553.10	2
1000	2000-09-29	54500.22	3
2000	2000-10-01	54850.29	1
2000	2000-09-30	49850.03	2
2000	2000-09-29	48000.00	3

What caused the data to reset the column Rank1?

Answer to Quiz – What Caused the 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 ;
```

<u>Product_ID</u>	<u>Sale_Date</u>	<u>Daily_Sales</u>	<u>Rank1</u>
1000	2000-10-03	64300.00	1
1000	2000-10-04	54553.10	2
1000	2000-09-29	54500.22	3
2000	2000-10-01	54850.29	1
2000	2000-09-30	49850.03	2
2000	2000-09-29	48000.00	3

GROUP BY

Quiz – Name those Sort Orders

RANK() OVER (ORDER BY Daily_Sales) AS **ANSI_Rank**

Is the default above **ASC** or **DESC**?

RANK(Daily_Sales) AS **NON_ANSI_Rank**

Is the default above **ASC** or **DESC**?

Answer the questions above.

Answer to Quiz – Name those Sort Orders

RANK() OVER (ORDER BY Daily_Sales) AS **ANSI_Rank**

Defaults to **ASC**

RANK(Daily_Sales) AS **NON_ANSI_Rank**

Defaults to **DESC**

Please note that by default these different syntaxes sort completely opposite.

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) ;
```

<u>Product_ID</u>	<u>Sale Date</u>	<u>Daily Sales</u>	<u>PercentRank1</u>
1000	2000-10-03	64300.00	0.000000
1000	2000-10-04	54553.10	0.166667
1000	2000-09-29	54500.22	0.333333
1000	2000-09-28	48850.40	0.500000
1000	2000-10-01	40200.43	0.666667
1000	2000-09-30	36000.07	0.833333
1000	2000-10-02	32800.50	1.000000
2000	2000-10-01	54850.29	0.000000
2000	2000-09-30	49850.03	0.166667
2000	2000-09-29	48000.00	0.333333
2000	2000-10-03	43200.18	0.500000
2000	2000-09-28	41888.88	0.666667
2000	2000-10-02	36021.93	0.833333
2000	2000-10-04	32800.50	1.000000

7 Rows in
Calculation
for 1000
Product_ID

7 Rows in
Calculation
for 2000
Product_ID

We now have added a Partition statement which produces 7 rows per Product_ID.

PERCENT_RANK() OVER with 14 rows in Calculation

```
SELECT Product_ID ,Sale_Date , Daily_Sales,  
       PERCENT_RANK() OVER ( ORDER BY Daily_Sales DESC)  
                           AS PercentRank1  
FROM Sales_Table WHERE Product_ID IN (1000, 2000) ;
```

Product ID	Sale Date	Daily Sales	PercentRank1
1000	2000-10-03	64300.00	0.000000
2000	2000-10-01	54850.29	0.076923
1000	2000-10-04	54553.10	0.153846
1000	2000-09-29	54500.22	0.230769
2000	2000-09-30	49850.03	0.307692
1000	2000-09-28	48850.40	0.384615
2000	2000-09-29	48000.00	0.461538
2000	2000-10-03	43200.18	0.538462
2000	2000-09-28	41888.88	0.615385
1000	2000-10-01	40200.43	0.692308
2000	2000-10-02	36021.93	0.769231
1000	2000-09-30	36000.07	0.846154
2000	2000-10-04	32800.50	0.923077
1000	2000-10-02	32800.50	0.923077

14 Rows in
Calculation
for both the
1000 and 2000
Product IDs

Percentage_Rank is just like RANK however, it gives you the Rank as a percent, but only a percent of all the other rows up to 100%.

PERCENT_RANK() OVER with 21 rows in Calculation

```
SELECT Product_ID ,Sale_Date , Daily_Sales,  
       PERCENT_RANK() OVER ( ORDER BY Daily_Sales DESC)  
                           AS PercentRank1  
FROM   Sales_Table ;
```

Product_ID	Sale Date	Daily_Sales	PercentRank1
1000	2000-10-03	64300.00	0.000000
3000	2000-09-28	61301.77	0.050000
2000	2000-10-01	54850.29	0.100000
1000	2000-10-04	54553.10	0.150000
1000	2000-09-29	54500.22	0.200000
2000	2000-09-30	49850.03	0.250000
1000	2000-09-28	48850.40	0.300000
2000	2000-09-29	48000.00	0.350000
3000	2000-09-30	43868.86	0.400000
2000	2000-10-03	43200.18	0.450000
2000	2000-09-28	41888.88	0.500000
1000	2000-10-01	40200.43	0.550000
2000	2000-10-02	36021.93	0.600000
1000	2000-09-30	36000.07	0.650000

Not all rows
are displayed in
this answer set

21 Rows in
Calculation
for all of the
Product_IDs

Percentage_Rank is just like RANK however, it gives you the Rank as a percent, but only a percent of all the other rows up to 100%.

Quiz – What Cause 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) ;
```

<u>Product_ID</u>	<u>Sale Date</u>	<u>Daily_Sales</u>	<u>PercentRank1</u>
1000	2000-10-03	64300.00	0.000000
1000	2000-10-04	54553.10	0.166667
1000	2000-09-29	54500.22	0.333333
1000	2000-09-28	48850.40	0.500000
1000	2000-10-01	40200.43	0.666667
1000	2000-09-30	36000.07	0.833333
1000	2000-10-02	32800.50	1.000000
2000	2000-10-01	54850.29	0.000000
2000	2000-09-30	49850.03	0.166667
2000	2000-09-29	48000.00	0.333333
2000	2000-10-03	43200.18	0.500000
2000	2000-09-28	41888.88	0.666667
2000	2000-10-02	36021.93	0.833333
2000	2000-10-04	32800.50	1.000000

What caused the Product_IDs to be sorted?

Answer to Quiz – What Cause 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) ;
```

<u>Product_ID</u>	<u>Sale Date</u>	<u>Daily_Sales</u>	<u>PercentRank1</u>
1000	2000-10-03	64300.00	0.000000
1000	2000-10-04	54553.10	0.166667
1000	2000-09-29	54500.22	0.333333
1000	2000-09-28	48850.40	0.500000
1000	2000-10-01	40200.43	0.666667
1000	2000-09-30	36000.07	0.833333
1000	2000-10-02	32800.50	1.000000
2000	2000-10-01	54850.29	0.000000
2000	2000-09-30	49850.03	0.166667
2000	2000-09-29	48000.00	0.333333
2000	2000-10-03	43200.18	0.500000
2000	2000-09-28	41888.88	0.666667
2000	2000-10-02	36021.93	0.833333
2000	2000-10-04	32800.50	1.000000

PARTITION BY caused the data to be sorted!

COUNT OVER for a Sequential Number

```
SELECT Product_ID ,Sale_Date , Daily_Sales,  
       COUNT(*) OVER (ORDER BY Product_ID, Sale_Date  
                      ROWS UNBOUNDED PRECEDING) AS Seq_Number  
FROM Sales_Table WHERE Product_ID IN (1000, 2000) ;
```

Product_ID	Sale_Date	Daily_Sales	Seq_Number
1000	2000-09-28	48850.40	1
1000	2000-09-29	54500.22	2
1000	2000-09-30	36000.07	3
1000	2000-10-01	40200.43	4
1000	2000-10-02	32800.50	5
1000	2000-10-03	64300.00	6
1000	2000-10-04	54553.10	7
2000	2000-09-28	41888.88	8
2000	2000-09-29	48000.00	9
2000	2000-09-30	49850.03	10
2000	2000-10-01	54850.29	11

Not all rows
are displayed in
this answer set

This is the COUNT OVER. It will provide a sequential number starting at 1. The Keyword(s) ROWS UNBOUNDED PRECEDING causes Seq_Number to start at the beginning and increase sequentially to the end.

Troubleshooting COUNT OVER

```
SELECT      Product_ID ,Sale_Date , Daily_Sales,  
COUNT(*) OVER (ORDER BY Product_ID, Sale_Date) AS No_Seq  
FROM Sales_Table WHERE Product_ID IN (1000, 2000) ;
```

Rows Unbounded Preceding is missing in this statement.

<u>Product_ID</u>	<u>Sale Date</u>	<u>Daily Sales</u>	<u>Seq Number</u>
1000	2000-09-28	48850.40	14
1000	2000-09-29	54500.22	14
1000	2000-09-30	36000.07	14
1000	2000-10-01	40200.43	14
1000	2000-10-02	32800.50	14
1000	2000-10-03	64300.00	14
1000	2000-10-04	54553.10	14
2000	2000-09-28	41888.88	14
2000	2000-09-29	48000.00	14
2000	2000-09-30	49850.03	14
2000	2000-10-01	54850.29	14
2000	2000-10-02	36021.93	14
2000	2000-10-03	43200.18	14
2000	2000-10-04	32800.50	14

14 rows
came back

When you don't have a ROWS UNBOUNDED PRECEDING, No_Seq get a value of 14 on every row. Why? Because 14 is the FINAL COUNT NUMBER.

Quiz – What caused the COUNT OVER to Reset?

```
SELECT Product_ID ,Sale_Date , Daily_Sales,  
       COUNT(*) OVER (PARTITION BY Product_ID  
                           ORDER BY Product_ID, Sale_Date  
                           ROWS UNBOUNDED PRECEDING) AS StartOver  
FROM Sales_Table WHERE Product_ID IN (1000, 2000) ;
```

<u>Product_ID</u>	<u>Sale_Date</u>	<u>Daily_Sales</u>	<u>StartOver</u>
1000	2000-09-28	48850.40	1
1000	2000-09-29	54500.22	2
1000	2000-09-30	36000.07	3
1000	2000-10-01	40200.43	4
1000	2000-10-02	32800.50	5
1000	2000-10-03	64300.00	6
1000	2000-10-04	54553.10	7
2000	2000-09-28	41888.88	1
2000	2000-09-29	48000.00	2
2000	2000-09-30	49850.03	3
2000	2000-10-01	54850.29	4
2000	2000-10-02	36021.93	5
2000	2000-10-03	43200.18	6
2000	2000-10-04	32800.50	7

What Keyword(s) caused StartOver to reset?

Answer to Quiz – What caused the COUNT OVER to Reset?

```
SELECT Product_ID ,Sale_Date , Daily_Sales,  
       COUNT(*) OVER (PARTITION BY Product_ID  
                           ORDER BY Product_ID, Sale_Date  
                           ROWS UNBOUNDED PRECEDING) AS StartOver  
FROM Sales_Table WHERE Product_ID IN (1000, 2000) ;
```

<u>Product_ID</u>	<u>Sale_Date</u>	<u>Daily_Sales</u>	<u>StartOver</u>
1000	2000-09-28	48850.40	1
1000	2000-09-29	54500.22	2
1000	2000-09-30	36000.07	3
1000	2000-10-01	40200.43	4
1000	2000-10-02	32800.50	5
1000	2000-10-03	64300.00	6
1000	2000-10-04	54553.10	7
2000	2000-09-28	41888.88	1
2000	2000-09-29	48000.00	2
2000	2000-09-30	49850.03	3
2000	2000-10-01	54850.29	4
2000	2000-10-02	36021.93	5
2000	2000-10-03	43200.18	6
2000	2000-10-04	32800.50	7

PARTITION BY

The MAX OVER Command

```
SELECT Product_ID ,Sale_Date , Daily_Sales,  
      MAX(Daily_Sales) OVER (ORDER BY Product_ID, Sale_Date  
                                ROWS UNBOUNDED PRECEDING) AS MaxOver  
FROM Sales_Table WHERE Product_ID IN (1000, 2000) ;
```

<u>Product_ID</u>	<u>Sale Date</u>	<u>Daily_Sales</u>	<u>MaxOver</u>
1000	2000-09-28	48850.40	48850.40
1000	2000-09-29	54500.22	54500.22
1000	2000-09-30	36000.07	54500.22
1000	2000-10-01	40200.43	54500.22
1000	2000-10-02	32800.50	54500.22
1000	2000-10-03	64300.00	64300.00
1000	2000-10-04	54553.10	64300.00
2000	2000-09-28	41888.88	64300.00
2000	2000-09-29	48000.00	64300.00
2000	2000-09-30	49850.03	64300.00
2000	2000-10-01	54850.29	64300.00
2000	2000-10-02	36021.93	64300.00
2000	2000-10-03	43200.18	64300.00
2000	2000-10-04	32800.50	64300.00

After the sort the Max() Over shows the Max Value up to that point.

MAX OVER with PARTITION BY Reset

```
SELECT Product_ID ,Sale_Date , Daily_Sales,  
       MAX(Daily_Sales) OVER (PARTITION BY Product_ID  
                                ORDER BY Product_ID, Sale_Date  
                                ROWS UNBOUNDED PRECEDING) AS MaxOver  
FROM Sales_Table WHERE Product_ID IN (1000, 2000) ;
```

Product_ID	Sale Date	Daily_Sales	MaxOver
1000	2000-09-28	48850.40	48850.40
1000	2000-09-29	54500.22	54500.22
1000	2000-09-30	36000.07	54500.22
1000	2000-10-01	40200.43	54500.22
1000	2000-10-02	32800.50	54500.22
1000	2000-10-03	64300.00	64300.00
1000	2000-10-04	54553.10	64300.00
2000	2000-09-28	41888.88	41888.88
2000	2000-09-29	48000.00	48000.00
2000	2000-09-30	49850.03	49850.03
2000	2000-10-01	54850.29	54850.29

Not all rows
are displayed in
this answer set

The largest value is 64300.00 in the column MaxOver. Once it was evaluated it did not continue until the end because of the PARTITION BY reset.

Troubleshooting MAX OVER

```
SELECT Product_ID ,Sale_Date , Daily_Sales,  
      MAX(Daily_Sales) OVER (ORDER BY Product_ID, Sale_Date )  
                                AS MaxOver  
FROM Sales_Table WHERE Product_ID IN (1000, 2000) ;
```

Rows Unbounded Preceding is missing in this statement.

Product_ID	Sale Date	Daily_Sales	MaxOver
1000	2000-09-28	48850.40	64300.00
1000	2000-09-29	54500.22	64300.00
1000	2000-09-30	36000.07	64300.00
1000	2000-10-01	40200.43	64300.00
1000	2000-10-02	32800.50	64300.00
1000	2000-10-03	64300.00	64300.00
1000	2000-10-04	54553.10	64300.00
2000	2000-09-28	41888.88	64300.00
2000	2000-09-29	48000.00	64300.00
2000	2000-09-30	49850.03	64300.00

Not all rows
are displayed in
this answer set

You can also use MAX as a OLAP. 64300.00 came back in MaxOver because that was the MAX value for Daily_Sales in this Answer Set. Notice that it **doesn't** have a **ROWS UNBOUNDED PRECEDING**.

The MIN OVER Command

```
SELECT Product_ID, Sale_Date ,Daily_Sales  
      ,Min(Daily_Sales) OVER (ORDER BY Product_ID, Sale_Date  
                                ROWS UNBOUNDED PRECEDING) AS MinOver  
FROM Sales_Table WHERE Product_ID IN (1000, 2000) ;
```

<u>Product_ID</u>	<u>Sale Date</u>	<u>Daily_Sales</u>	<u>MinOver</u>
1000	2000-09-28	48850.40	48850.40
1000	2000-09-29	54500.22	48850.40
1000	2000-09-30	36000.07	36000.07
1000	2000-10-01	40200.43	36000.07
1000	2000-10-02	32800.50	32800.50
1000	2000-10-03	64300.00	32800.50
1000	2000-10-04	54553.10	32800.50
2000	2000-09-28	41888.88	32800.50
2000	2000-09-29	48000.00	32800.50
2000	2000-09-30	49850.03	32800.50
2000	2000-10-01	54850.29	32800.50
2000	2000-10-02	36021.93	32800.50
2000	2000-10-03	43200.18	32800.50
2000	2000-10-04	32800.50	32800.50

After the sort the MIN () Over shows the Max Value up to that point.

Troubleshooting MIN OVER

```
SELECT Product_ID ,Sale_Date , Daily_Sales,  
       MIN(Daily_Sales) OVER (ORDER BY Product_ID, Sale_Date )  
                           AS MinOver  
FROM Sales_Table WHERE Product_ID IN (1000, 2000) ;
```

Rows Unbounded Preceding is missing in this statement.

Product ID	Sale Date	Daily Sales	MinOver
1000	2000-09-28	48850.40	32800.50
1000	2000-09-29	54500.22	32800.50
1000	2000-09-30	36000.07	32800.50
1000	2000-10-01	40200.43	32800.50
1000	2000-10-02	32800.50	32800.50
1000	2000-10-03	64300.00	32800.50
1000	2000-10-04	54553.10	32800.50
2000	2000-09-28	41888.88	32800.50
2000	2000-09-29	48000.00	32800.50
2000	2000-09-30	49850.03	32800.50
2000	2000-10-01	54850.29	32800.50

Not all rows
are displayed in
this answer set

Min only displayed 32800.50 because there is NOT a ROWS UNBOUNDED PRECEDING statement so it found the lowest Daily_Sales and repeated it.

Quiz – Fill in the Blank

```
SELECT Product_ID ,Sale_Date , Daily_Sales,  
       Min(Daily_Sales) OVER (PARTITION BY Product_ID  
                               ORDER BY Product_ID, Sale_Date  
                               ROWS UNBOUNDED PRECEDING) AS MinOver  
FROM Sales_Table WHERE Product_ID IN (1000, 2000) ;
```

<u>Product_ID</u>	<u>Sale Date</u>	<u>Daily_Sales</u>	<u>MinOver</u>
1000	2000-09-28	48850.40	48850.40
1000	2000-09-29	54500.22	48850.40
1000	2000-09-30	36000.07	36000.07
1000	2000-10-01	40200.43	36000.07
1000	2000-10-02	32800.50	32800.50
1000	2000-10-03	64300.00	32800.50
1000	2000-10-04	54553.10	32800.50
2000	2000-09-28	41888.88	41888.88
2000	2000-09-29	48000.00	41888.88
2000	2000-09-30	49850.03	41888.88
2000	2000-10-01	54850.29	41888.88
2000	2000-10-02	36021.93	36021.93
2000	2000-10-03	43200.18	
2000	2000-10-04	32800.50	

The last two answers (MinOver) are blank so you can fill in the blank.

Answer to Quiz – Fill in the Blank

```
SELECT Product_ID ,Sale_Date , Daily_Sales,  
Min(Daily_Sales) OVER (PARTITION BY Product_ID  
ORDER BY Product_ID, Sale_Date  
ROWS UNBOUNDED PRECEDING) AS MinOver  
FROM Sales_Table WHERE Product_ID IN (1000, 2000) ;
```

<u>Product_ID</u>	<u>Sale_Date</u>	<u>Daily_Sales</u>	<u>MinOver</u>
1000	2000-09-28	48850.40	48850.40
1000	2000-09-29	54500.22	48850.40
1000	2000-09-30	36000.07	36000.07
1000	2000-10-01	40200.43	36000.07
1000	2000-10-02	32800.50	32800.50
1000	2000-10-03	64300.00	32800.50
1000	2000-10-04	54553.10	32800.50
2000	2000-09-28	41888.88	41888.88
2000	2000-09-29	48000.00	41888.88
2000	2000-09-30	49850.03	41888.88
2000	2000-10-01	54850.29	41888.88
2000	2000-10-02	36021.93	36021.93
2000	2000-10-03	43200.18	36021.93
2000	2000-10-04	32800.50	32800.50

The Row_Number Command

```
SELECT Product_ID ,Sale_Date , Daily_Sales,  
       ROW_NUMBER() OVER (ORDER BY Product_ID, Sale_Date)  
             AS Seq_Number  
FROM Sales_Table WHERE Product_ID IN (1000, 2000) ;
```

Product_ID	Sale_Date	Daily_Sales	Seq_Number
1000	2000-09-28	48850.40	1
1000	2000-09-29	54500.22	2
1000	2000-09-30	36000.07	3
1000	2000-10-01	40200.43	4
1000	2000-10-02	32800.50	5
1000	2000-10-03	64300.00	6
1000	2000-10-04	54553.10	7
2000	2000-09-28	41888.88	8
2000	2000-09-29	48000.00	9
2000	2000-09-30	49850.03	10
2000	2000-10-01	54850.29	11

Not all rows
are displayed in
this answer set

The **ROW_NUMBER()** Keyword(s) caused **Seq_Number** to increase sequentially.
Notice that this does **NOT** have a **Rows Unbounded Preceding** and it still works!

Quiz – How did the Row_Number Reset?

```
SELECT Product_ID ,Sale_Date , Daily_Sales,  
       ROW_NUMBER() OVER (PARTITION BY Product_ID  
                           ORDER BY Product_ID, Sale_Date ) AS StartOver  
FROM Sales_Table WHERE Product_ID IN (1000, 2000) ;
```

<u>Product_ID</u>	<u>Sale Date</u>	<u>Daily_Sales</u>	<u>StartOver</u>
1000	2000-09-28	48850.40	1
1000	2000-09-29	54500.22	2
1000	2000-09-30	36000.07	3
1000	2000-10-01	40200.43	4
1000	2000-10-02	32800.50	5
1000	2000-10-03	64300.00	6
1000	2000-10-04	54553.10	7
2000	2000-09-28	41888.88	1
2000	2000-09-29	48000.00	2
2000	2000-09-30	49850.03	3
2000	2000-10-01	54850.29	4
2000	2000-10-02	36021.93	5
2000	2000-10-03	43200.18	6
2000	2000-10-04	32800.50	7

What Keyword(s) caused StartOver to **reset**?

Quiz – How did the Row_Number Reset?

```
SELECT Product_ID ,Sale_Date , Daily_Sales,  
       ROW_NUMBER() OVER (PARTITION BY Product_ID  
                           ORDER BY Product_ID, Sale_Date ) AS StartOver  
FROM Sales_Table WHERE Product_ID IN (1000, 2000) ;
```

<u>Product_ID</u>	<u>Sale Date</u>	<u>Daily Sales</u>	<u>StartOver</u>
1000	2000-09-28	48850.40	1
1000	2000-09-29	54500.22	2
1000	2000-09-30	36000.07	3
1000	2000-10-01	40200.43	4
1000	2000-10-02	32800.50	5
1000	2000-10-03	64300.00	6
1000	2000-10-04	54553.10	7
2000	2000-09-28	41888.88	1
2000	2000-09-29	48000.00	2
2000	2000-09-30	49850.03	3
2000	2000-10-01	54850.29	4
2000	2000-10-02	36021.93	5
2000	2000-10-03	43200.18	6
2000	2000-10-04	32800.50	7

What Keyword(s) caused StartOver to reset?

PARTITION BY

Testing Your Knowledge

```
SELECT Product_ID , Sale_Date, Daily_Sales,  
      CSUM(Daily_Sales, Product_ID, Sale_Date) AS "CSum"  
FROM Sales_Table WHERE Product_ID BETWEEN 1000 and 2000  
GROUP BY Product_ID ;
```

This is the **CSUM**. However, what we want to see is the **Sum()Over ANSI version**. Use the information in the **CSUM** and **convert this to the equivalent Sum()Over**.

Testing Your Knowledge

```
SELECT Product_ID , Sale_Date, Daily_Sales,  
      CSUM(Daily_Sales, Product_ID, Sale_Date) AS "CSum"  
FROM Sales_Table WHERE Product_ID BETWEEN 1000 and 2000  
GROUP BY Product_ID ;
```

```
SELECT Product_ID , Sale_Date, Daily_Sales,  
      SUM(Daily_Sales) OVER (PARTITION BY Product_ID  
                             ORDER BY Product_ID, Sale_Date  
                             ROWS UNBOUNDED PRECEDING) AS SumANSI  
FROM Sales_Table  
WHERE Product_ID BETWEEN 1000 and 2000 ;
```

Both statements are exactly the same except the bottom example uses ANSI syntax.

Testing Your Knowledge

```
SELECT Product_ID , Sale_Date, Daily_Sales,  
      MAVG( Daily_Sales, 3, Product_ID, Sale_Date) AS AVG_for_3_Rows  
FROM Sales_Table WHERE Product_ID BETWEEN 1000 and 2000 ;
```

Write the equivalent to the SQL above using ANSI Syntax such as AVG () Over.

Testing Your Knowledge

```
SELECT Product_ID , Sale_Date, Daily_Sales,  
MAVG( Daily_Sales, 3, Product_ID, Sale_Date) AS AVG_for_3_Rows  
FROM Sales_Table WHERE Product_ID BETWEEN 1000 and 2000 ;
```

```
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 WHERE Product_ID BETWEEN 1000 and 2000 ;
```

The SQL above is equivalent except the bottom example uses ANSI Syntax.

Testing Your Knowledge

```
SELECT Product_ID ,Sale_Date , Daily_Sales,  
      RANK(Daily_Sales) AS "Rank"  
FROM Sales_Table  
WHERE Product_ID IN (1000, 2000) ;
```

This is the Rank. However, what we want to see is the RANK()Over. Use the information in the Rank to make it the Rank()Over.

Testing Your Knowledge

```
SELECT Product_ID ,Sale_Date , Daily_Sales,  
      RANK(Daily_Sales) AS "Rank"  
FROM Sales_Table  
WHERE Product_ID IN (1000, 2000) ;
```

```
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)
```

The SQL above is equivalent except the bottom example uses ANSI Syntax. Also notice the sort key. DESC is the default in the top example.

Chapter 22

The Quantile Function

“There is flattery in friendship.”

- William Shakespeare

Table of Contents Chapter 22 - The Quantile Function

- [The Quantile Function and Syntax](#)
- [A Quantile Example](#)
- [A Quantile Example using DESC Mode](#)
- [QUALIFY to find Products in the top Partitions](#)
- [QUALIFY to find Products in the top Partitions Sorted DESC](#)
- [QUALIFY to find Products in the top Partitions Sorted ASC](#)
- [QUALIFY to find Products in top Partitions with Tiebreaker](#)
- [Using Tertiles \(Partitions of Four\)](#)
- [How Quantile Works](#)

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 here is daily sales.

```
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 uses 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 SELECT 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.

Chapter 23

Temporary Tables

“Those who would give up essential liberty, to purchase a little temporary safety, deserve neither liberty nor safety.”

-Benjamin Franklin

Table of Contents Chapter 23 - Temporary Tables

- [There are Three types of Temporary Tables](#)
- [CREATING A Derived Table](#)
- [Naming the Derived Table](#)
- [Aliasing the Columns of the Derived Table](#)
- [Multiple Ways to Alias the Columns in a Derived Table](#)
- [CREATING A Derived Table using the WITH Command](#)
- [Naming the Derived Table using the WITH Command](#)
- [Naming the Derived Table Columns using WITH](#)
- [The Same Derived Query shown Three Different Ways](#)
- [A Derived Table that Joins to an Existing Table](#)
- [Quiz - Answer the Questions](#)
- [Answer to Quiz - Answer the Questions](#)
- [Clever Tricks on Aliasing Columns in a Derived Table](#)
- [A Derived Table lives only for the lifetime of a single query](#)
- [An Example of Two Derived Tables in a Single Query](#)
- [Creating a Volatile Table](#)
- [You Populate a Volatile Table with an INSERT/SELECT](#)
- [The Three Steps to Use a Volatile Table](#)
- [The HELP Volatile Table Command Shows your Volatiles](#)
- [CREATING A Global Temporary Table](#)
- [Populating A Global Temporary Table with INSERT/SELECT](#)
- [The Three Steps to using a Global Temporary Table](#)

There are Three types of Temporary Tables

Derived Table

Exists only within a query

Materialized by a **SELECT Statement** inside a query

Space comes from the User's **Spool** space

Deleted when the **query ends**

Volatile

Created by the **User** and materialized with an **INSERT/SELECT**

Space comes from the User's **Spool** space

Table and Data are deleted only after a User **Logs off** the session

Global Temporary Table

Table definition is created by a User and the **table definition** is **permanent**

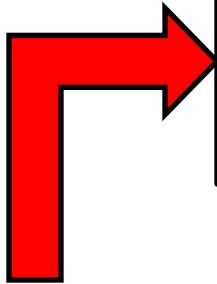
Materialized with an **INSERT/SELECT**

Space comes from the User's **TEMP** Space

When **User logs off** the session the **data is deleted**, but the **table definition stays**

Many Users can populate the same **Global table**, but each has **their own copy**

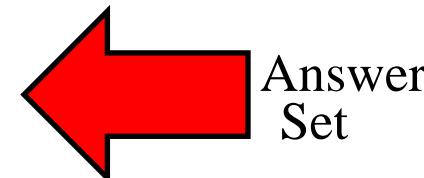
CREATING A Derived Table



```
SELECT *
FROM (SELECT AVG(salary) FROM
Employee_Table) TeraTom(AVGSAL);
```

A query within a
query.

AVGSAL
46782.15

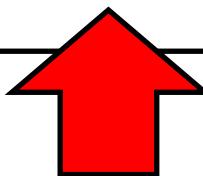


Answer
Set

The **SELECT Statement** that creates and populates the Derived table **is always inside Parentheses.**

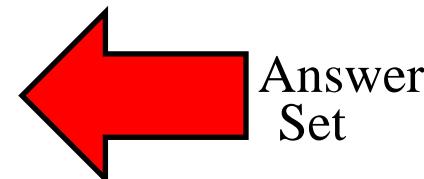
Naming the Derived Table

```
SELECT *
FROM  (SELECT AVG(salary) FROM
Employee_Table) TeraTom(AVGSAL);
```



The name of the
Derived Table

AVGSAL
46782.15



Answer
Set

In the example above, **TeraTom** is the name we gave the Derived Table. It is **mandatory** that you **always** name the table or it errors.

Aliasing the Columns of the Derived Table

```
SELECT *  
FROM (SELECT AVG(salary) FROM  
Employee_Table) TeraTom(AVGSAL);
```

Alias CAN be done
here

AVGSAL
46782.15

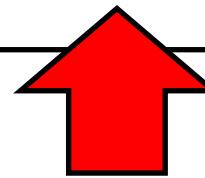
Answer
Set

AVGSAL is the **name** we gave to the **column** in our Derived Table that we call TeraTom. Our SELECT (which builds the columns) shows we are only going to have one column in our derived table and we have named that column AVGSAL.

Multiple Ways to Alias the Columns in a Derived Table

1

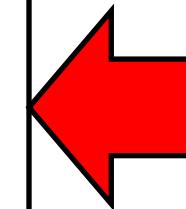
```
SELECT *
FROM (SELECT AVG(salary) FROM
Employee_Table) TeraTom(AVGSAL) ;
```



Alias CAN be done
here

2

```
SELECT *
FROM (SELECT AVG(salary) as AVGSAL
FROM Employee_Table) TeraTom ;
```



or
here

The queries above produce the same result as they are equivalent. The only difference is that we specifically named our column AVGSAL in a different place.

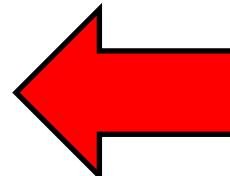
CREATING A Derived Table using the WITH Command

Creating the Derived Table
Before we run the query!



```
WITH TeraTom(AVGSAL) AS  
(SELECT AVG(salary)  
FROM Employee_Table)  
SELECT * FROM TeraTom ;
```

AVGSAL
46782.15



Answer Set

When using the WITH Command, we can CREATE our Derived table before running the main query. The only issue here is that you can only have 1 WITH.

Naming the Derived Table using the WITH Command

Name of the Derived Table



```
WITH TeraTom(AVGSAL) AS  
(SELECT AVG(salary)  
FROM Employee_Table)  
SELECT * FROM TeraTom ;
```

AVGSAL
46782.15

A large red arrow points from the text "Answer Set" towards the result value "46782.15".

Answer
Set

Using the **WITH command**, you must place the name of the table as you see it in the example. TeraTom is the name of our **Derived Table**.

Naming the Derived Table Columns using WITH

```
WITH TeraTom(AVGSAL) AS  
(SELECT AVG(salary)  
FROM Employee_Table)  
SELECT * FROM TeraTom ;
```

Alias Must be done
here

AVGSAL
46782.15

Answer
Set

When using the WITH Command, you must place your column's alias after the table name. In this example, it is where you see AVGSAL.

The Same Derived Query shown Three Different Ways

1

```
SELECT *
FROM (SELECT AVG(salary) FROM
Employee_Table) TeraTom(AVGSAL);
```

Alias CAN be done here

2

```
SELECT *
FROM (SELECT AVG(salary) as AVGSAL
FROM Employee_Table) TeraTom ;
```

or
here

3

```
WITH TeraTom(AVGSAL) AS
(SELECT AVG(salary)
FROM Employee_Table)
SELECT * FROM TeraTom ;
```

Alias must be done
here in WITH syntax

A Derived Table that Joins to an Existing Table

```
SELECT Dept_No, First_Name, Last_Name, AVGSAL  
FROM Employee_Table  
INNER JOIN  
(SELECT Dept_No, AVG(Salary) FROM Employee_Table  
GROUP BY Dept_No) as TeraTom (Depty, AVGSAL)  
ON Dept_No = Depty
```

Show all employees and their Average Salary per department!

<u>Dept No</u>	<u>First Name</u>	<u>Last Name</u>	<u>AVGSAL</u>
100	Mandee	Chambers	48850.00
10	Smythe	Richard	64300.00
200	Billy	Coffing	44944.44
200	John	Smith	44944.44
300	Lorraine	Larkins	40200.00
400	Cletus	Strickling	48333.33
400	William	Reilly	48333.33
400	Herbert	Harrison	48333.33

Answer Set

The first THREE columns in the Answer Set came from the Employee_Table.
AVGSAL came from the derived table named TeraTom.

Quiz - Answer the Questions

```
SELECT Dept_No, First_Name, Last_Name, AVGSAL  
FROM Employee_Table  
INNER JOIN  
(SELECT Dept_No, AVG(Salary)  
     FROM Employee_Table  
    GROUP BY Dept_No) as TeraTom (Dept, AVGSAL)  
ON Dept_No = Dept
```

What is the name of the derived table? _____

How many columns are in the derived table? _____

What is the name of the derived table columns? _____

Is there more than one row in the derived table? _____

What common keys join the Employee and Derived? _____

Why were the join keys named differently? _____

Answer to Quiz - Answer the Questions

```
SELECT Dept_No, First_Name, Last_Name, AVGSAL  
FROM Employee_Table  
INNER JOIN  
(SELECT Dept_No, AVG(Salary)  
FROM Employee_Table  
GROUP BY Dept_No) as TeraTom (Depty, AVGSAL)  
ON Dept_No = Depty
```

What is the name of the derived table? **TeraTom**

How many columns are in the derived table? **2**

What's the name of the derived columns? **Depty** and **AVGSAL**

Is there more than one row in the derived table? **Yes**

What keys join the tables? **Dept_No** and **Depty**

Why were the join keys named differently? If both were named **Dept_No** we would error, unless we full qualified.

Clever Tricks on Aliasing Columns in a Derived Table

1

```
SELECT Dept_No, First_Name, Last_Name, AVGSAL  
FROM Employee_Table  
INNER JOIN  
(SELECT Dept_No, AVG(Salary)  
FROM Employee_Table  
GROUP BY Dept_No) as TeraTom (Dept, AVGSAL)  
ON Dept_No = Dept
```

Alias Here

2

```
SELECT Dept_No, First_Name, Last_Name, AVGSAL  
FROM Employee_Table  
INNER JOIN  
(SELECT Dept_No as Depty, AVG(Salary) as AVGSAL  
FROM Employee_Table  
GROUP BY Dept_No) as TeraTom  
ON Dept_No = Depty
```

Alias Here

The Columns **Dept** and **AVGSAL** can be named with a different technique.

1

```
SELECT Dept_No, First_Name, Last_Name, AVGSAL
FROM Employee_Table
INNER JOIN
```

A red bracket is positioned above the word 'AVGSAL' in the first query. Two red arrows point from the word 'AVGSAL' down to the corresponding 'AVGSAL' in the subquery below.

Alias Here

```
(SELECT Dept_No as DeptY, AVG(Salary) as AVGSAL
     FROM Employee_Table
      GROUP BY Dept_No) as TeraTom
ON Dept_No = DeptY ;
```

2

```
SELECT E.Dept_No, First_Name, Last_Name, AVGSAL
FROM Employee_Table as E
INNER JOIN
```

A blue bracket is positioned above the letter 'E' in 'E.Dept_No'. A blue arrow points from the letter 'E' down to the 'E' in 'Employee_Table'.

Implicit
Alias

```
(SELECT Dept_No, AVG(Salary) as AVGSAL
     FROM Employee_Table
      GROUP BY Dept_No) as TeraTom
```

A red bracket is positioned above the word 'AVGSAL' in the second query. A red arrow points from the word 'AVGSAL' down to the 'AVGSAL' in the subquery below.

Alias Here

```
ON E.Dept_No = TeraTom.Dept_No ;
```

In example 2 why did we only give **AVGSAL** an alias? **Dept_No** defaulted **implicitly** to **Dept_No**. You must name the aggregate specifically, but normal columns can default to their current name. That's why **Dept_No** is fully qualified!

1

```
WITH TeraTom (Dept_No, AVGSAL) AS  
(SELECT Dept_No, AVG(Salary) FROM Employee_Table  
GROUP BY Dept_No)  
SELECT TeraTom.Dept_No, First_Name, Last_Name, AVGSAL  
FROM Employee_Table as E  
INNER JOIN  
TeraTom  
ON E.Dept_No = TeraTom.Dept_No ;
```

2

```
SELECT E.Dept_No, First_Name, Last_Name, AVGSAL  
FROM Employee_Table as E
```

```
INNER JOIN
```

```
(SELECT Dept_No, AVG(Salary) as AVGSAL  
FROM Employee_Table  
GROUP BY Dept_No) as TeraTom
```

```
ON E.Dept_No = TeraTom.Dept_No ;
```

Implicit
Alias

Alias Here

Example 1 uses the WITH command to create the Derived Table, but notice that the column aliases must be defined right after the With TeraTom statement.

A Derived Table lives only for the lifetime of a single query

BT ;

Begin Transaction

1

```
WITH T (Dept_No, AVGSAL) AS  
(SELECT Dept_No, AVG(Salary) FROM Employee_Table  
GROUP BY Dept_No)  
SELECT T.Dept_No, First_Name, Last_Name, AVGSAL  
FROM Employee_Table as E  
INNER JOIN  
T  
ON E.Dept_No = T.Dept_No ;
```

2

SELECT * FROM T ;

Error – Query Fails....
T does Not exist.

ET;

End Transaction

The second query fails in this Multi-Statement transaction because the derived table is only good for the life of a single query. T no longer exists!

An Example of Two Derived Tables in a Single Query

```
WITH T (Dept_No, AVGSAL) AS  
(SELECT Dept_No, AVG(Salary) FROM Employee_Table  
GROUP BY Dept_No)
```

```
SELECT T.Dept_No, First_Name, Last_Name,  
      AVGSAL, Counter
```

```
FROM Employee_Table as E
```

```
INNER JOIN
```

```
T
```

```
ON E.Dept_No = T.Dept_No
```

```
INNER JOIN
```

```
(SELECT Employee_No, SUM(1)  
OVER(PARTITION BY Dept_No  
ORDER BY Dept_No, Last_Name Rows Unbounded Preceding)  
FROM Employee_Table) as S (Employee_No, Counter)
```

```
ON E.Employee_No = S.Employee_No  
ORDER BY T.Dept_No;
```

TeraTom Coffing CEO	
Jane Stevens VP North	Ricardo Gonzales VP South
Hitesh Patel North Manager	Inquayee Mumba South Manager
North Analysts	South Analysts
Robert Pantelle	Betty Boston
Ming Zao	Kelly Roberts
Constantine Mikas	Brett Valens

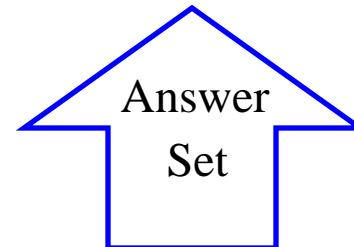
Emp	LastN	Pos_Name	Mgr	Depth
1	Coffing	CEO	?	0
20	Stevens	VP North	1	1
10	Gonzales	VP South	1	1
100	Patel	North Manager	10	2
200	Mumba	South Manager	20	2
5000	Pantelle	Analyst North	100	3
3000	Zao	Analyst North	100	3
1000	Mikas	Analyst North	100	3
4000	Roberts	Analyst South	200	3
6000	Boston	Analyst South	200	3
2000	Valens	Analyst South	200	3

WITH RECURSIVE Hierarchy_DT

```
(Emp, LastN, Pos_Name, Mgr, DEPTH) AS
(
SELECT Employee_No, Last_Name,
Position_Name, Mgr_Employee_No, 0
FROM Hierarchy_Table
WHERE Mgr_Employee_No IS NULL

UNION ALL

SELECT Employee_No, Last_Name,
Position_Name, Mgr_Employee_No, DEPTH+1
  from Hierarchy_DT , Hierarchy_Table
 WHERE Emp= Mgr_Employee_No
)
select Emp, LastN, Pos_Name, Mgr, DEPTH
  FROM  Hierarchy_DT
 order by DEPTH, Mgr ;
```



A RECURSIVE Derive Table always use the WITH Command and will loop through data and bring back the hierarchy from it.

The name of this RECURSIVE Derived Table is 'Hierarchy_DT'.

TeraTom Coffing CEO

Jane Stevens
VP North

Ricardo Gonzales
VP South

Hitesh Patel
North Manager

Inquayee Mumba
South Manager

North Analysts

Robert Pantelle
Ming Zao
Constantine Mikas

South Analysts

Betty Boston
Kelly Roberts
Brett Valens

<u>Emp</u>	<u>LastN</u>	<u>Pos_Name</u>	<u>Mgr</u>	<u>Depth</u>
1	Coffing	CEO	?	0
20	Stevens	VP North	1	1
10	Gonzales	VP South	1	1
100	Patel	North Manager	10	2
200	Mumba	South Manager	20	2
5000	Pantelle	Analyst North	100	3
3000	Zao	Analyst North	100	3
1000	Mikas	Analyst North	100	3
4000	Roberts	Analyst South	200	3
6000	Boston	Analyst South	200	3
2000	Valens	Analyst South	200	3

WITH RECURSIVE Hierarchy_DT

(Emp, LastN, Pos_Name, Mgr, DEPTH) AS

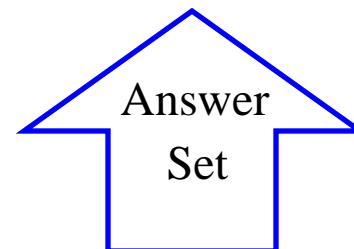
(

SELECT Employee_No, Last_Name,
Position_Name, Mgr_Employee_No, 0
FROM Hierarchy_Table
WHERE Mgr_Employee_No IS NULL

UNION ALL

SELECT Employee_No, Last_Name,
Position_Name, Mgr_Employee_No, DEPTH+1
from Hierarchy_DT , Hierarchy_Table
WHERE Emp= Mgr_Employee_No
)
select Emp, LastN, Pos_Name, Mgr, DEPTH
FROM Hierarchy_DT
order by DEPTH, Mgr;

It has five columns in the Recursive Derived Table **Hierarchy_DT**.



TeraTom Coffing CEO

Jane Stevens VP North	Ricardo Gonzales VP South
Hitesh Patel North Manager	Inquayee Mumba South Manager
North Analysts Robert Pantelle Ming Zao Constantine Mikas	South Analysts Betty Boston Kelly Roberts Brett Valens

Emp	LastN	Pos_Name	Mgr	Depth
1	Coffing	CEO	?	0
20	Stevens	VP North	1	1
10	Gonzales	VP South	1	1
100	Patel	North Manager	10	2
200	Mumba	South Manager	20	2
5000	Pantelle	Analyst North	100	3
3000	Zao	Analyst North	100	3
1000	Mikas	Analyst North	100	3
4000	Roberts	Analyst South	200	3
6000	Boston	Analyst South	200	3
2000	Valens	Analyst South	200	3

WITH RECURSIVE Hierarchy_DT
 (Emp, LastN, Pos_Name, Mgr, DEPTH) AS
 (
 SELECT Employee_No, Last_Name,
 Position_Name, Mgr_Employee_No, 0
 FROM Hierarchy_Table
 WHERE Mgr_Employee_No IS NULL
 UNION ALL
 SELECT Employee_No, Last_Name,
 Position_Name, Mgr_Employee_No, DEPTH+1
 from Hierarchy_DT , Hierarchy_Table
 WHERE Emp= Mgr_Employee_No
)
 select Emp, LastN, Pos_Name, Mgr, DEPTH
 FROM Hierarchy_DT
 order by DEPTH, Mgr;



TeraTom Coffing is the only employee in the Hierarchy_Table that doesn't report to a manager above them because he is the **CEO**.

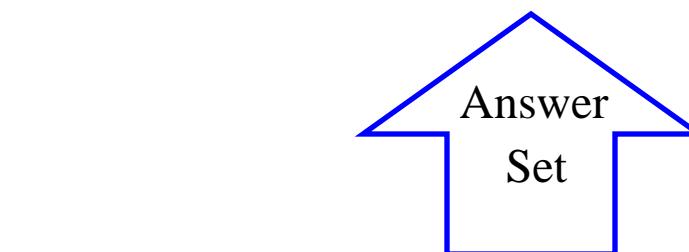
TeraTom Coffing is the only employee whose MGR is **NULL (?)**.

TeraTom Coffing CEO

Jane Stevens VP North	Ricardo Gonzales VP South
Hitesh Patel North Manager	Inquayee Mumba South Manager
North Analysts Robert Pantelle Ming Zao Constantine Mikas	South Analysts Betty Boston Kelly Roberts Brett Valens

<u>Emp</u>	<u>LastN</u>	<u>Pos_Name</u>	<u>Mgr</u>	<u>Depth</u>
1	Coffing	CEO	?	0
20	Stevens	VP North	1	1
10	Gonzales	VP South	1	1
100	Patel	North Manager	10	2
200	Mumba	South Manager	20	2
5000	Pantelle	Analyst North	100	3
3000	Zao	Analyst North	100	3
1000	Mikas	Analyst North	100	3
4000	Roberts	Analyst South	200	3
6000	Boston	Analyst South	200	3
2000	Valens	Analyst South	200	3

WITH RECURSIVE Hierarchy_DT
 (Emp, LastN, Pos_Name, Mgr, DEPTH) AS
 (
 SELECT Employee_No, Last_Name,
 Position_Name, Mgr_Employee_No, 0
 FROM Hierarchy_Table
 WHERE Mgr_Employee_No IS NULL
 UNION ALL
 SELECT Employee_No, Last_Name,
 Position_Name, Mgr_Employee_No, DEPTH+1
 from Hierarchy_DT , Hierarchy_Table
 WHERE Emp= Mgr_Employee_No
)
 select Emp, LastN, Pos_Name, Mgr, DEPTH
 FROM Hierarchy_DT
 order by DEPTH, Mgr;



DEPTH is the only Teradata Keyword in Recursive table Hierarchy_DT.

DEPTH provides insight at what level the row falls into the hierarchy.

TeraTom Coffing CEO

Jane Stevens
VP North

Ricardo Gonzales
VP South

Hitesh Patel
North Manager

Inquayee Mumba
South Manager

North Analysts

Robert Pantelle
Ming Zao
Constantine Mikas

South Analysts

Betty Boston
Kelly Roberts
Brett Valens

Emp	LastN	Pos_Name	Mgr	Depth
1	Coffing	CEO	?	0
20	Stevens	VP North	1	1
10	Gonzales	VP South	1	1
100	Patel	North Manager	10	2
200	Mumba	South Manager	20	2
5000	Pantelle	Analyst North	100	3
3000	Zao	Analyst North	100	3
1000	Mikas	Analyst North	100	3
4000	Roberts	Analyst South	200	3
6000	Boston	Analyst South	200	3
2000	Valens	Analyst South	200	3

WITH RECURSIVE Hierarchy_DT
(Emp, LastN, Pos_Name, Mgr, DEPTH) AS

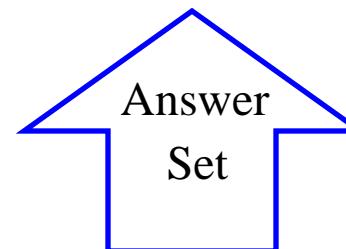
(
SELECT Employee_No, Last_Name,
Position_Name, Mgr_Employee_No, 0
FROM Hierarchy_Table
WHERE Mgr_Employee_No IS NULL

0

UNION ALL

SELECT Employee_No, Last_Name,
Position_Name, Mgr_Employee_No, DEPTH+1
from Hierarchy_DT , Hierarchy_Table
WHERE Emp= Mgr_Employee_No
)
select Emp, LastN, Pos_Name, Mgr, DEPTH
FROM Hierarchy_DT
order by DEPTH, Mgr;

When the BLUE SQL runs the
first time, 0 is the value in Depth.



TeraTom Coffing is the only row placed in the recursive derived table Hierarchy_DT as seen in Red above.

TeraTom Coffing CEO

Jane Stevens VP North	Ricardo Gonzales VP South
Hitesh Patel North Manager	Inquayee Mumba South Manager
North Analysts Robert Pantelle Ming Zao Constantine Mikas	South Analysts Betty Boston Kelly Roberts Brett Valens

Emp	LastN	Pos_Name	Mgr	Depth
1	Coffing	CEO	?	0
20	Stevens	VP North	1	1
10	Gonzales	VP South	1	1
100	Patel	North Manager	10	2
200	Mumba	South Manager	20	2
5000	Pantelle	Analyst North	100	3
3000	Zao	Analyst North	100	3
1000	Mikas	Analyst North	100	3
4000	Roberts	Analyst South	200	3
6000	Boston	Analyst South	200	3
2000	Valens	Analyst South	200	3

WITH RECURSIVE Hierarchy_DT
(Emp, LastN, Pos_Name, Mgr, DEPTH) AS
(
SELECT Employee_No, Last_Name,
Position_Name, Mgr_Employee_No, 0
FROM Hierarchy_Table
WHERE Mgr_Employee_No IS NULL

UNION ALL

SELECT Employee_No, Last_Name,
Position_Name, Mgr_Employee_No, DEPTH+1
from Hierarchy_DT , Hierarchy_Table
WHERE Emp= Mgr_Employee_No
)
select Emp, LastN, Pos_Name, Mgr, DEPTH
FROM Hierarchy_DT
order by DEPTH, Mgr;

0

Hierarchy_DT (Depth 0)

Emp	LastN	Pos_Name	Mgr	Depth
1	Coffing	CEO	?	0

When the highlighted SQL runs the first time, Coffing is the only row inserted inside Hierarchy_DT because it's the only one with a NULL value in Mgr.

TeraTom Coffing CEO

Jane Stevens
VP North

Ricardo Gonzales
VP South

Hitesh Patel
North Manager

Inquayee Mumba
South Manager

North Analysts

South Analysts

Robert Pantelle

Betty Boston

Ming Zao

Kelly Roberts

Constantine Mikas

Brett Valens

Emp	LastN	Pos_Name	Mgr	Depth
1	Coffing	CEO	?	0
20	Stevens	VP North	1	1
10	Gonzales	VP South	1	1
100	Patel	North Manager	10	2
200	Mumba	South Manager	20	2
5000	Pantelle	Analyst North	100	3
3000	Zao	Analyst North	100	3
1000	Mikas	Analyst North	100	3
4000	Roberts	Analyst South	200	3
6000	Boston	Analyst South	200	3
2000	Valens	Analyst South	200	3

WITH RECURSIVE Hierarchy_DT
(Emp, LastN, Pos_Name, Mgr, DEPTH) AS
(
SELECT Employee_No, Last_Name,
Position_Name, Mgr_Employee_No, 0
FROM Hierarchy_Table
WHERE Mgr_Employee_No IS NULL

UNION ALL

SELECT Employee_No, Last_Name,
Position_Name, Mgr_Employee_No, DEPTH+1
from Hierarchy_DT , Hierarchy_Table
WHERE Emp= Mgr_Employee_No
)

select Emp, LastN, Pos_Name, Mgr, DEPTH
FROM Hierarchy_DT
order by DEPTH, Mgr;

1

Hierarchy_DT (Depth 1)

Emp	LastN	Pos_Name	Mgr	Depth
1	Coffing	CEO	?	0
20	Stevens	VP North	1	1
10	Gonzales	VP South	1	1

When the highlighted SQL runs the UNION ALL, Stevens and Gonzales are the only two directly reporting to TeraTom.

TeraTom Coffing CEO

Jane Stevens VP North	Ricardo Gonzales VP South
Hitesh Patel North Manager	Inquayee Mumba South Manager
North Analysts Robert Pantelle Ming Zao Constantine Mikas	South Analysts Betty Boston Kelly Roberts Brett Valens

Emp	LastN	Pos_Name	Mgr	Depth
1	Coffing	CEO	?	0
20	Stevens	VP North	1	1
10	Gonzales	VP South	1	1
100	Patel	North Manager	10	2
200	Mumba	South Manager	20	2
5000	Pantelle	Analyst North	100	3
3000	Zao	Analyst North	100	3
1000	Mikas	Analyst North	100	3
4000	Roberts	Analyst South	200	3
6000	Boston	Analyst South	200	3
2000	Valens	Analyst South	200	3

WITH RECURSIVE Hierarchy_DT
 (Emp, LastN, Pos_Name, Mgr, DEPTH) AS
 (
 SELECT Employee_No, Last_Name,
 Position_Name, Mgr_Employee_No, 0
 FROM Hierarchy_Table
 WHERE Mgr_Employee_No IS NULL
 UNION ALL
 SELECT Employee_No, Last_Name,
 Position_Name, Mgr_Employee_No, DEPTH+1
 from Hierarchy_DT , Hierarchy_Table
 WHERE Emp= Mgr_Employee_No
)
 select Emp, LastN, Pos_Name, Mgr, DEPTH
 FROM Hierarchy_DT
 order by DEPTH, Mgr;

Hierarchy_DT (Depth 1)

Emp	LastN	Pos_Name	Mgr	Depth
1	Coffing	CEO	?	0
20	Stevens	VP North	1	1
10	Gonzales	VP South	1	1

The original Depth was set to 0, but in the First run of the Union All, the DEPTH is set to 1.

1

TeraTom Coffing CEO

Jane Stevens VP North	Ricardo Gonzales VP South
Hitesh Patel North Manager	Inquayee Mumba South Manager
North Analysts Robert Pantelle Ming Zao Constantine Mikas	South Analysts Betty Boston Kelly Roberts Brett Valens

Emp	LastN	Pos_Name	Mgr	Depth
1	Coffing	CEO	?	0
20	Stevens	VP North	1	1
10	Gonzales	VP South	1	1
100	Patel	North Manager	10	2
200	Mumba	South Manager	20	2
5000	Pantelle	Analyst North	100	3
3000	Zao	Analyst North	100	3
1000	Mikas	Analyst North	100	3
4000	Roberts	Analyst South	200	3
6000	Boston	Analyst South	200	3
2000	Valens	Analyst South	200	3

WITH RECURSIVE Hierarchy_DT
 (Emp, LastN, Pos_Name, Mgr, DEPTH) AS
 (
 SELECT Employee_No, Last_Name,
 Position_Name, Mgr_Employee_No, 0
 FROM Hierarchy_Table
 WHERE Mgr_Employee_No IS NULL
 UNION ALL
 SELECT Employee_No, Last_Name,
 Position_Name, Mgr_Employee_No, DEPTH+1
 from Hierarchy_DT , Hierarchy_Table
 WHERE Emp= Mgr_Employee_No
)

select Emp, LastN, Pos_Name, Mgr, DEPTH
 FROM Hierarchy_DT
 order by DEPTH, Mgr;

Emp	LastN	Pos_Name	Mgr	Depth
1	Coffing	CEO	?	0
20	Stevens	VP North	1	1
10	Gonzales	VP South	1	1
100	Patel	North Manager	10	2
200	Mumba	South Manager	20	2

When the UNION ALL loops and runs a second time, Patel and Mumba report directly to Stevens and Gonzales.

2

TeraTom Coffing CEO	
Jane Stevens VP North	Ricardo Gonzales VP South
Hitesh Patel North Manager	Inquayee Mumba South Manager
North Analysts	South Analysts
Robert Pantelle	Betty Boston
Ming Zao	Kelly Roberts
Constantine Mikas	Brett Valens

Emp	LastN	Pos_Name	Mgr	Depth
1	Coffing	CEO	?	0
20	Stevens	VP North	1	1
10	Gonzales	VP South	1	1
100	Patel	North Manager	10	2
200	Mumba	South Manager	20	2
5000	Pantelle	Analyst North	100	3
3000	Zao	Analyst North	100	3
1000	Mikas	Analyst North	100	3
4000	Roberts	Analyst South	200	3
6000	Boston	Analyst South	200	3
2000	Valens	Analyst South	200	3

WITH RECURSIVE Hierarchy_DT
 (Emp, LastN, Pos_Name, Mgr, DEPTH) AS
 (
 SELECT Employee_No, Last_Name,
 Position_Name, Mgr_Employee_No, 0
 FROM Hierarchy_Table
 WHERE Mgr_Employee_No IS NULL
 UNION ALL
 SELECT Employee_No, Last_Name,
 Position_Name, Mgr_Employee_No, DEPTH+1
 from Hierarchy_DT , Hierarchy_Table
 WHERE Emp= Mgr_Employee_No
)
 select Emp, LastN, Pos_Name, Mgr, DEPTH
 FROM Hierarchy_DT
 order by DEPTH, Mgr;

When the UNION ALL loops and runs a third time, SIX people will end up reporting to Patel and Mumba.



TeraTom Coffing CEO

Jane Stevens VP North	Ricardo Gonzales VP South
Hitesh Patel North Manager	Inquayee Mumba South Manager
North Analysts Robert Pantelle Ming Zao Constantine Mikas	South Analysts Betty Boston Kelly Roberts Brett Valens

Emp	LastN	Pos_Name	Mgr	Depth
1	Coffing	CEO	?	0
20	Stevens	VP North	1	1
10	Gonzales	VP South	1	1
100	Patel	North Manager	10	2
200	Mumba	South Manager	20	2
5000	Pantelle	Analyst North	100	3
3000	Zao	Analyst North	100	3
1000	Mikas	Analyst North	100	3
4000	Roberts	Analyst South	200	3
6000	Boston	Analyst South	200	3
2000	Valens	Analyst South	200	3

WITH RECURSIVE Hierarchy_DT
 (Emp, LastN, Pos_Name, Mgr, DEPTH) AS
 (
 SELECT Employee_No, Last_Name,
 Position_Name, Mgr_Employee_No, 0
 FROM Hierarchy_Table
 WHERE Mgr_Employee_No IS NULL
 UNION ALL
 SELECT Employee_No, Last_Name,
 Position_Name, Mgr_Employee_No, DEPTH+1
 from Hierarchy_DT , Hierarchy_Table
 WHERE Emp= Mgr_Employee_No
)
 select Emp, LastN, Pos_Name, Mgr, DEPTH
 FROM Hierarchy_DT
 order by DEPTH, Mgr;

When the UNION ALL loops and runs a fourth time, 0 people are added to Hierarchy_DT.

When NO row is added during a loop then the query knows it is time to leave that loop.

4

TeraTom Coffing CEO

Jane Stevens
VP North

Ricardo Gonzales
VP South

Hitesh Patel
North Manager

Inquayee Mumba
South Manager

North Analysts

Robert Pantelle
Ming Zao
Constantine Mikas

South Analysts

Betty Boston
Kelly Roberts
Brett Valens

Emp	LastN	Pos_Name	Mgr	Depth
1	Coffing	CEO	?	0
20	Stevens	VP North	1	1
10	Gonzales	VP South	1	1
100	Patel	North Manager	10	2
200	Mumba	South Manager	20	2
5000	Pantelle	Analyst North	100	3
3000	Zao	Analyst North	100	3
1000	Mikas	Analyst North	100	3
4000	Roberts	Analyst South	200	3
6000	Boston	Analyst South	200	3
2000	Valens	Analyst South	200	3

WITH RECURSIVE Hierarchy_DT
(Emp, LastN, Pos_Name, Mgr, DEPTH) AS
(
SELECT Employee_No, Last_Name,
Position_Name, Mgr_Employee_No, 0
FROM Hierarchy_Table
WHERE Mgr_Employee_No IS NULL

UNION ALL

SELECT Employee_No, Last_Name,
Position_Name, Mgr_Employee_No, DEPTH+1
from Hierarchy_DT , Hierarchy_Table
WHERE Emp= Mgr_Employee_No
)

select Emp, LastN, Pos_Name, Mgr, DEPTH
FROM Hierarchy_DT
order by DEPTH, Mgr;

No Rows added in
Depth 4

When 0 rows are added to the derived table
Hierarchy_DT, the loop will end.

4

TeraTom Coffing CEO

Jane Stevens VP North	Ricardo Gonzales VP South
Hitesh Patel North Manager	Inquayee Mumba South Manager
North Analysts Robert Pantelle Ming Zao Constantine Mikas	South Analysts Betty Boston Kelly Roberts Brett Valens

Emp	LastN	Pos_Name	Mgr	Depth
1	Coffing	CEO	?	0
20	Stevens	VP North	1	1
10	Gonzales	VP South	1	1
100	Patel	North Manager	10	2
200	Mumba	South Manager	20	2
5000	Pantelle	Analyst North	100	3
3000	Zao	Analyst North	100	3
1000	Mikas	Analyst North	100	3
4000	Roberts	Analyst South	200	3
6000	Boston	Analyst South	200	3
2000	Valens	Analyst South	200	3

```

WITH RECURSIVE Hierarchy_DT
(Emp, LastN, Pos_Name, Mgr, DEPTH) AS
(
SELECT Employee_No, Last_Name,
Position_Name, Mgr_Employee_No, 0
FROM Hierarchy_Table
WHERE Mgr_Employee_No IS NULL

UNION ALL

SELECT Employee_No, Last_Name,
Position_Name, Mgr_Employee_No, DEPTH+1
  from Hierarchy_DT , Hierarchy_Table
 WHERE Emp= Mgr_Employee_No
)
select Emp, LastN, Pos_Name, Mgr, DEPTH
  FROM  Hierarchy_DT
 order by DEPTH, Mgr;

```



When the loop ends in Depth 4, the highlighted SQL will run once to build the final answer set above.

Creating a Volatile Table

```
CREATE VOLATILE TABLE Dept_Agg_Vol , NO LOG  
    ( Dept_no          Integer  
      ,Sum_Salary     Decimal(10,2)  
    )  
ON COMMIT PRESERVE ROWS ;
```

NO Log is the default and it means don't use the Transient Journal, which gives you Rollback capabilities and better data integrity, but since this is a Volatile Table who cares. **No Log** is faster when doing Maintenance.

ON COMMIT PRESERVE ROWS is NOT the default. You must use these Keywords if you want your data to stay in the Volatile Table after you populate it, otherwise after the load transaction the data is deleted. That is referred to as **ON COMMIT DELETE ROWS**.

This statement creates a Volatile Table!

You Populate a Volatile Table with an INSERT/SELECT

1

```
CREATE VOLATILE TABLE Dept_Agg_Vol , NO LOG  
    ( Dept_no          Integer  
      ,Sum_Salary     Decimal(10,2)  
    )  
ON COMMIT PRESERVE ROWS ;
```

2

```
INSERT INTO Dept_Agg_Vol  
SELECT  Dept_no  
      ,SUM(Salary)  
FROM Employee_Table  
GROUP BY Dept_no ;
```

- 1) A USER Creates a Volatile Table and then 2) populates the Volatile Table with an INSERT/SELECT Statement.

The Three Steps to Use a Volatile Table

1

```
CREATE VOLATILE TABLE Dept_Agg_Vol , NO LOG  
        ( Dept_no          Integer  
          ,Sum_Salary      Decimal(10,2)  
        )  
ON COMMIT PRESERVE ROWS ;
```

2

```
INSERT INTO Dept_Agg_Vol  
SELECT  Dept_no  
       ,SUM(Salary)  
FROM Employee_Table  
GROUP BY Dept_no ;
```

3

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

Only **you** can see this data because your **session number** is associated with this **Volatile Table**. You can't even see this table if you login and query it from another session!

- 1) A USER Creates a Volatile Table and then 2) populates the Volatile Table with an INSERT/SELECT Statement, and then 3) Query it until you Logoff.

The HELP Volatile Table Command Shows your Volatiles

```
CREATE VOLATILE TABLE MyVolTbl
```

```
(  
    Dept_no          Integer  
    ,Sum_Salary     Decimal(10,2)  
    ,Avg_Salary     Decimal(7,2)  
    ,Max_Salary     Decimal(7,2)  
    ,Min_Salary     Decimal(7,2)  
    ,Cnt_Salary     Integer )
```

```
ON COMMIT PRESERVE ROWS ;
```

1

```
INSERT INTO MyVolTbl
```

```
SELECT   Dept_no  
        ,SUM(Salary)  
        ,AVG(Salary)  
        ,MAX(Salary)  
        ,MIN(Salary)  
        ,COUNT(Salary)  
FROM Employee_Table  
GROUP BY Dept_no ;
```

2

3

```
HELP Volatile Table ;
```

SessionID	TableName	Table Id	Protection	CreatorName	CommitOption	TransactionLog
1010	MyVolTbl	10C0C4	N	TeraTom	P	N

Volatile Tables are stored in the Cache Memory of the PE.

CREATING A Global Temporary Table

```
CREATE Global Temporary TABLE Dept_Agg_GLO  
  ( Dept_no      Integer  
    ,Sum_Salary   Decimal(10,2)  
  )  
ON COMMIT PRESERVE ROWS ;
```

ON COMMIT PRESERVE ROWS is NOT the default. You must use these Keywords if you want your data to stay in the Volatile Table after you populate it otherwise after the load transaction the data is deleted. That is referred to as ON COMMIT DELETE ROWS.

The Table Definition stays Permanently. When a user logs off the data Inside the Global Temporary Table is deleted, but the definition stays around ready to be populated again.

This syntax creates a Global Temporary Table, which is stored in the Data Dictionary of Teradata. A Global Temporary Table survives a Teradata Restart.

Populating A Global Temporary Table with INSERT/SELECT

1

```
CREATE Global Temporary TABLE Dept_Agg_GLO
  ( Dept_no          Integer
    ,Sum_Salary      Decimal(10,2)
  )
ON COMMIT PRESERVE ROWS ;
```

2

```
INSERT INTO Dept_Agg_GLO
SELECT  Dept_no
       ,SUM(Salary)
FROM Employee_Table
GROUP BY Dept_No ;
```

- 1) A USER Creates a Global Temporary Table **one time** and then 2) populates the Global Temporary Table with an INSERT/SELECT Statement.

The Three Steps to using a Global Temporary Table

1

```
CREATE Global Temporary TABLE Dept_Agg_GLO  
    ( Dept_no      Integer  
      ,Sum_Salary   Decimal(10,2)  
    )  
ON COMMIT PRESERVE ROWS ;
```

2

```
INSERT INTO Dept_Agg_GLO  
SELECT Dept_no  
      ,SUM(Salary)  
FROM Employee_Table  
GROUP BY Dept_no ;
```

3

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

Any user who has **TEMP** Space can Populate (Materialize) this table and many users can do so simultaneously, but nobody has access to anyone else's data. Separate copies are made per user.

- 1) A USER Creates a Volatile Table and then 2) populates the Volatile Table with an **INSERT/SELECT** Statement, and then 3) Query it until you Logoff. All data is deleted when a user logs off, but the **table definition stays forever**, unless dropped.

Chapter 24

Sub-query Functions

“A little man often casts a long shadow.”

- Italian Proverb

Table of Contents Chapter 24 - Sub-query Functions

- [An IN List is much like a Subquery](#)
- [An IN List Never has Duplicates – Just like a Subquery](#)
- [An IN List Ignores Duplicates](#)
- [The Subquery](#)
- [How a Basic Subquery Works](#)
- [The Final Answer Set from the Subquery](#)
- [Quiz- Answer the Difficult Question](#)
- [Answer to Quiz- Answer the Difficult Question](#)
- [Should you use a Subquery of a Join?](#)
- [Quiz- Write the Subquery](#)
- [Answer to Quiz- Write the Subquery](#)
- [Quiz- Write the More Difficult Subquery](#)
- [Answer to Quiz- Write the More Difficult Subquery](#)
- [Quiz- Write the Subquery with an Aggregate](#)
- [Answer to Quiz- Write the Subquery with an Aggregate](#)
- [Quiz- Write the Correlated Subquery](#)
- [Answer to Quiz- Write the Correlated Subquery](#)
- [The Basics of a Correlated Subquery](#)
- [The Top Query always runs first in a Correlated Subquery](#)
- [The Bottom Query runs Last in a Correlated Subquery](#)
- [Quiz- Who is coming back in the Final Answer Set?](#)

Continued on next page

Table of Contents Chapter 24 - Sub-query Functions Continued

- [Correlated Subquery Example vs. a Join with a Derived Table](#)
- [Correlated Subquery that Finds Duplicates](#)
- [Quiz- Write the NOT Subquery](#)
- [Answer to Quiz- Write the NOT Subquery](#)
- [Quiz- Write the Subquery using a WHERE Clause](#)
- [Answer to Quiz- Write the Subquery using a WHERE Clause](#)
- [Quiz- Write the Subquery with Two Parameters](#)
- [Answer to Quiz- Write the Subquery with Two Parameters](#)
- [How the Double Parameter Subquery Works](#)
- [More on how the Double Parameter Subquery Works](#)
- [Quiz – Write the Triple Subquery](#)
- [Answer to Quiz – Write the Triple Subquery](#)
- [Quiz – How many rows return on a NOT IN with a NULL?](#)
- [Answer – How many rows return on a NOT IN with a NULL?](#)
- [How to handle a NOT IN with Potential NULL Values](#)
- [IN is equivalent to =ANY](#)
- [Using a Correlated Exists](#)
- [How a Correlated Exists matches up](#)
- [The Correlated NOT Exists](#)
- [The Correlated NOT Exists Answer Set](#)
- [Quiz – How many rows come back from this NOT Exists?](#)
- [Answer – How many rows come back from this NOT Exists?](#)

An IN List is much like a Subquery

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	Lorraine	40200.00
1256349	400	Harrison	Herbert	54500.00
2341218	400	Reilly	William	36000.00
1121334	400	Strickling	Cletus	54500.00

```
SELECT *
FROM Employee_Table
WHERE Dept_No IN (100, 200)
```

Employee_No	Dept_No	Last_Name	First_Name	Salary
1232578	100	Chambers	Mandee	48850.00
1324657	200	Coffing	Billy	41888.88
1333454	200	Smith	John	48000.00

Answer Set

This query is very simple and easy to understand. It uses an IN List to find all Employees who are in Dept_No 100 or Dept_No 200.

An IN List Never has Duplicates – Just like a Subquery

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	Lorraine	40200.00	
1256349	400	Harrison	Herbert	54500.00	
2341218	400	Reilly	William	36000.00	
1121334	400	Strickling	Cletus	54500.00	

```
SELECT *
FROM Employee_Table
WHERE Dept_No IN (100, 100,200, 200);
```



What is going on with this IN List? Why in the world are their duplicates in there? Will this query even work? What will the result set look like? Turn the page!

An IN List Ignores Duplicates

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

```
SELECT *
FROM Employee_Table
WHERE Dept_No IN (100, 100, 200, 200);
```

Employee_No	Dept_No	Last_Name	First_Name	Salary
1232578	100	Chambers	Mandee	48850.00
1324657	200	Coffing	Billy	41888.88
1333454	200	Smith	John	48000.00

Answer Set

Duplicate values are ignored here. We got the same rows back as before and it is as if the system ignored the duplicate values in the IN List. That is exactly what happened.

The Subquery

Employee_Table					Department_Table	
Employee_No	Dept_No	Last_Name	First_Name	Salary	Dept_No	Department_Name
1232578	100	Chambers	Mandee	48850.00	100	Marketing
1256349	400	Harrison	Herbert	54500.00	200	Research and Dev
2341218	400	Reilly	William	36000.00	300	Sales
2312225	300	Larkins	Lorraine	40200.00	400	Customer Support
2000000	?	Jones	Squiggy	32800.50	500	Human Resources
1000234	10	Smythe	Richard	32800.00		
1121334	400	Strickling	Cletus	54500.00		
1324657	200	Coffing	Billy	41888.88		
1333454	200	Smith	John	48000.00		

There is a **Top Query**
and a **Bottom Query!**

```
SELECT *
FROM Employee_Table
WHERE Dept_No IN (
    SELECT Dept_No
    FROM Department_Table);
```

Which Query
Runs First?

The query above is a Subquery, which means there are multiple queries in the same SQL. The bottom query runs first and its purpose in life is to build a distinct list of values that it passes to the top query. The top query then returns the result set. This query solves the problem: Show all Employees in Valid Departments!

How a Basic Subquery Works

Employee_Table					Department_Table	
Employee_No	Dept_No	Last_Name	First_Name	Salary	Dept_No	Department_Name
1232578	100	Chambers	Mandee	48850.00	100	Marketing
1256349	400	Harrison	Herbert	54500.00	200	Research and Dev
2341218	400	Reilly	William	36000.00	300	Sales
2312225	300	Larkins	Lorraine	40200.00	400	Customer Support
2000000	?	Jones	Squiggy	32800.50	500	Human Resources
1000234	10	Smythe	Richard	32800.00		
1121334	400	Strickling	Cletus	54500.00		
1324657	200	Coffing	Billy	41888.88		
1333454	200	Smith	John	48000.00		

```
SELECT      *
FROM        Employee_Table
WHERE       Dept_No IN (
    SELECT    Dept_No
    FROM      Department_Table);
```

Bottom Query
Runs First

100
200
300
400
500

Top Query
Runs Next

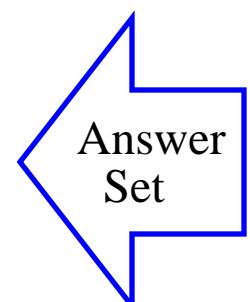
```
SELECT *
FROM Employee_Table
WHERE Dept_No IN (100, 200, 300, 400, 500);
```

The Final Answer Set from the Subquery

Employee_Table					Department_Table	
Employee_No	Dept_No	Last_Name	First_Name	Salary	Dept_No	Department_Name
1232578	100	Chambers	Mandee	48850.00	100	Marketing
1256349	400	Harrison	Herbert	54500.00	200	Research and Dev
2341218	400	Reilly	William	36000.00	300	Sales
2312225	300	Larkins	Lorraine	40200.00	400	Customer Support
2000000	?	Jones	Squiggy	32800.50	500	Human Resources
1000234	10	Smythe	Richard	32800.00		
1121334	400	Strickling	Cletus	54500.00		
1324657	200	Coffing	Billy	41888.88		
1333454	200	Smith	John	48000.00		

```
SELECT * FROM Employee_Table  
WHERE Dept_No IN (  
    SELECT Dept_No FROM Department_Table);
```

Employee_No	Dept_No	Last_Name	First_Name	Salary
1232578	100	Chambers	Mandee	48850.00
1324657	200	Coffing	Billy	41888.88
1333454	200	Smith	John	48000.00
2312225	300	Larkins	Lorraine	40200.00
1256349	400	Harrison	Herbert	54500.00
2341218	400	Reilly	William	36000.00
1121334	400	Strickling	Cletus	54500.00



Answer
Set

Quiz- Answer the Difficult Question

Employee_Table					Department_Table	
Employee_No	Dept_No	Last_Name	First_Name	Salary	Dept_No	Department_Name
1232578	100	Chambers	Mandee	48850.00	100	Marketing
1256349	400	Harrison	Herbert	54500.00	200	Research and Dev
2341218	400	Reilly	William	36000.00	300	Sales
2312225	300	Larkins	Lorraine	40200.00	400	Customer Support
2000000	?	Jones	Squiggy	32800.50	500	Human Resources
1000234	10	Smythe	Richard	32800.00		
1121334	400	Strickling	Cletus	54500.00		
1324657	200	Coffing	Billy	41888.88		
1333454	200	Smith	John	48000.00		

How are **Subqueries** similar to **Joins** between two tables?

A great question was asked above. Do you know the key to answering? Turn the page!

Answer to Quiz- Answer the Difficult Question

Employee_Table					Department_Table	
Employee_No	Dept_No	Last_Name	First_Name	Salary	Dept_No	Department_Name
1232578	100	Chambers	Mandee	48850.00	100	Marketing
1256349	400	Harrison	Herbert	54500.00	200	Research and Dev
2341218	400	Reilly	William	36000.00	300	Sales
2312225	300	Larkins	Lorraine	40200.00	400	Customer Support
2000000	?	Jones	Squiggy	32800.50	500	Human Resources
1000234	10	Smythe	Richard	32800.00		
1121334	400	Strickling	Cletus	54500.00		
1324657	200	Coffing	Billy	41888.88		
1333454	200	Smith	John	48000.00		

How are Subqueries similar to Joins between two tables?

A Subquery between two tables or a Join between two tables will each need a common key that represents the relationship.

Just like Dept_No and Dept_No!

A Subquery will use a common key linking the two tables together very similar to a join! When subquerying between two tables look for the common link between the two tables. Most of the time they both have a column with the same name, but not always.

Should you use a Subquery or a Join?

Employee_Table					Department_Table	
Employee_No	Dept_No	Last_Name	First_Name	Salary	Dept_No	Department_Name
1232578	100	Chambers	Mandee	48850.00	100	Marketing
1256349	400	Harrison	Herbert	54500.00	200	Research and Dev
2341218	400	Reilly	William	36000.00	300	Sales
2312225	300	Larkins	Lorraine	40200.00	400	Customer Support
2000000	?	Jones	Squiggy	32800.50	500	Human Resources
1000234	10	Smythe	Richard	32800.00		
1121334	400	Strickling	Cletus	54500.00		
1324657	200	Coffing	Billy	41888.88		
1333454	200	Smith	John	48000.00		

When do I **Subquery**?

```
SELECT *
FROM Employee_Table
WHERE Dept_No IN (
    SELECT Dept_No
    FROM Department_Table);
```

When do I perform a **Join**?

```
SELECT E.*
FROM Employee_Table as E
Inner Join
    Department_Table as D
ON E.Dept_No = D.Dept_No;
```

Both queries above return the same data. If you only want to see a report where the final result set has only columns from one table try a Subquery. Obviously, if you need columns on the report where the final result set has columns from both tables you have to do a Join.

Quiz- Write the Subquery

Customer_Table		Order_Table		
Customer_Number	Customer_Name	Order_Number	Customer_Number	Order_Total
11111111	Billy's Best Choice	123456	11111111	12347.53
31313131	Acme Products	123512	11111111	8005.91
31323134	ACE Consulting	123552	31323134	5111.47
57896883	XYZ Plumbing	123585	87323456	15231.62
87323456	Databases N-U	123777	57896883	23454.84

Write the Subquery

Select all columns in the Customer_Table if the customer has placed an order!

Here is your opportunity to show how smart you are. Write a Subquery that will bring back everything from the Customer_Table if the customer has placed an order in the Order_Table. Good luck! Advice: Look for the common key among both tables!

Answer to Quiz- Write the Subquery

Customer_Table		Order_Table		
Customer_Number	Customer_Name	Order_Number	Customer_Number	Order_Total
11111111	Billy's Best Choice	123456	11111111	12347.53
31313131	Acme Products	123512	11111111	8005.91
31323134	ACE Consulting	123552	31323134	5111.47
57896883	XYZ Plumbing	123585	87323456	15231.62
87323456	Databases N-U	123777	57896883	23454.84

Write the Subquery

Select all columns in the Customer_Table if the customer has placed an order!

```
SELECT *
FROM Customer_Table
WHERE Customer_Number IN (
    SELECT Customer_Number
    FROM Order_Table);
```

Customer_Number	Customer_Name
31323134	ACE Consulting
57896883	XYZ Plumbing
11111111	Billy's Best Choice
87323456	Databases N-U

The common key among both tables is Customer_Number. The bottom query runs first and delivers a **distinct** list of Customer_Numbers, which the top query uses in the IN List!

Quiz- Write the More Difficult Subquery

Customer_Table		Order_Table		
Customer_Number	Customer_Name	Order_Number	Customer_Number	Order_Total
11111111	Billy's Best Choice	123456	11111111	12347.53
31313131	Acme Products	123512	11111111	8005.91
31323134	ACE Consulting	123552	31323134	5111.47
57896883	XYZ Plumbing	123585	87323456	15231.62
87323456	Databases N-U	123777	57896883	23454.84

Write the Subquery

Select all columns in the Customer_Table if the customer has placed an order over \$10,000.00 Dollars!

Here is your opportunity to show how smart you are. Write a Subquery that will bring back everything from the Customer_Table if the customer has placed an order in the Order_Table that is greater than \$10,000.00.

Answer to Quiz- Write the More Difficult Subquery

Customer_Table

<u>Customer_Number</u>	<u>Customer_Name</u>
11111111	Billy's Best Choice
31313131	Acme Products
31323134	ACE Consulting
57896883	XYZ Plumbing
87323456	Databases N-U

Order_Table

<u>Order_Number</u>	<u>Customer_Number</u>	<u>Order_Total</u>
123456	11111111	12347.53
123512	11111111	8005.91
123552	31323134	5111.47
123585	87323456	15231.62
123777	57896883	23454.84

Write the Subquery

Select all columns in the Customer_Table if the customer has placed an order over \$10,000.00 Dollars!

```
SELECT *
FROM Customer_Table
WHERE Customer_Number IN (
    SELECT Customer_Number
    FROM Order_Table
    WHERE Order_Total > 10000.00);
```

<u>Customer_Number</u>	<u>Customer_Name</u>
11111111	Billy's Best Choice
57896883	XYZ Plumbing
87323456	Databases N-U

Here is your answer!

Quiz- Write the Subquery with an Aggregate

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	Lorraine	40200.00
1256349	400	Harrison	Herbert	54500.00
2341218	400	Reilly	William	36000.00
1121334	400	Strickling	Cletus	54500.00

Write the Subquery

Select **all** columns in the **Employee_Table** if the employee makes a greater **Salary** than the **AVERAGE** Salary.

Another opportunity knocking! Would someone please answer the query door!

Answer to Quiz- Write the Subquery with an Aggregate

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	Lorraine	40200.00
1256349	400	Harrison	Herbert	54500.00
2341218	400	Reilly	William	36000.00
1121334	400	Strickling	Cletus	54500.00

Write the Subquery

Select **all** columns in the **Employee_Table** if the employee makes a greater **Salary** than the **AVERAGE** Salary.

```
SELECT * FROM Employee_Table  
WHERE    Salary > (  
          SELECT AVG(Salary)  
          FROM    Employee_Table) ;
```

Quiz- Write the Correlated Subquery

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	Lorraine	40200.00
1256349	400	Harrison	Herbert	54500.00
2341218	400	Reilly	William	36000.00
1121334	400	Strickling	Cletus	54500.00

Write the **Correlated Subquery**

Select **all** columns in the **Employee_Table** if the employee makes a greater **Salary** than the **AVERAGE** Salary
(Within their own Department).

Another opportunity knocking! This is a tough one and only the best get this written right.

Answer to Quiz- Write the Correlated Subquery

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	Lorraine	40200.00
1256349	400	Harrison	Herbert	54500.00
2341218	400	Reilly	William	36000.00
1121334	400	Strickling	Cletus	54500.00

Write the Correlated Subquery

Select **all** columns in the Employee_Table if the employee makes a greater Salary than the **AVERAGE** Salary (Within their own Department).

```
SELECT * FROM Employee_Table as EE  
WHERE     Salary > (  
        SELECT AVG(Salary)  
        FROM Employee_Table as EEEE  
        WHERE EE.Dept_No = EEEE.Dept_No) ;
```

The Basics of a Correlated Subquery

The Top Query is Co-Related (Correlated) with the Bottom Query.

The same table is used twice, but given a different alias both times.

The bottom WHERE clause co-relates Dept_No from Top and Bottom.

```
SELECT *
FROM Employee_Table as EE
WHERE    Salary > (
    SELECT  AVG(Salary)
    FROM    Employee_Table as EEEE
    WHERE   EE.Dept_No = EEEE.Dept_No);
```

Does the **Top** or **Bottom** Query run first?

The Top Query always runs first in a Correlated Subquery

```
SELECT *
FROM Employee_Table as EE
WHERE Salary > (
    SELECT AVG(Salary)
    FROM Employee_Table as EEEE
    WHERE EE.Dept_No = EEEE.Dept_No);
```

The Top Query always runs **first** in a Correlated Subquery?

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	Lorraine	40200.00
1256349	400	Harrison	Herbert	54500.00
2341218	400	Reilly	William	36000.00
1121334	400	Strickling	Cletus	54500.00

Results only
after the TOP
Query has run.

This is **NOT**
the Final
ANSWER Set.

The Bottom Query runs Last in a Correlated Subquery

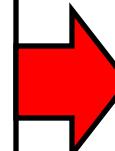
```
SELECT * FROM Employee_Table as EE  
WHERE     Salary > (  
    SELECT AVG(Salary)  
    FROM Employee_Table as EEEE  
    WHERE EE.Dept_No = EEEE.Dept_No);
```

The Top Query always runs 1st in a Correlated Subquery?



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	Lorraine	40200.00
1256349	400	Harrison	Herbert	54500.00
2341218	400	Reilly	William	36000.00
1121334	400	Strickling	Cletus	54500.00

Dept_No	AVG(Salary)
?	32800.50
10	32800.00
100	48850.00
200	44944.44
300	40200.00
400	48333.33

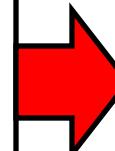


The Bottom Query run 2nd to get the Average Salary once for each distinct Dept_No!

Quiz- Who is coming back in the Final Answer Set?

```
SELECT * FROM Employee_Table as EE  
WHERE     Salary > (  
    SELECT AVG(Salary)  
    FROM Employee_Table as EEEE  
    WHERE EE.Dept_No = EEEE.Dept_No);
```

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	Lorraine	40200.00
1256349	400	Harrison	Herbert	54500.00
2341218	400	Reilly	William	36000.00
1121334	400	Strickling	Cletus	54500.00



Dept_No	AVG(Salary)
?	32800.50
10	32800.00
100	48850.00
200	44944.44
300	40200.00
400	48333.33

Which Employees will be in the Final Answer Set?

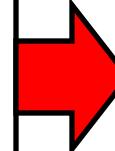
Look at the results from the TOP Query on Left and then look at how the Bottom Query is run once per Dept_No and figure out in your head who is coming back.

```

SELECT * FROM Employee_Table as EE
WHERE      Salary > (
    SELECT  AVG(Salary)
    FROM    Employee_Table as EEEE
    WHERE   EE.Dept_No = EEEE.Dept_No) ;

```

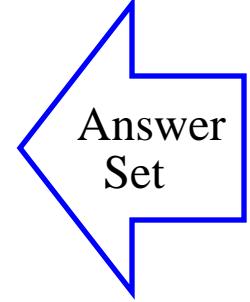
<u>Employee_No</u>	<u>Dept_No</u>	<u>Last_Name</u>	<u>First_Name</u>	<u>Salary</u>
2000000	?	Jones	Squiggy	32800.50
1000234	10	Smythe	Richard	32800.00
1232578	100	Chambers	Mandee	48850.00
1324657	200	Coffing	Billy	41888.88
1333454	200	Smith	John	48000.00
2312225	300	Larkins	Lorraine	40200.00
1256349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



<u>Dept_No</u>	<u>AVG(Salary)</u>
?	32800.50
10	32800.00
100	48850.00
200	44944.44
300	40200.00
400	48333.33

Which Employees will be in the Final Answer Set?

<u>Employee_No</u>	<u>Dept_No</u>	<u>Last_Name</u>	<u>First_Name</u>	<u>Salary</u>
1333454	200	Smith	John	48000.00
1256349	400	Harrison	Herbert	54500.00
1121334	400	Strickling	Cletus	54500.00



Answer
Set

Correlated Subquery Example vs. a Join with a Derived Table

```
SELECT Last_Name, Dept_No, Salary  
FROM Employee_Table as EE  
WHERE Salary > (  
    SELECT AVG(Salary)  
    FROM Employee_Table as EEEE  
    WHERE EE.Dept_No = EEEE.Dept_No);
```

Correlated Subquery

Last_Name	Dept_No	Salary
Smith	200	48000.00
Harrison	400	54500.00
Strickling	400	54500.00

```
SELECT E.*, AVGSAL  
FROM Employee_Table as E  
INNER JOIN  
    (SELECT Dept_No, AVG(Salary)  
     FROM Employee_Table  
     GROUP BY Dept_No)  
    as TeraTom (Dept, AVGSAL)  
ON Dept_No = Dept  
AND Salary > AVGSAL;
```

Join with a Derived Table

Last_Name	Dept_No	Salary	AVGSAL
Smith	200	48000.00	44944.44
Harrison	400	54500.00	48333.33
Strickling	400	54500.00	48333.33

Both queries above will bring back all employees making a salary that is greater than the average salary in their department. The biggest difference is that the Join with the Derived Table also shows the Average Salary in the result set.

Correlated Subquery that Finds Duplicates

This is just a general example, because the Claim_Pay_Table does not exist in our database.



```
SELECT Provider_Name, Claim_Id, Subscriber_Name  
      ,Member_Name, Claim_Payment  
FROM Claims_Table clm JOIN Subscriber_Table sub  
  on clm.Subscriber_Id=sub.Subscriber_Id and  
clm.Member_No=sub.Member_No  
  JOIN Claim_Pay_Table cpt  
on cpt.Claim_Id=clm.Claim_Id  
WHERE 1 < (select count(*) from Claim_Table  
           where clm.Claim_Id=Claim_Id and  
clm.subscriber_no=Subscriber_No  
           and clm.Member_No=Member_No and  
clm.Provider_No=Provider_No);
```

There is another challenge faced by many people today and it relates to improper controls being used in the front-end systems and human error. Specifically, duplicate records can be a problem in the data as well as in the functioning of the organization. For instance, a physician may bill the healthcare insurance company twice and they might even pay it twice. Either way, this activity should exist in the data warehouse. The following query can be used to find this erroneous type of data or occurrence:

Quiz- Write the NOT Subquery

Customer_Table		Order_Table		
Customer_Number	Customer_Name	Order_Number	Customer_Number	Order_Total
11111111	Billy's Best Choice	123456	11111111	12347.53
31313131	Acme Products	123512	11111111	8005.91
31323134	ACE Consulting	123552	31323134	5111.47
57896883	XYZ Plumbing	123585	87323456	15231.62
87323456	Databases N-U	123777	57896883	23454.84

Write the Subquery

Select **all** columns in the **Customer_Table**
if the Customer has **NOT** placed an order.

Another opportunity knocking! Write the above query!

Answer to Quiz- Write the NOT Subquery

Customer_Table		Order_Table		
Customer_Number	Customer_Name	Order_Number	Customer_Number	Order_Total
11111111	Billy's Best Choice	123456	11111111	12347.53
31313131	Acme Products	123512	11111111	8005.91
31323134	ACE Consulting	123552	31323134	5111.47
57896883	XYZ Plumbing	123585	87323456	15231.62
87323456	Databases N-U	123777	57896883	23454.84

Write the Subquery

Select **all** columns in the **Customer_Table** if the Customer has **NOT** placed an order.

```
SELECT *
FROM Customer_Table
WHERE Customer_Number
      NOT IN
(SELECT Customer_Number
FROM Order_Table
WHERE Customer_Number
      IS NOT NULL);
```

```
SELECT *
FROM Customer_Table
WHERE Customer_Number
      NOT = ALL
(SELECT Customer_Number
FROM Order_Table
WHERE Customer_Number
      IS NOT NULL);
```

Wow! You can see that both queries are the same with just a few different techniques.

Quiz- Write the Subquery using a WHERE Clause

Customer_Table		Order_Table		
Customer_Number	Customer_Name	Order_Number	Customer_Number	Order_Total
11111111	Billy's Best Choice	123456	11111111	12347.53
31313131	Acme Products	123512	11111111	8005.91
31323134	ACE Consulting	123552	31323134	5111.47
57896883	XYZ Plumbing	123585	87323456	15231.62
87323456	Databases N-U	123777	57896883	23454.84

Write the Subquery

Select all columns in the Order_Table that were placed by a customer with ‘Bill’ anywhere in their name.

Another opportunity to show your brilliance is ready for you to make it happen.

Answer to Quiz- Write the Subquery using a WHERE Clause

Customer_Table		Order_Table		
Customer_Number	Customer_Name	Order_Number	Customer_Number	Order_Total
11111111	Billy's Best Choice	123456	11111111	12347.53
31313131	Acme Products	123512	11111111	8005.91
31323134	ACE Consulting	123552	31323134	5111.47
57896883	XYZ Plumbing	123585	87323456	15231.62
87323456	Databases N-U	123777	57896883	23454.84

Write the Subquery

Select all columns in the Order_Table that were placed by a customer with ‘Bill’ anywhere in their name.

```
SELECT      * FROM          Order_Table
WHERE      Customer_Number IN
          (SELECT Customer_Number FROM Customer_Table
           WHERE Customer_Name LIKE '%Bill%') ;
```

Great job on writing your query just like the above.

Quiz- Write the Subquery with Two Parameters

Customer_Table		Order_Table		
Customer_Number	Customer_Name	Order_Number	Customer_Number	Order_Total
11111111	Billy's Best Choice	123456	11111111	12347.53
31313131	Acme Products	123512	11111111	8005.91
31323134	ACE Consulting	123552	31323134	5111.47
57896883	XYZ Plumbing	123585	87323456	15231.62
87323456	Databases N-U	123777	57896883	23454.84

Write the Subquery

What is the **highest** dollar order for **each** Customer? This Subquery will involve two parameters!

Get ready to be amazed at either yourself or the Answer on the next page!

Answer to Quiz- Write the Subquery with Two Parameters

Customer_Table		Order_Table		
Customer_Number	Customer_Name	Order_Number	Customer_Number	Order_Total
11111111	Billy's Best Choice	123456	11111111	12347.53
31313131	Acme Products	123512	11111111	8005.91
31323134	ACE Consulting	123552	31323134	5111.47
57896883	XYZ Plumbing	123585	87323456	15231.62
87323456	Databases N-U	123777	57896883	23454.84

Write the Subquery

What is the **highest** dollar order for **each** Customer? This Subquery will involve two parameters!

```
SELECT Customer_Number, Order_Number, Order_Total  
FROM Order_Table  
WHERE (Customer_Number, Order_Total) IN  
(SELECT Customer_Number, MAX(Order_Total)  
FROM Order_Table GROUP BY 1);
```

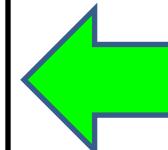
This is how you utilize multiple parameters in a Subquery! Turn the page for more.

How the Double Parameter Subquery Works

Customer_Table		Order_Table		
Customer_Number	Customer_Name	Order_Number	Customer_Number	Order_Total
11111111	Billy's Best Choice	123456	11111111	12347.53
31313131	Acme Products	123512	11111111	8005.91
31323134	ACE Consulting	123552	31323134	5111.47
57896883	XYZ Plumbing	123585	87323456	15231.62
87323456	Databases N-U	123777	57896883	23454.84

```
SELECT Customer_Number, Order_Number, Order_Total  
FROM Order_Table  
WHERE (Customer_Number, Order_Total) IN  
(SELECT Customer_Number, MAX(Order_Total)  
FROM Order_Table GROUP BY 1);
```

Customer_Number	Max(Order_Total)
11111111	12347.53
31323134	5111.47
87323456	15231.62
57896883	23454.84



These 4 rows
are sent to
the top query

The bottom query runs first returning two columns. Next page for more info!

More on how the Double Parameter Subquery Works

Customer_Table		Order_Table		
Customer_Number	Customer_Name	Order_Number	Customer_Number	Order_Total
11111111	Billy's Best Choice	123456	11111111	12347.53
31313131	Acme Products	123512	11111111	8005.91
31323134	ACE Consulting	123552	31323134	5111.47
57896883	XYZ Plumbing	123585	87323456	15231.62
87323456	Databases N-U	123777	57896883	23454.84

```
SELECT Customer_Number, Order_Number, Order_Total  
FROM Order_Table  
WHERE (Customer_Number, Order_Total) IN  
(  
    11111111 ,12347.53  
    31323134 , 5111.47  
    87323456 ,15231.62  
    57896883 ,23454.84 );
```

The top query now uses the In-list

The IN list is built and the top query can now process for the final Answer Set.

Quiz – Write the Triple Subquery

Customer_Table		Order_Table		
Customer_Number	Customer_Name	Order_Number	Customer_Number	Order_Total
11111111	Billy's Best Choice	123456	11111111	12347.53
31313131	Acme Products	123512	11111111	8005.91
31323134	ACE Consulting	123552	31323134	5111.47
57896883	XYZ Plumbing	123585	87323456	15231.62
87323456	Databases N-U	123777	57896883	23454.84

Write the Subquery

What is the **Customer_Name** who has the **highest** dollar order among all customers? This query will have multiple Subqueries!

Good luck in writing this. Remember that this will involve multiple Subqueries.

Answer to Quiz – Write the Triple Subquery

Customer_Table		Order_Table		
Customer_Number	Customer_Name	Order_Number	Customer_Number	Order_Total
11111111	Billy's Best Choice	123456	11111111	12347.53
31313131	Acme Products	123512	11111111	8005.91
31323134	ACE Consulting	123552	31323134	5111.47
57896883	XYZ Plumbing	123585	87323456	15231.62
87323456	Databases N-U	123777	57896883	23454.84

Write the Subquery

What is the **Customer_Name** who has the **highest** dollar order among all customers? This query will have multiple Subqueries!

```
SELECT Customer_Name  
FROM Customer_Table  
WHERE Customer_Number IN  
(SELECT Customer_Number FROM Order_Table  
WHERE Order_Total IN  
(SELECT Max(Order_Total) FROM Order_Table)) ;
```

The query is above and of course the answer is XYZ Plumbing.

Quiz – How many rows return on a NOT IN with a NULL?

Customer_Table		Order_Table		
Customer_Number	Customer_Name	Order_Number	Customer_Number	Order_Total
11111111	Billy's Best Choice	123456	11111111	12347.53
31313131	Acme Products	123512	11111111	8005.91
31323134	ACE Consulting	123552	31323134	5111.47
57896883	XYZ Plumbing	123585	87323456	15231.62
87323456	Databases N-U	123777	57896883	23454.84
		000099	?	99999.99

We added a Null Value to the Order_Table



```
SELECT Customer_Name  
FROM Customer_Table  
WHERE Customer_Number NOT IN  
(SELECT Customer_Number FROM Order_Table) ;
```

How many rows return from the query conceptually?

We really didn't place a new row inside the Order_Table with a NULL Customer_Number, but in theory if we had how many rows would return?

Answer – How many rows return on a NOT IN with a NULL?

Customer_Table		Order_Table		
Customer_Number	Customer_Name	Order_Number	Customer_Number	Order_Total
11111111	Billy's Best Choice	123456	11111111	12347.53
31313131	Acme Products	123512	11111111	8005.91
31323134	ACE Consulting	123552	31323134	5111.47
57896883	XYZ Plumbing	123585	87323456	15231.62
87323456	Databases N-U	123777	57896883	23454.84
		000099	?	99999.99

We added a Null Value to the Order_Table



```
SELECT Customer_Name  
FROM Customer_Table  
WHERE Customer_Number NOT IN  
(SELECT Customer_Number FROM Order_Table) ;
```

How many rows return from the query conceptually? **ZERO**

The answer is no rows. This is because when you have a NULL value in a NOT IN list the system doesn't know the value of NULL so it returns nothing.

How to handle a NOT IN with Potential NULL Values

Customer_Table		Order_Table		
Customer_Number	Customer_Name	Order_Number	Customer_Number	Order_Total
11111111	Billy's Best Choice	123456	11111111	12347.53
31313131	Acme Products	123512	11111111	8005.91
31323134	ACE Consulting	123552	31323134	5111.47
57896883	XYZ Plumbing	123585	87323456	15231.62
87323456	Databases N-U	123777	57896883	23454.84
		000099	?	99999.99

```
SELECT      Customer_Name  
FROM        Customer_Table  
WHERE       Customer_Number NOT IN  
            (SELECT Customer_Number FROM Order_Table  
             WHERE Customer_Number IS NOT NULL ) ;
```

NULL

How many rows return **NOW** from the query? 1 **Acme Products**

You can utilize a WHERE clause that tests to make sure Customer_Number IS NOT NULL. This should be used when a NOT IN could encounter a NULL.

IN is equivalent to =ANY

Customer_Table		Order_Table		
Customer_Number	Customer_Name	Order_Number	Customer_Number	Order_Total
11111111	Billy's Best Choice	123456	11111111	12347.53
31313131	Acme Products	123512	11111111	8005.91
31323134	ACE Consulting	123552	31323134	5111.47
57896883	XYZ Plumbing	123585	87323456	15231.62
87323456	Databases N-U	123777	57896883	23454.84

Instead of an IN you can use the = ANY

```
SELECT Customer_Number  
      ,Customer_Name  
FROM   Customer_Table  
WHERE  Customer_Number IN  
       (SELECT Customer_Number  
        FROM Order_Table ) ;
```

```
SELECT Customer_Number  
      ,Customer_Name  
FROM   Customer_Table  
WHERE  Customer_Number = ANY  
       (SELECT Customer_Number  
        FROM Order_Table ) ;
```

Instead of using the IN, you can use the = ANY command. These queries work the SAME. The above queries will produce the same result set.

Using a Correlated Exists

Customer_Table		Order_Table		
Customer_Number	Customer_Name	Order_Number	Customer_Number	Order_Total
11111111	Billy's Best Choice	123456	11111111	12347.53
31313131	Acme Products	123512	11111111	8005.91
31323134	ACE Consulting	123552	31323134	5111.47
57896883	XYZ Plumbing	123585	87323456	15231.62
87323456	Databases N-U	123777	57896883	23454.84

Use EXISTS to find which Customers have placed an Order?

```
SELECT      Customer_Number, Customer_Name  
FROM        Customer_Table as Top1  
WHERE       EXISTS  
           (SELECT * FROM Order_Table as Bot1  
            Where Top1.Customer_Number = Bot1.Customer_Number ) ;
```

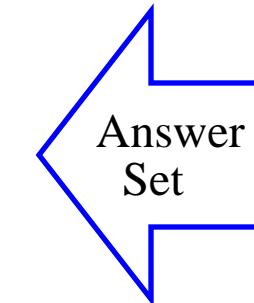
The EXISTS command will determine via a Boolean if something is True or False. If a customer placed and order it EXISTS and using the Correlated Exists statement only customers who have placed an order will return in the answer set. EXISTS is different than IN as it is less restrictive as you will soon understand.

How a Correlated Exists matches up

Customer_Table		Order_Table			
<u>Customer_Number</u>	<u>Customer_Name</u>	<u>Order Number</u>	<u>Customer Number</u>	<u>Order Total</u>	
11111111	Billy's Best Choice	123456	11111111	12347.53	
31313131	Acme Products	123512	11111111	8005.91	
31323134	ACE Consulting	123552	31323134	5111.47	
57896883	XYZ Plumbing	123585	87323456	15231.62	
87323456	Databases N-U	123777	57896883	23454.84	

```
SELECT Customer_Number, Customer_Name  
FROM Customer_Table as Top1  
WHERE EXISTS  
(SELECT * FROM Order_Table as Bot1  
Where Top1.Customer_Number = Bot1.Customer_Number) ;
```

Customer_Number	Customer_Name
11111111	Billy's Best Choice
31323134	ACE Consulting
57896883	XYZ Plumbing
87323456	Databases N-U



Only customers who placed an order return with the above Correlated EXISTS.

The Correlated NOT Exists

Customer_Table		Order_Table		
Customer_Number	Customer_Name	Order_Number	Customer_Number	Order_Total
11111111	Billy's Best Choice	123456	11111111	12347.53
31313131	Acme Products	123512	11111111	8005.91
31323134	ACE Consulting	123552	31323134	5111.47
57896883	XYZ Plumbing	123585	87323456	15231.62
87323456	Databases N-U	123777	57896883	23454.84

Use NOT EXISTS to find which Customers have NOT placed an Order?

```
SELECT      Customer_Number, Customer_Name  
FROM        Customer_Table as Top1  
WHERE       NOT EXISTS  
(SELECT * FROM Order_Table as Bot1  
Where Top1.Customer_Number = Bot1.Customer_Number) ;
```

The EXISTS command will determine via a Boolean if something is True or False. If a customer placed and order it EXISTS and using the Correlated Exists statement only customers who have placed an order will return in the answer set. EXISTS is different than IN as it is less restrictive as you will soon understand.

The Correlated NOT Exists Answer Set

Customer_Table		Order_Table		
<u>Customer_Number</u>	<u>Customer_Name</u>	<u>Order_Number</u>	<u>Customer_Number</u>	<u>Order_Total</u>
11111111	Billy's Best Choice	123456	11111111	12347.53
31313131	Acme Products	123512	11111111	8005.91
31323134	ACE Consulting	123552	31323134	5111.47
57896883	XYZ Plumbing	123585	87323456	15231.62
87323456	Databases N-U	123777	57896883	23454.84

Use NOT EXISTS to find which Customers have NOT placed an Order?

```
SELECT      Customer_Number, Customer_Name  
FROM        Customer_Table as Top1  
WHERE       NOT EXISTS  
(SELECT *  FROM Order_Table as Bot1  
Where Top1.Customer_Number = Bot1.Customer_Number) ;
```

Customer_Number	Customer_Name
31313131	Acme Products

Answer
Set

The only customer who did NOT place an order was Acme Products.

Quiz – How many rows come back from this NOT Exists?

Customer_Table		Order_Table		
Customer_Number	Customer_Name	Order_Number	Customer_Number	Order_Total
11111111	Billy's Best Choice	123456	11111111	12347.53
31313131	Acme Products	123512	11111111	8005.91
31323134	ACE Consulting	123552	31323134	5111.47
57896883	XYZ Plumbing	123585	87323456	15231.62
87323456	Databases N-U	123777	57896883	23454.84
		000099	?	99999.99

We added a Null Value to the Order_Table

 NULL

```
SELECT Customer_Number, Customer_Name  
FROM Customer_Table as Top1  
WHERE NOT EXISTS  
(SELECT * FROM Order_Table as Bot1  
Where Top1.Customer_Number = Bot1.Customer_Number);
```

How many rows return from the query conceptually?

A NULL value in a list for queries with NOT IN returned nothing, but you must now decide if that is also true for the NOT EXISTS. How many rows will return?

Answer – How many rows come back from this NOT Exists?

Customer_Table		Order_Table		
Customer_Number	Customer_Name	Order_Number	Customer_Number	Order_Total
11111111	Billy's Best Choice	123456	11111111	12347.53
31313131	Acme Products	123512	11111111	8005.91
31323134	ACE Consulting	123552	31323134	5111.47
57896883	XYZ Plumbing	123585	87323456	15231.62
87323456	Databases N-U	123777	57896883	23454.84
		000099	?	99999.99

 NULL

We added a Null Value to the Order_Table

```
SELECT      Customer_Number, Customer_Name  
FROM        Customer_Table as Top1  
WHERE       NOT EXISTS  
(SELECT * FROM Order_Table as Bot1  
Where Top1.Customer_Number = Bot1.Customer_Number) ;
```

How many rows return from the query conceptually? 1

NOT EXISTS is unaffected by a NULL in the list, that's why it is more flexible!

Chapter 25

Substrings and

Positioning Functions

“It is not the position but the disposition.’

- J.E. Dinger

Table of Contents Chapter 25 - Substrings and Positioning Functions

- [The CHARACTERS Command Counts Characters](#)
- [The CHARACTERS Command – Spaces can Count too](#)
- [Troubleshooting the CHARACTERS Command](#)
- [TRIM for Troubleshooting the CHARACTERS Command](#)
- [CHARACTERS and CHARACTER LENGTH equivalent](#)
- [OCTET LENGTH](#)
- [The TRIM Command trims both Leading and Trailing Spaces](#)
- [Trim and Trailing is Case Sensitive](#)
- [Trim and Trailing works if Case right](#)
- [Trim Combined with the CHARACTERS Command](#)
- [How to TRIM only the Trailing Spaces](#)
- [How to TRIM Trailing Letters](#)
- [A Visual Example of How to TRIM Trailing Letters](#)
- [How to TRIM Trailing Letters and use CHARACTER Length](#)
- [The SUBSTRING Command](#)
- [How SUBSTRING Works](#)
- [How SUBSTRING Works with NO ENDING POSITION](#)
- [Using SUBSTRING to move Backwards](#)
- [How SUBSTRING Works with a Starting Position of Zero](#)
- [How SUBSTRING Works with a Starting Position of -1](#)
- [How SUBSTRING Works with an Ending Position of 0](#)
- [An Example using SUBSTRING, TRIM and CHAR Together](#)

Continued on next page

Table of Contents Chapter 25 - Substrings and Positioning Functions Continued

- [SUBSTRING and SUBSTR are equal, but use different syntax](#)
- [The POSITION Command finds a Letters Position](#)
- [The POSITION Command is brilliant with SUBSTRING](#)
- [Quiz – Name that SUBSTRING Starting and For Length](#)
- [Answer to Quiz – Name that Starting and For Length](#)
- [Quiz – Find that SUBSTRING Starting Position](#)
- [Answer to Quiz – Find that SUBSTRING Starting Position](#)
- [Quiz – Find that SUBSTRING Starting FOR Length](#)
- [Answer to Quiz – Find that Starting FOR Length](#)
- [Quiz – Why Did only one Row Return](#)
- [Answer to Quiz – Why Did only one Row Return](#)
- [Concatenation](#)
- [Concatenation and SUBSTRING](#)
- [Four Concatenations Together](#)
- [Troubleshooting Concatenation](#)

The CHARACTERS Command Counts Characters

Employee_Table

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

VARCHAR

```
SELECT First_Name  
,CHARACTERS(First_Name) AS C_Length  
FROM Employee_Table  
WHERE CHARACTERS(First_Name) < 7  
ORDER BY 1;
```

Answer Set

First Name	C Length
Billy	5
Cletus	6
John	4
Mandee	6

The CHARACTERS command counts the number of characters . If 'Tom' was in the Employee_Table, his length would be 3

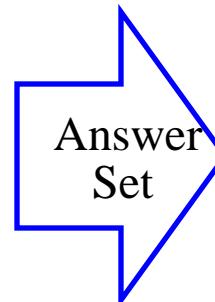
The CHARACTERS Command – Spaces can Count too

Employee_Table

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

VARCHAR

```
SELECT First_Name  
,CHARACTERS(First_Name) AS C_Length  
FROM Employee_Table  
WHERE CHARACTERS(First_Name) < 7  
ORDER BY 1;
```



First Name	C Length
Billy	5
Cletus	6
John	4
Mandee	6

If ‘To m’ was in the Employee_Table, his length would be 5. Yes, spaces do count as characters.

Troubleshooting the CHARACTERS Command

Last_Name is a **CHAR(20)** fixed length field.

CHAR(20)

```
SELECT Last_Name  
,CHARACTERS(Last_Name) AS C_Length  
FROM Employee_Table ;
```

Last_Name as a Char(20)

Jones	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	← Spaces
Hanson	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	
McRoberts	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	
Johnson	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	

When it comes to Characters, 20 would be the length of each and every name. That is because it has been set as a **CHAR(20)** in the table create syntax.

TRIM for Troubleshooting the CHARACTERS Command

Last_Name is a CHAR(20) fixed length field.

CHAR(20)

```
SELECT      Last_Name
,CHARACTERS(TRIM(Last_Name)) AS C_Length
FROM Employee_Table ;
```

Last_Name	C_Length
Jones	5
Smith	5
Smythe	6
Harrison	8
Chambers	8
Strickling	10
Reilly	6
Coffing	7
Larkins	7

The TRIM command will trim off any spaces before and after the Last_name.

CHARACTERS and CHARACTER_LENGTH equivalent

Query 1

```
SELECT      First_Name  
,CHARACTERS(First_Name) AS C_Length  
FROM Employee_Table ;
```

Query 2

```
SELECT      First_Name  
,CHARACTER_Length(First_Name) AS C_Length  
FROM Employee_Table ;
```

These two queries will get you the **SAME EXACT** answer set in your report.

OCTET_LENGTH

Query 1

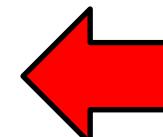
```
SELECT      First_Name  
,CHARACTERS(First_Name) AS C_Length  
FROM Employee_Table ;
```

Query 2

```
SELECT      First_Name  
,CHARACTER_Length(First_Name) AS C_Length  
FROM Employee_Table ;
```

Query 3

```
SELECT      First_Name  
,Octet_Length(First_Name) AS C_Length  
FROM Employee_Table ;
```



You can also use the OCTET LENGTH command. These three queries get the same exact answer sets! Query 2 and 3 are ANSI Standard.

The TRIM Command trims both Leading and Trailing Spaces

Query 1

```
SELECT      Last_Name  
,Trim(Last_Name) AS No_Spaces  
FROM Employee_Table ;
```

Query 2

```
SELECT      Last_Name  
,Trim(Both from Last_Name) AS No_Spaces  
FROM Employee_Table ;
```

Both queries trim both the leading and trailing spaces from Last_Name.

Trim and Trailing is Case Sensitive

VARCHAR

Query 1

```
SELECT      First_Name,  
Trim(trailing 'Y' from First_Name) AS No_Y  
FROM Employee_Table ;
```

‘Billy’ does not TRIM the trailing ‘y’ because it was after a capitol ‘Y’

For LEADING and TRAILNG, it IS case sensitive.

Trim and Trailing works if Case right

VARCHAR

Query 1

```
SELECT      First_Name,  
Trim(trailing 'y' from First_Name) AS No_y  
FROM Employee_Table ;
```

‘Billy’ now TRIMs the trailing ‘y’ and the answer becomes ‘Bill’

For LEADING and TRAILNG, it IS case sensitive.

Trim Combined with the CHARACTERS Command

Query 1

```
SELECT      Last_Name  
,Characters(Trim(Last_Name)) AS No_Spaces  
FROM Employee_Table ;
```

2
spaces 2
spaces



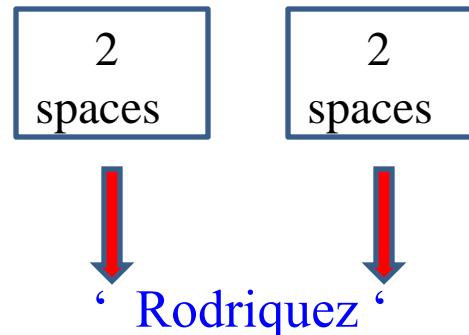
‘ Rodriquez ‘

This will allow for the character count to only be 9 because both the leading and trailing spaces have been cut.

How to TRIM only the Trailing Spaces

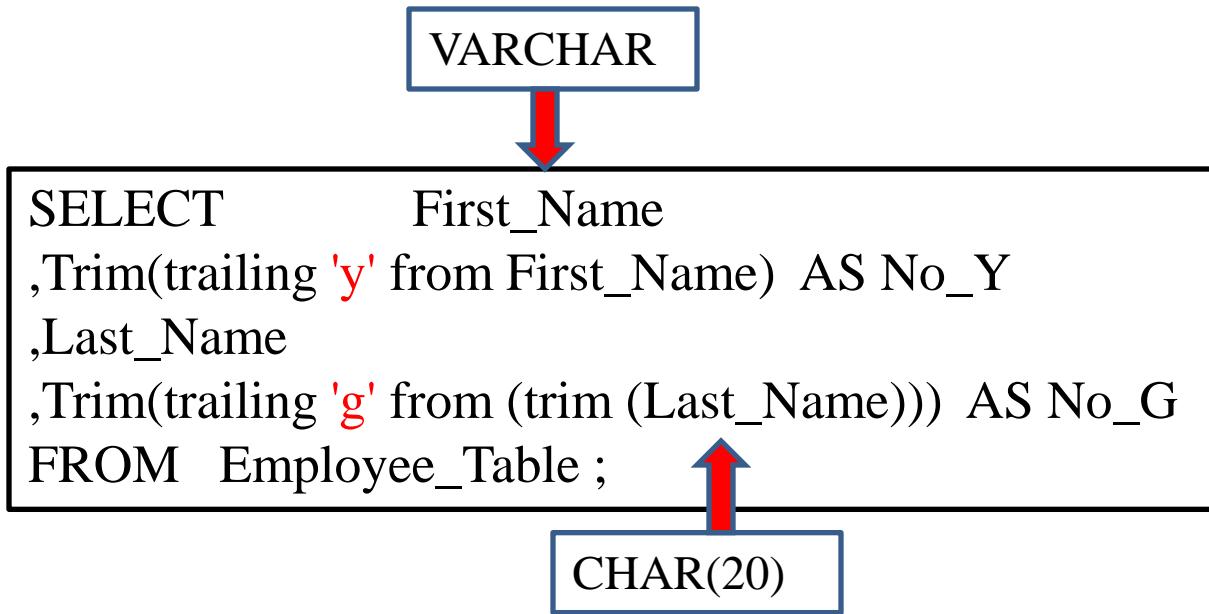
Query 1

```
SELECT      Last_Name  
,Characters(Trim(trailing  from Last_Name)) AS No_Spaces  
FROM Employee_Table ;
```



The TRAILING FROM Command allows you to only TRIM the spaces behind the Last_Name. Now, we will still get a character count of 11 because we are only cutting off the trailing spaces and not also the beginning spaces.

How to TRIM Trailing Letters



‘Billy’ ‘Coffing’

‘ becomes ‘Bill Coffin’

If ‘Billy’ was a First_Name and ‘Coffing’
be for No_Y and No_G?

‘ is the Last_Name, what is the result

A Visual Example of How to TRIM Trailing Letters

VARCHAR

```
SELECT First_Name  
,Trim(trailing 'y' from First_Name) AS No_Y  
,Last_Name  
,Trim(trailing 'g' from (trim (Last_Name))) AS No_G  
FROM Employee_Table ;
```

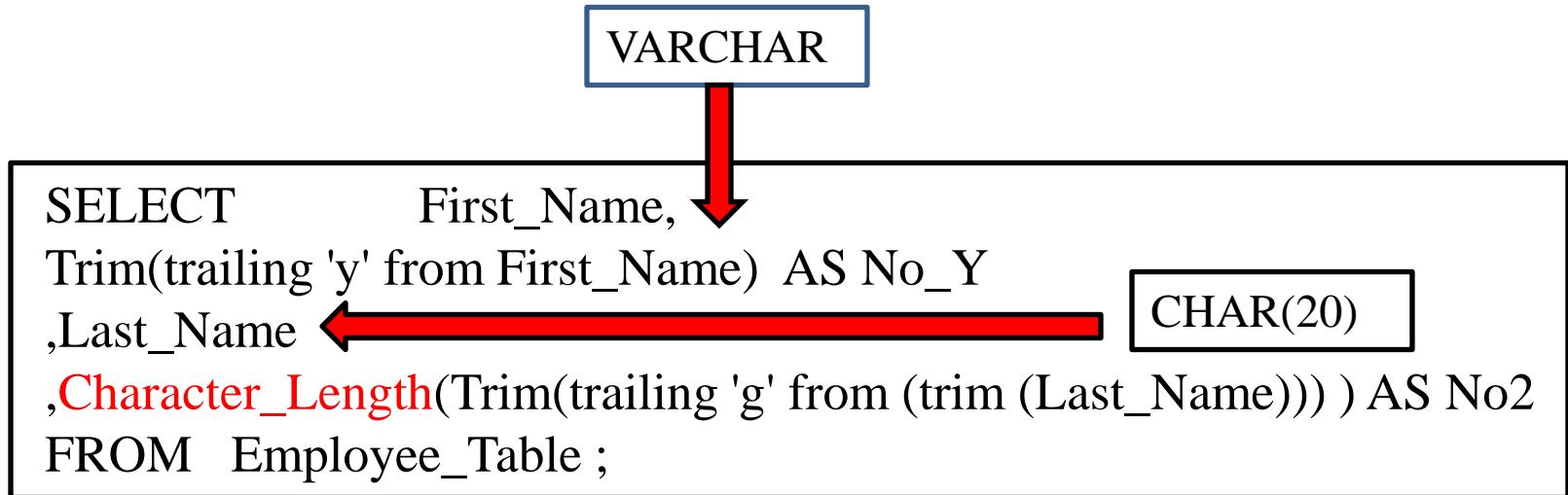
CHAR(20)

‘Billy‘ ‘Coffing‘

Answer:

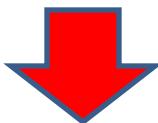
Bill Coffin

How to TRIM Trailing Letters and use CHARACTER_Length



‘Coffing

‘



Answer:

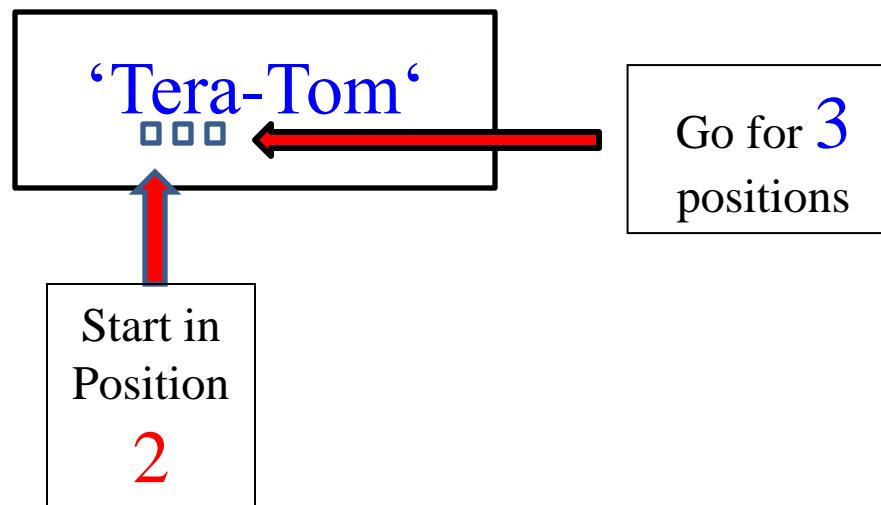
6

The SUBSTRING Command

VARCHAR(12)



```
SELECT      First_Name,  
SUBSTRING(First_Name FROM 2 for 3) AS Quiz  
FROM Employee_Table ;
```

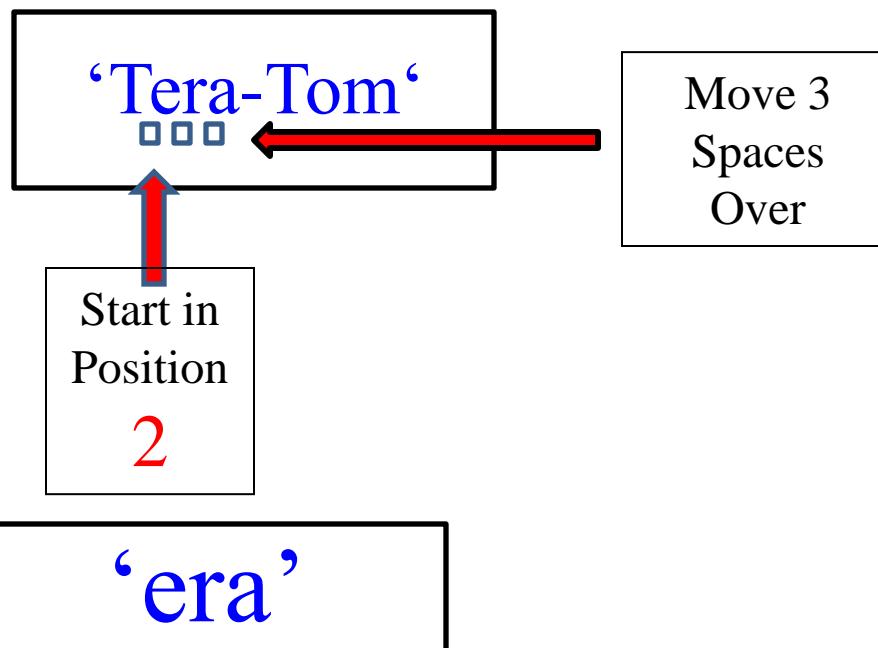


This is a SUBSTRING. What this does is start in position 2 and go for 3 positions, thus delivering the answer ‘ERA’.

How SUBSTRING Works

VARCHAR(12)

```
SELECT First_Name,  
SUBSTRING(First_Name FROM 2 for 3) AS Quiz  
FROM Employee_Table ;
```



In this example, if 'Tera-Tom' was the First_Name in the Employee_Table, we would start on the 2 character in the name and travel three characters. We'd take those characters and trim the rest and end up with 'era'.

How SUBSTRING Works with NO ENDING POSITION

VARCHAR(12)



```
SELECT First_Name,  
SUBSTRING(First_Name FROM 2) AS Quiz  
FROM Employee_Table ;
```

‘Tera-Tom’

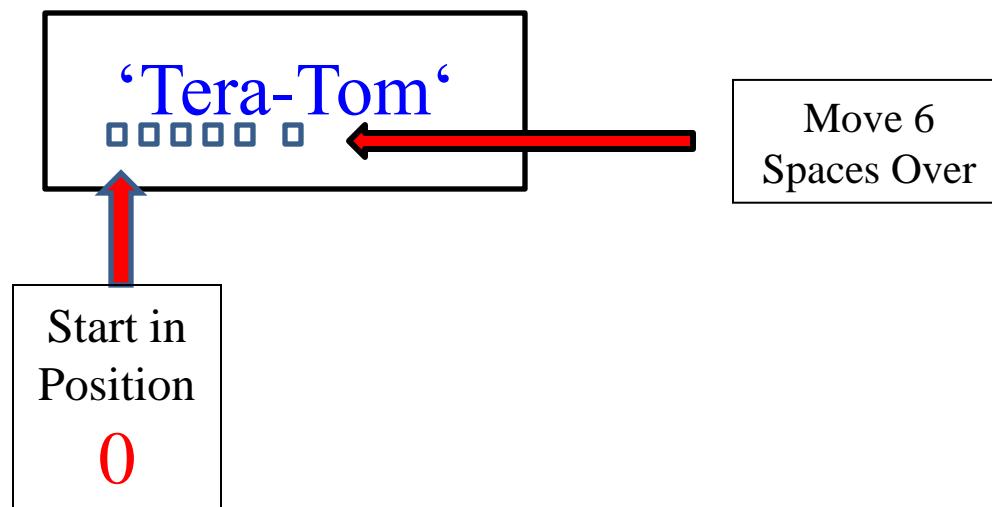
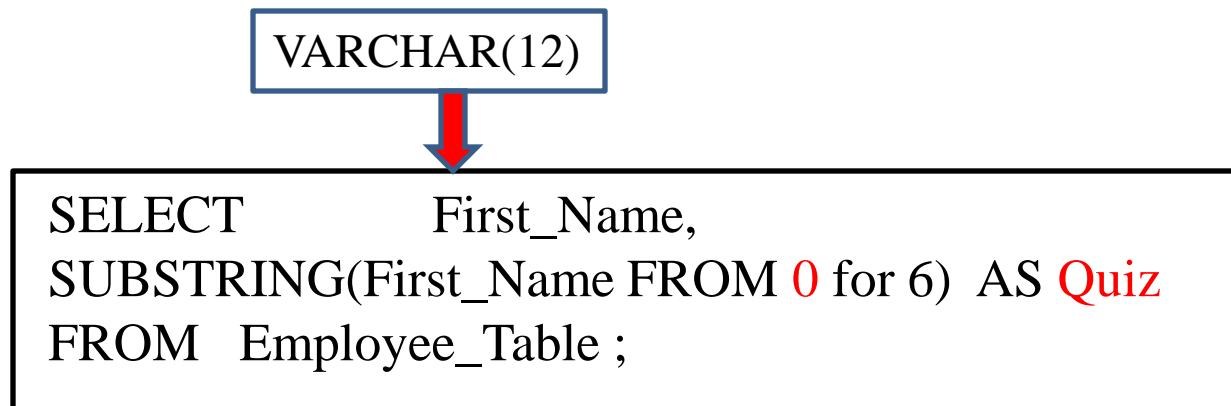
Start in Position
2

All the Way Over

‘era-Tom’

If you don't tell the Substring the end position, it will go all the way to the end.

Using SUBSTRING to move Backwards

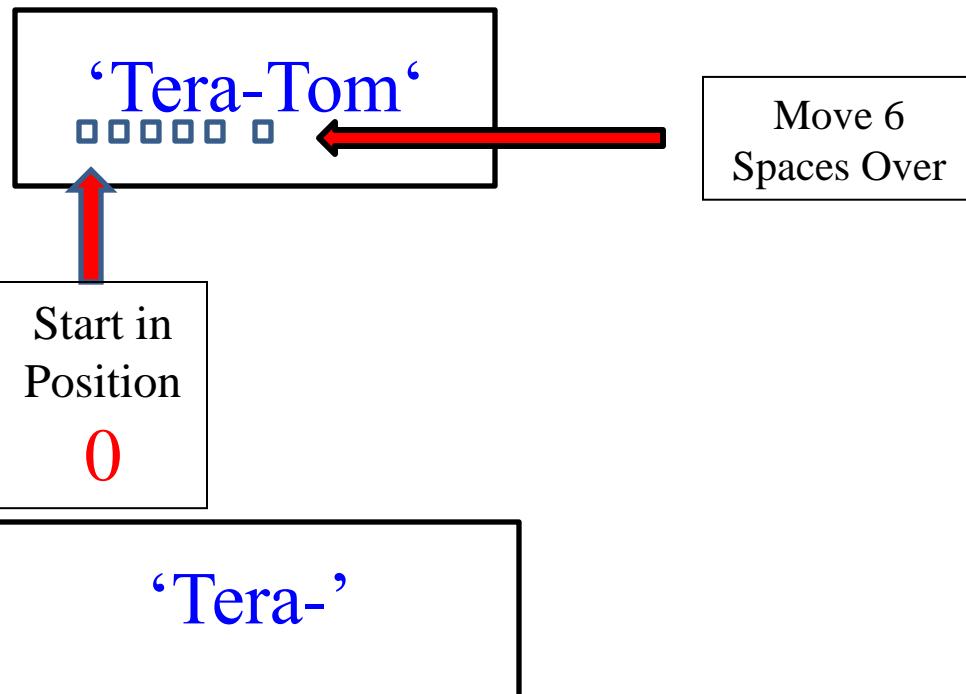


If ‘Tera-Tom’ was the First_Name in the Employee_Table what would be his result be for Quiz ?

How SUBSTRING Works with a Starting Position of Zero

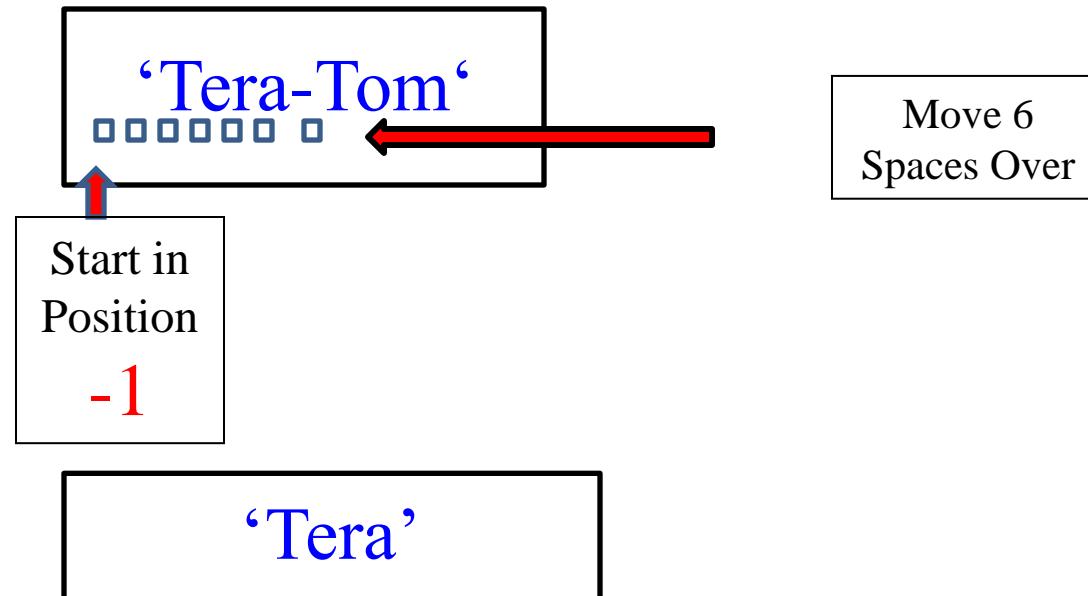
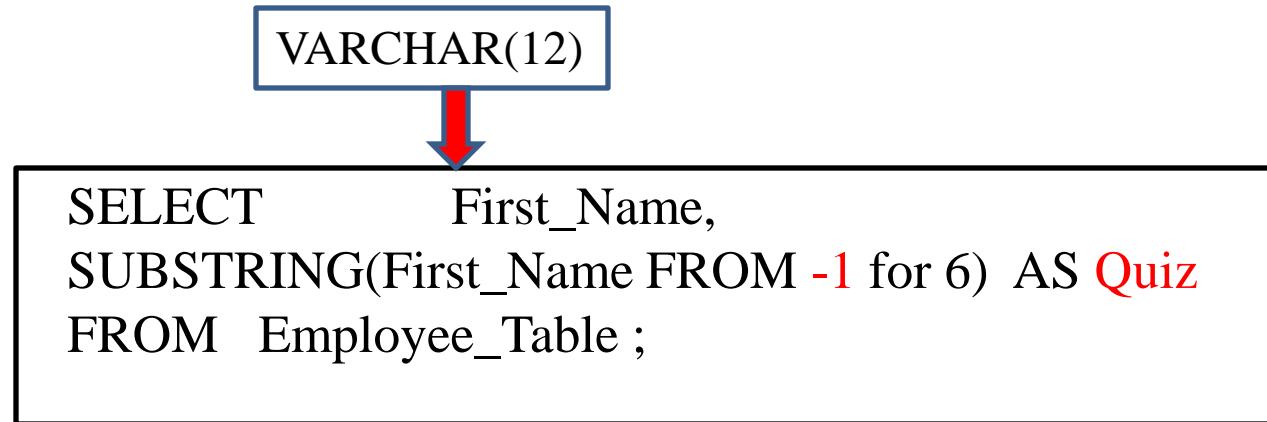
VARCHAR(12)

```
SELECT First_Name,  
SUBSTRING(First_Name FROM 0 for 6) AS Quiz  
FROM Employee_Table ;
```



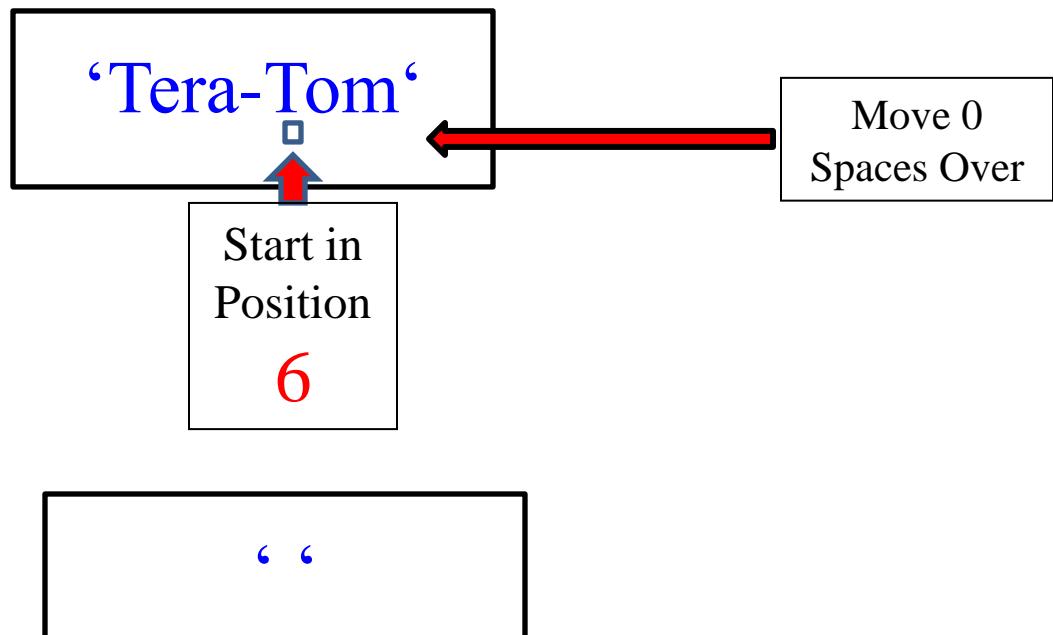
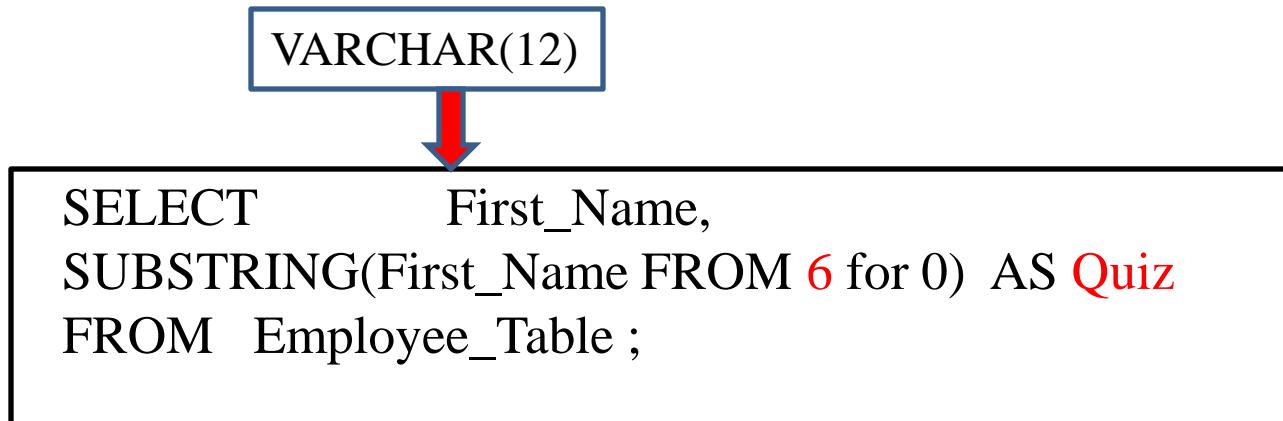
Since you are starting in the 0 position, you have to count a space in front of the name. Then you count over six from there.

How SUBSTRING Works with a Starting Position of -1



If ‘Tera-Tom’ was the First_Name in the Employee_Table what would be his result be for Quiz? TERA. Since you are starting in the -1 position, you have to count two spaces in front of the name. Then you count over six from there.

How SUBSTRING Works with an Ending Position of 0



If 'Tera-Tom' was the First_Name in the Employee_Table what would be his result be for Quiz? The result would be nothing.

An Example using SUBSTRING, TRIM and CHAR Together

CHAR(20)



```
SELECT Last_Name  
,SUBSTRING(Last_Name FROM  
CHARACTERS( TRIM (TRAILING FROM Last_Name)) -1 FOR 2) AS Letters  
FROM Employee_Table;
```

Last_Name	Letters
Jones	es
Smith	th
Smythe	he
Harrison	on
Chambers	rs
Strickling	ng
Reilly	ly
Coffing	ng
Larkins	ns

You can also combine substring and TRIM. If 'Coffing' was the Last_Name in the example above what would be his result for What_Letters?

' was the Last_Name in

SUBSTRING and SUBSTR are equal, but use different syntax

Query 1

VARCHAR(12)



```
SELECT First_Name,  
SUBSTRING(First_Name FROM 2 for 3)  
      AS Quiz  
FROM Employee_Table ;
```

Query 2

VARCHAR(12)



```
SELECT First_Name,  
SUBSTR (First_Name , 2 ,3)  
      AS Quiz  
FROM Employee_Table ;
```

Both queries above are going to yield the same results! **SUBSTR** is just a different way of doing a substring.

The POSITION Command finds a Letters Position

```
SELECT Last_Name  
      ,Position('e' in Last_Name) AS Find_The_E  
  FROM Employee_Table  
 WHERE Last_Name IN ('Smith', 'Jones');
```

Answer Set

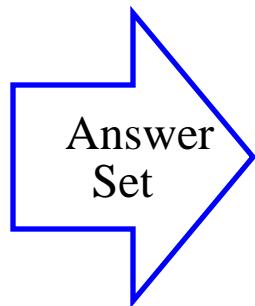
Last_Name	Find_The_E
Jones	4
Smith	0

E is 4th pos

This is the position counter. What it will do is tell you what position a letter is on. Why did Jones have a 4 in the result set? The 'e' was in the 4th position. Why did Smith get a zero. There is no 'e' in Smith. If there are two 'e's only the first is reported.

The POSITION Command is brilliant with SUBSTRING

```
SELECT Dept_No  
      ,Department_Name as Depty  
      ,SUBSTRING(Depty FROM 1 FOR POSITION(' ' IN  
                                         Department_Name) -1) as Word1  
FROM Department_Table;
```

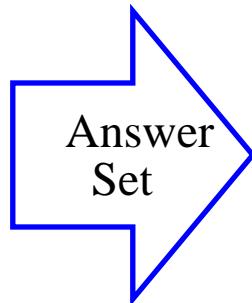


Dept_No	Depty	Word1
100	Marketing	Marketing
200	Research and Develop	Research
300	Sales	Sales
400	Customer Support	Customer
500	Human Resources	Human

What was the starting position of the Substring in the above query? It was one. The ending position (FOR length) was calculated to look for the first space and then subtract 1. So for “Research and Develop” the starting position was one and For 9-1 = 8.

Quiz – Name that SUBSTRING Starting and For Length

```
SELECT Dept_No  
      ,Department_Name as Depty  
     ,SUBSTRING(Depty FROM 1 FOR POSITION(' ' IN Department_Name) -1) as Word1  
FROM Department_Table;
```



Dept_No	Depty	Word1
100	Marketing	Marketing
200	Research and Develop	Research
300	Sales	Sales
400	Customer Support	Customer
500	Human Resources	Human

The FOR Length is calculated by finding the length up to the first SPACE and then subtracting 1.

Marketing (FROM FOR)

Research and Develop (FROM FOR)

Sales (FROM FOR)

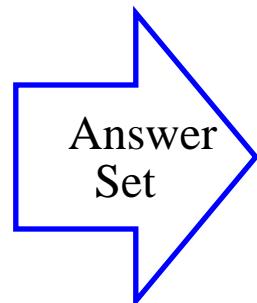
Customer Support (FROM FOR)

Human Resources (FROM FOR)

Fill in the number for the FROM and the FOR numbers above for each row. Next page!

Answer to Quiz – Name that Starting and For Length

```
SELECT Dept_No  
      ,Department_Name as Depty  
     ,SUBSTRING(DeptY FROM 1 FOR POSITION(' ' IN Department_Name) -1) as Word1  
  FROM Department_Table;
```



Dept_No	DeptY	Word1
100	Marketing	Marketing
200	Research and Develop	Research
300	Sales	Sales
400	Customer Support	Customer
500	Human Resources	Human

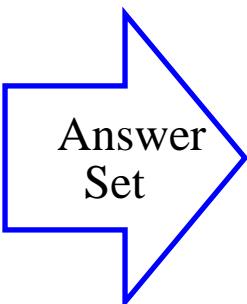
The FOR Length is calculated by finding the length up to the first SPACE and then subtracting 1.

- Marketing (FROM 1 FOR 9)
- Research and Develop (FROM 1 FOR 8)
- Sales (FROM 1 FOR 5)
- Customer Support (FROM 1 FOR 8)
- Human Resources (FROM 1 FOR 5)

The FOR was calculated in the POSITION Subquery.

Quiz – Find that SUBSTRING Starting Position

```
SELECT DISTINCT Department_Name as Dept_Name  
,SUBSTRING(Department_Name FROM  
POSITION(' ' IN Department_Name) +1) as Word2  
FROM Department_Table  
WHERE POSITION(' ' IN trim(Department_Name)) >0;
```



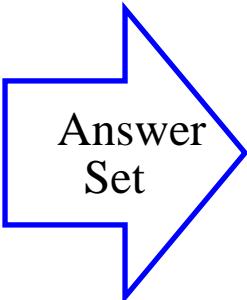
Dept_Name	Word2
Customer Support	Support
Human Resources	Resources
Research and Develop	and Develop

What is the Starting Position here?

What is the Starting position of the Substring in the above query? Hint: This only looks for a Dept_Name that has two words or more.

Answer to Quiz – Find that SUBSTRING Starting Position

```
SELECT DISTINCT Department_Name as Dept_Name  
,SUBSTRING(Department_Name FROM  
POSITION(' ' IN Department_Name) +1) as Word2  
FROM Department_Table  
WHERE POSITION(' ' IN trim(Department_Name)) >0;
```



Dept_Name	Word2
Customer Support	Support
Human Resources	Resources
Research and Develop	and Develop

The Starting Position is calculated by finding the length up to the first SPACE and then adding 1.

Customer Support (**FROM 10**)

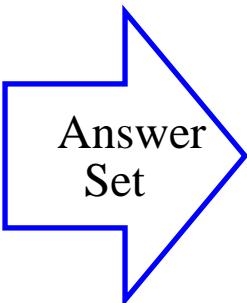
Human Resources (**FROM 7**)

Research and Develop **FROM 10**)

What is the Starting position of the Substring in the above query? See above!

Quiz – Find that SUBSTRING Starting FOR Length

```
SELECT DISTINCT Department_Name as Dept_Name  
,SUBSTRING(Department_Name FROM  
POSITION(' ' IN Department_Name) +1) as Word2  
FROM Department_Table  
WHERE POSITION(' ' IN trim(Department_Name)) >0;
```



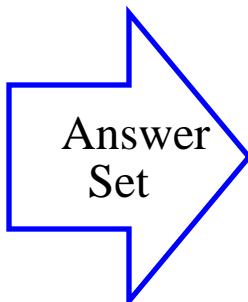
Dept_Name	Word2
Customer Support	Support
Human Resources	Resources
Research and Develop	and Develop

What is the FOR Position length here?

What is the FOR Length of the Substring in the above query?

Answer to Quiz – Find that Starting FOR Length

```
SELECT DISTINCT Department_Name as Dept_Name  
,SUBSTRING(Department_Name FROM  
POSITION(' ' IN Department_Name) +1) as Word2  
FROM Department_Table  
WHERE POSITION(' ' IN trim(Department_Name)) >0;
```



<u>Dept_Name</u>	<u>Word2</u>
Customer Support	Support
Human Resources	Resources
Research and Develop	and Develop

The Starting Position is calculated by finding the length up to the first SPACE and then adding 1.

Customer Support (FROM 10 FOR “go to the end”)

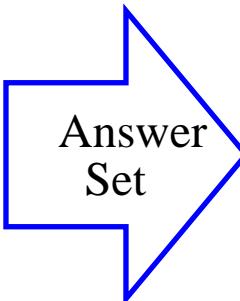
Human Resources (FROM 7 FOR “go to the end”)

Research and Develop FROM 10 FOR “go to the end”)

What is the FOR Position Length of the Substring in the above query? Since there was no FOR statement the Substring defaults to the end. So this query finds the first space and then adds 1 to the Starting Position and allows the rest to default to the end.

Quiz – Why Did only one Row Return

```
SELECT Department_Name
      ,SUBSTRING(Department_Name from
                  POSITION(' ' IN Department_Name) + 1 +
                  POSITION(' ' IN SUBSTRING(Department_Name
FROM POSITION(' ' IN Department_Name) + 1))) as Third_Word
      FROM Department_Table
      WHERE POSITION(' ' IN
                  TRIM(Substring(Department_Name from
                                  POSITION(' ' in Department_Name) + 1))))> 0
```

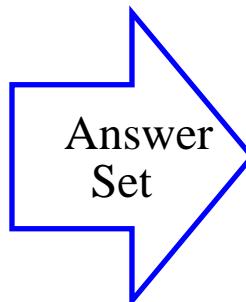


<u>Dept_Name</u>	<u>Third_Word</u>
Research and Develop	Develop

Why did only one row come back?

Answer to Quiz – Why Did only one Row Return

```
SELECT Department_Name  
      ,SUBSTRING(Department_Name from  
                  POSITION(' ' IN Department_Name) + 1 +  
                  POSITION(' ' IN SUBSTRING(Department_Name  
                  FROM POSITION(' ' IN Department_Name) + 1))) as Third_Word  
FROM Department_Table  
WHERE POSITION(' ' IN  
              TRIM(Substring(Department_Name from  
                  POSITION(' ' in Department_Name) + 1)))> 0
```



Dept_Name	Third_Word
Research and Develop	Develop

It has 3 words

Why did only one row come back? It's the Only Department Name with three words. The SUBSTRING and the WHERE clause both look for the first space and if they find it then look for the second space. If they find that add 1 to it and their Starting Position is the third word. There is no FOR position so it defaults to “go to the end”.

Concatenation

Two Pipe Symbols
together (no space)
mean concatenate

```
SELECT First_Name  
      ,Last_Name  
      ,First_Name  
      || ''  
      || Last_Name as Full_Name  
FROM Employee_Table  
WHERE First_Name = 'Squiggy'
```

Answer
Set

First_Name	Last_Name	Full_Name
Squiggy	Jones	Squiggy Jones

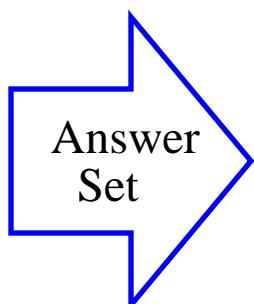
See those || ? Those represent concatenation. That allows you to combine multiple columns into one column. The || (Pipe Symbol) on your keyboard is just above the ENTER key. Don't put a space in between, but just put two Pipe Symbols together.

In this example, we have combined the **first name**, then a **single space**, and then the **last name** to get a new column called 'Full name', like **Squiggy Jones**.

Concatenation and SUBSTRING

```
SELECT First_Name  
      ,Last_Name  
      ,Substring(First_Name, 1, 1) || '.' || Last_Name  
                           as Full_Name  
  
FROM Employee_Table  
WHERE First_Name = 'Squiggy'
```

A Period (.)
and a space



First Name	Last Name	Full Name
Squiggy	Jones	S. Jones

Of the three items being concatenated together what is the first item of concatenation in the example above? The first initial of the First_Name. Then we concatenated a literal space and a period. Then we concatenated the Last_Name.

Four Concatenations Together

```
SELECT First_Name  
      ,Last_Name
```

CHAR(20)

VARCHAR



```
,TRIM(Last_Name)||' '|| Substring(First_Name, 1, 1)||':'  
                           as Last_Name_1st
```

```
FROM Employee_Table
```

```
WHERE First_Name = 'Squiggy' ;
```

Answer
Set

First_Name	Last_Name	Last_Name_1st
Squiggy	Jones	Jones S.

Why did we TRIM the Last_Name? To get rid of the spaces or the output would have looked odd. How many items are being concatenated in the example above? There are 4 items concatenated. We start with the Last_Name (after we trim it), then we have a single space, then we have the First Initial of the First Name and then we have a Period.

Troubleshooting Concatenation

```
SELECT First_Name  
      ,Last_Name  
      ,TRIM (Last_Name)||' '|| Substring(First_Name, 1, 1) || ':'  
                           as Last_Name_1st  
  
FROM Employee_Table  
WHERE First_Name = 'Squiggy' ;
```

ERROR

What happened above to cause the error. Can you see it? The Pipe Symbols || have a space between them like | |, when it should be |||. It is a tough one to spot so be careful.

Chapter 26

Interrogating the Data

“We live in a world of things and our only connection with them is that we know how to manipulate or to consume them.”

-Erich Fromm

Table of Contents Chapter 26 - Interrogating the Data

- [Quiz – What would the Answer be?](#)
- [Answer to Quiz – What would the Answer be?](#)
- [The NULLIFZERO Command](#)
- [Quiz – Fill in the Blank Values in the Answer Set](#)
- [Answer to Quiz – Fill in the Blank Values in the Answer Set](#)
- [Quiz – Fill in the Answers for the NULLIF Command](#)
- [Quiz – Fill in the Answers for the NULLIF Command](#)
- [The ZEROIFNULL Command](#)
- [Answer to the ZEROIFNULL Question](#)
- [The COALESCE Command](#)
- [The COALESCE Answer Set](#)
- [The Coalesce Quiz](#)
- [Answers to the Coalesce Quiz](#)
- [The Basics of CAST \(Convert And STore\)](#)
- [Some Great CAST \(Convert And STore\) Examples](#)
- [Some Great CAST \(Convert And STore\) Examples](#)
- [Some Great CAST \(Convert And STore\) Examples](#)
- [A Teradata Extension – The Implied Cast](#)
- [The Basics of the CASE Statements](#)
- [The Basics of the CASE Statement shown Visually](#)
- [Valued Case Statement Vs Searched Case Statement](#)
- [Valued Case Statement](#)

Continued on next page

Table of Contents Chapter 26 - Interrogating the Data Continued

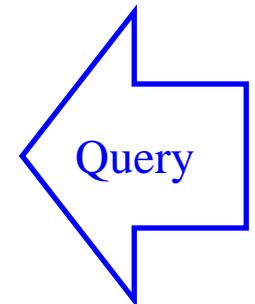
- [Searched Case Statement](#)
- [Searched Case Statement](#)
- [When NO ELSE is present in CASE Statement](#)
- [When NO ELSE is present in CASE Statement](#)
- [When an ELSE is present in CASE Statement](#)
- [When NO ELSE is present in CASE Statement](#)
- [When an Alias is NOT used in a CASE Statement](#)
- [When an Alias is NOT used in a CASE Statement](#)
- [When NO ELSE is present in CASE Statement](#)
- [Combining Searched Case and Valued Case](#)
- [A Trick for getting a Horizontal Case](#)
- [Nested Case](#)
- [Put a CASE in the ORDER BY](#)

Quiz – What would the Answer be?

Sample_Table

Class_Code	Grade_Pt
Fr	0

```
SELECT Class_Code  
      ,Grade_pt / (Grade_pt * 2 ) as Math1  
FROM Sample_Table  
ORDER BY 1,2 ;
```



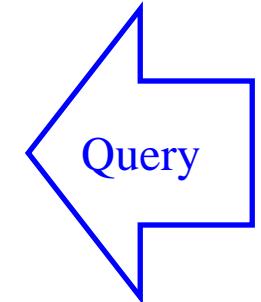
Can you guess what would return in the Answer Set?

Use the fake table above called Sample_Table and try and predict what the Answer will be if this query was running on the system.

Answer to Quiz – What would the Answer be?

Sample_Table	
Class_Code	Grade_Pt
Fr	0

```
SELECT Class_Code  
      ,Grade_pt / (Grade_pt * 2 ) as Math1  
FROM Sample_Table  
ORDER BY 1,2 ;
```



Can you guess what would return in the Answer Set?

ERROR – Division by Zero

You get an error when you DIVIDE by ZERO! Let's turn the page and fix it!

The NULLIFZERO Command

Sample_Table

Class_Code	Grade_Pt
Fr	0

Notice that in our sample table, the value is '0'.

```
SELECT Class_Code  
,Grade_Pt / ( NULLIFZERO(Grade_pt) * 2 ) AS Math1  
FROM Sample_Table;
```

Query

Class_Code	Math1
Fr	?

Answer set

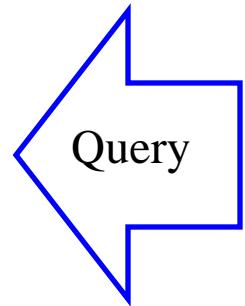
What the NULLIFZERO does is make a zero into a NULL. So, the answer set you'd get from this is a simple 'FR', and then a NULL value represented usually by a '?'. If you have a calculation where a ZERO could kill the operation and you don't want that you can use the NULLIFZERO command to convert any zero value to a NULL value.

Quiz – Fill in the Blank Values in the Answer Set

Sample_Table

Cust_No	Acc_Balance	Location
0	?	3

```
SELECT NULLIFZERO(Cust_No)
      ,NULLIFZERO(Acc_Balance)
      ,NULLIFZERO(Location)
FROM Sample_Table ;
```



Cust_No	Acc_Balance	Location
0	0	3

Answer set

Okay! Time to show me your brilliance! What would the Answer Set produce?

Answer to Quiz – Fill in the Blank Values in the Answer Set

Sample_Table

Cust_No	Acc_Balance	Location
0	?	3

```
SELECT NULLIFZERO(Cust_No)
      ,NULLIFZERO(Acc_Balance)
      ,NULLIFZERO(Location)
FROM Sample_Table ;
```

Query

Cust_No	Acc_Balance	Location
?	?	3

Answer set

Here is the answer set! How'd you do? The NULLIFZERO command found a zero in Cust_No so it made it Null. The others were not zero so they retained their value. The only time NULLIFZERO changes data is if it finds a zero, and then it changes it to null.

Quiz – Fill in the Answers for the NULLIF Command

Sample_Table

Cust_No	Acc_Balance	Location
0	?	3

```
SELECT NULLIF(Cust_No, 0)      AS Cust1  
    ,NULLIF(Cust_No, 3)      AS Cust2  
    ,NULLIF(Acc_Balance, 0)  AS Acc1  
    ,NULLIF(Acc_Balance, 3)  AS Acc2  
    ,NULLIF(Location, 0)     AS Loc1  
    ,NULLIF(Location, 3)     AS Loc2  
  
FROM Sample_Table;
```

Cust1	Cust2	Acc1	Acc2	Loc1	Loc2
-------	-------	------	------	------	------

What would the answers be?

Query

Answer set

You can also use the NULLIF(). What you are asking Teradata to do is to **NULL** the answer if the COLUMN matches the number in the parentheses. What would the above Answer Set produce from your analysis?

Quiz – Fill in the Answers for the NULLIF Command

Sample_Table

Cust_No	Acc_Balance	Location
0	?	3

```
SELECT NULLIF(Cust_No, 0)      AS Cust1  
    ,NULLIF(Cust_No, 3)      AS Cust2  
    ,NULLIF(Acc_Balance, 0)  AS Acc1  
    ,NULLIF(Acc_Balance, 3)  AS Acc2  
    ,NULLIF(Location, 0)     AS Loc1  
    ,NULLIF(Location, 3)     AS Loc2  
  
FROM Sample_Table;
```

Query

Cust1	Cust2	Acc1	Acc2	Loc1	Loc2
?	0	?	?	3	?

Answer set

Look at the answers above and if it doesn't make sense go over it again until it does.

The ZEROIFNULL Command

Sample_Table		
Cust_No	Acc_Balance	Location
0	?	3

Notice the Null! We're turning it into a '0' shortly!

```
SELECT ZEROIFNULL(Cust_No) as Cust  
      ,ZEROIFNULL(Acc_Balance) as Balance  
      ,ZEROIFNULL(Location) as Location  
FROM Sample_Table ;
```

Query

Cust	Balance	Location
_____	_____	_____

What would the answers be?

Answer set

This is the **ZEROIFNULL**. What it will do is put a zero into a place where a NULL shows up. What would the Answer Set produce?

Answer to the ZEROIFNULL Question

Sample_Table

Cust_No	Acc_Balance	Location
0	?	3



Notice the Null! We're turning it into a '0' shortly!

```
SELECT ZEROIFNULL(Cust_No) as Cust  
,ZEROIFNULL(Acc_Balance) as Balance  
,ZEROIFNULL(Location) as Location  
FROM Sample_Table ;
```

Query

Cust	Balance	Location
0	0	3

Answer set

The answer set placed a zero in the place of the NULL Acc_Balance, but the other values didn't change because they were NOT Null.

The COALESCE Command

Sample_Table

Last_Name	Home_Phone	Work_Phone	Cell_Phone
Jones	555-1234	444-1234	?
Patel	?	456-7890	454-6789
Gonzales	?	?	354-0987
Nguyen	?	?	?

```
SELECT Last_Name  
,COALESCE (Home_Phone, Work_Phone, Cell_Phone) as Phone  
FROM Sample_Table ;
```

Last_Name Phone
What would the answers
be for all four rows?

Coalesce returns the first non-Null value in a list, and if all values are Null returns Null.

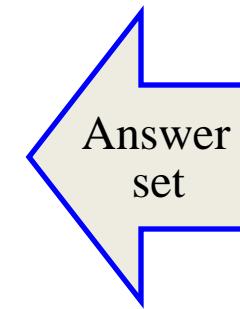
The COALESCE Answer Set

Sample_Table

Last_Name	Home_Phone	Work_Phone	Cell_Phone
Jones	555-1234	444-1234	?
Patel	?	456-7890	454-6789
Gonzales	?	?	354-0987
Nguyen	?	?	?

```
SELECT Last_Name  
,COALESCE (Home_Phone, Work_Phone, Cell_Phone) as Phone  
FROM Sample_Table ;
```

Last_Name	Phone
Jones	555-1234
Patel	456-7890
Gonzales	354-0987
Nguyen	?



Coalesce returns the first non-Null value in a list, and if all values are Null returns Null.

The Coalesce Quiz

Sample_Table			
Last_Name	Home_Phone	Work_Phone	Cell_Phone
Jones	555-1234	444-1234	?
Patel	?	456-7890	454-6789
Gonzales	?	?	354-0987
Nguyen	?	?	?

```
SELECT Last_Name  
      ,COALESCE (Home_Phone, Work_Phone, Cell_Phone, 'No Phone')  
                 as Phone  
FROM Sample_Table ;
```

Last_Name	Phone
What would the answers be for all four rows?	

Coalesce returns the first non-Null value in a list, and if all values are Null returns Null. Since we decided in the above query we don't want NULLs notice we have placed a literal 'No Phone' in the list. How will this effect the Answer Set?

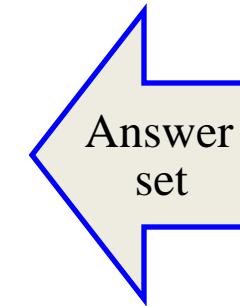
Answers to the Coalesce Quiz

Sample_Table

Last_Name	Home_Phone	Work_Phone	Cell_Phone
Jones	555-1234	444-1234	?
Patel	?	456-7890	454-6789
Gonzales	?	?	354-0987
Nguyen	?	?	?

```
SELECT Last_Name  
      ,COALESCE (Home_Phone, Work_Phone, Cell_Phone, 'No Phone')  
                  as Phone  
FROM Sample_Table ;
```

Last_Name	Phone
Jones	555-1234
Patel	456-7890
Gonzales	354-0987
Nguyen	No Phone



Answers are above! We put a literal in the list so there's no chance of NULL returning.

The Basics of CAST (Convert And STore)

CAST will convert a column or value's data type temporarily into another data type.

Below is the syntax:

```
SELECT CAST(<column-name> AS <data-type>[(<length>)] )  
FROM <table-name> ;
```

Examples using CAST:

CAST (<smallint-data> AS CHAR(5))	/* convert smallint to character */
CAST (<decimal-data> AS INTEGER)	/* truncates decimals */
CAST (<byteint-data> AS SMALLINT)	/* convert binary to smallint */
CAST (<char-data> AS BYTE (128))	/* convert character to binary */
CAST (<byteint-data> AS VARCHAR(5))	/* convert byteint to character */
CAST (<integer-data> AS FLOAT)	/* convert integer to float point */

Data can be converted from one type to another by using the CAST function. As long as the data involved does not break any data rules (i.e. placing alphabetic or special characters into a numeric data type) the conversion works. The name of the CAST function comes from the Convert And STore operation that it performs.

Some Great CAST (Convert And STore) Examples

```
SELECT  CAST('ABCDE' AS CHAR(1)) AS Trunc  
        ,CAST(128 AS CHAR(3)) AS OK  
        ,CAST(127 AS INTEGER) AS Bigger ;
```

<u>Trunc</u>	<u>OK</u>	<u>Bigger</u>
A	128	127

The first CAST truncates the five characters (left to right) to form the single character 'A'. In the second CAST, the integer 128 is converted to three characters and left justified in the output. The 127 was initially stored in a SMALLINT (5 digits - up to 32767) and then converted to an INTEGER. Hence, it uses 11 character positions for its display, ten numeric digits and a sign (positive assumed) and right justified as numeric.

Some Great CAST (Convert And STore) Examples

```
SELECT CAST(121.53 AS SMALLINT) AS Whole  
      ,CAST(121.53 AS DECIMAL(3,0)) AS Rounder ;
```

<u>Whole</u>	<u>Rounder</u>
121	122

The value of 121.53 was initially stored as a DECIMAL as 5 total digits with 2 of them to the right of the decimal point. Then it is converted to a SMALLINT using CAST to remove the decimal positions. Therefore, it truncates data by stripping off the decimal portion. It does not round data using this data type. On the other hand, the CAST in the fifth column called Rounder is converted to a DECIMAL as 3 digits with no digits (3,0) to the right of the decimal, so it will round data values instead of truncating. Since .53 is greater than .5, it is rounded up to 122.

Some Great CAST (Convert And STore) Examples

```
SELECT Order_Number as OrdNo  
      ,Customer_Number as CustNo  
      ,Order_Date  
      ,Order_Total  
      ,CAST(Order_Total as integer)      as Chopped  
      ,CAST(Order_Total as Decimal(5,0)) as Rounded  
FROM Order_Table ;
```

OrdNo	CustNo	Order_Date	Order_Total	Chopped	Rounded
123585	87323456	10/10/1999	15231.62	15231	15232
123777	57896883	09/09/1999	23454.84	23454	23455
123512	11111111	01/01/1999	8005.91	8005	8006
123456	11111111	05/04/1998	12347.53	12347	12348
123552	31323134	10/01/1999	5111.47	5111	5111

The Column Chopped takes Order_Total (a Decimal (10,2) and CASTS it as an integer, which chops off the decimals. Rounded CASTS Order_Total as a Decimal (5,0), which takes the decimals and rounds up if the decimal is .50 or above.

A Teradata Extension – The Implied Cast

```
SELECT <column-name> ( <data-type> [(<length>)]  
FROM <table-name> ;
```

```
SELECT      'ABCDE' (CHAR(1)) AS Shortened  
           ,128 (CHAR(3))      AS OOPS1  
           ,-128 (CHAR(3))     AS OOPS2  
           ,128 (INTEGER)      AS Bigger  
           ,121.13 (SMALLINT)   AS Whole ;
```

<u>Shortened</u>	<u>OOPS1</u>	<u>OOPS2</u>	<u>Bigger</u>	<u>Whole</u>
A		- 12	128	121

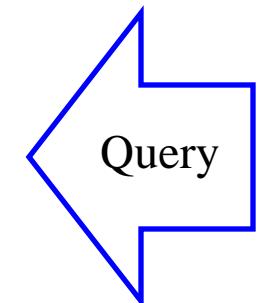
This Teradata extension conversion is requested by placing the “implied” data type conversion in parentheses after the column name. What happened in the column named OOPS1 and OOPS2? the value 128 is 1 greater than 127 and therefore too large of a value to store in a BYTEINT. So it is automatically stored as a SMALLINT (5 digits plus a sign) before the conversion. The implicit conversion changes it to a character type with the first 3 characters being returned. As a result, only the first 3 spaces are seen in the report (_ _ 128). Likewise, OOPS2 is stored as (_ _ -128) with the first three characters (2 spaces and -) shown in the output.

The Basics of the CASE Statements

Sample_Table

Course_Name	Credits
Tera-Tom on SQL	1

```
SELECT Course_Name  
      ,CASE Credits  
        WHEN 1 THEN 'One Credit'  
        WHEN 2 THEN 'Two Credits'  
        WHEN 3 THEN 'Three Credits'  
      END AS CreditAlias  
FROM Sample_Table ;
```



Course_Name	CreditAlias
-------------	-------------

What will the answer be to this query?

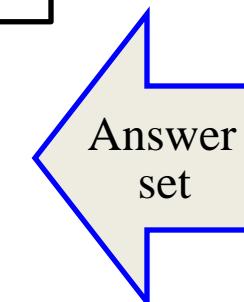
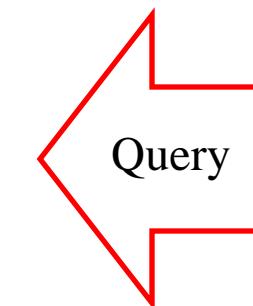
This is a CASE STATEMENT, which allows you to do evaluate a column in your table and from that, come up with a new answer for your report. Every CASE begins with a CASE and they all must end with a corresponding END. What would the answer be?

The Basics of the CASE Statement shown Visually

Sample_Table	
Course_Name	Credits
Tera-Tom on SQL	1

```
SELECT Course_Name  
      ,CASE Credits  
        WHEN 1 THEN 'One Credit'  
        WHEN 2 THEN 'Two Credits'  
        WHEN 3 THEN 'Three Credits'  
      END AS CreditAlias  
FROM Sample_Table ;
```

Course_Name	CreditAlias
Tera-Tom on SQL	One Credit



This is a CASE STATEMENT, which allows you to do evaluate a column in your table and from that, come up with a new answer for your report. Every CASE begins with a CASE and they all must end with a corresponding END. What would the answer be?

Valued Case Statement Vs Searched Case Statement

```
SELECT Course_Name  
      ,CASE Credits ←  
        WHEN 1 THEN 'One Credit'  
        WHEN 2 THEN 'Two Credits'  
        WHEN 3 THEN 'Three Credits'  
        Else 'Credits not found'  
      END AS CreditAlias  
FROM Course_Table ;
```

Value follows the word CASE so this is a **VALUED CASE!**

Rules:

You can only check for equality
You can only check the value Credits

```
SELECT Course_Name  
      ,CASE ←  
        WHEN Credits <= 1 THEN 'One'  
        WHEN Credits = 2 THEN 'Two'  
        WHEN Credits < 4 THEN 'Three'  
        WHEN Course_Name like 'Tera%' Then 'Four'  
        Else 'Don''t know'  
      END AS CreditAlias  
FROM Course_Table ;
```

NO Value follows CASE so It's a **SEARCHED CASE!**

Rules:

Check any way you want
Check other values also

The second example is better unless you have a simple query like the first example.

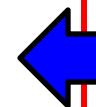
Valued Case Statement

Course_Table				
Course_ID	Course_Name	Credits	Seats	
100	Database Concepts	3	50	
200	Introduction to SQL	3	20	
210	Advanced SQL	3	22	
220	V2R3 SQL Features	2	25	
300	Physical Database Design	4	20	
400	Database Administration	4	16	

```
SELECT Course_Name  
,CASE Credits  
    WHEN 1 THEN 'One Credit'  
    WHEN 2 THEN 'Two Credits'  
    WHEN 3 THEN 'Three Credits'  
    Else 'Credits not found'  
END AS CreditAlias  
FROM Course_Table ORDER BY 1 ;
```

Course_Name
Advanced SQL
Database Administration
Database Concepts
Introduction to SQL
Physical Database Design
V2R3 SQL Features

CreditAlias



Fill in the Answers
for CreditAlias

Look at the CASE Statement and look at the Course_Table and fill in the Answer Set.

Valued Case Statement

Course_Table

Course_ID	Course_Name	Credits	Seats
100	Teradata Concepts	3	50
200	Introduction to SQL	3	20
210	Advanced SQL	3	22
220	V2R3 SQL Features	2	25
300	Physical Database Design	4	20
400	Database Administration	4	16

SELECT Course_Name

,CASE Credits

WHEN 1 THEN 'One Credit'

WHEN 2 THEN 'Two Credits'

WHEN 3 THEN 'Three Credits'

Else 'Credits not found'

END AS CreditAlias

FROM Course_Table ORDER BY 1 ;

Course_Name

Advanced SQL

Database Administration

Introduction to SQL

Physical Database Design

Teradata Concepts

V2R3 SQL Features

CreditAlias

Three Credits

Credits not found

Three Credits

Credits not found

Three Credits

Two Credits

Above is the full answer set.

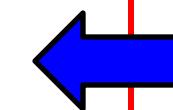
Searched Case Statement

Course_Table				
Course_ID	Course_Name	Credits	Seats	
100	Teradata Concepts	3	50	
200	Introduction to SQL	3	20	
210	Advanced SQL	3	22	
220	V2R3 SQL Features	2	25	
300	Physical Database Design	4	20	
400	Database Administration	4	16	

```
SELECT Course_Name as CCC  
,CASE  
    WHEN Credits <= 1 THEN 'One'  
    WHEN Credits = 2 THEN 'Two'  
    WHEN Credits < 4 THEN 'Three'  
    WHEN CCC like 'Tera%' Then '4'  
    Else 'Don''t know'  
END AS CreditAlias  
FROM Course_Table ORDER BY 1;
```

CCC
Advanced SQL
Database Administration
Introduction to SQL
Physical Database Design
Teradata Concepts
V2R3 SQL Features

CreditAlias



Fill in the Answers
for CreditAlias

Look at the CASE Statement and look at the Course_Table and fill in the Answer Set.

Searched Case Statement

Course_Table				
Course_ID	Course_Name	Credits	Seats	
100	Teradata Concepts	3	50	
200	Introduction to SQL	3	20	
210	Advanced SQL	3	22	
220	V2R3 SQL Features	2	25	
300	Physical Database Design	4	20	
400	Database Administration	4	16	

```
SELECT Course_Name as CCC  
,CASE  
    WHEN Credits <= 1 THEN 'One'  
    WHEN Credits = 2 THEN 'Two'  
    WHEN Credits < 4 THEN 'Three'  
    WHEN CCC like 'Tera%' Then '4'  
        Else 'Don''t know'  
    END AS CreditAlias  
FROM Course_Table ORDER BY 1;
```

CCC	CreditAlias
Advanced SQL	Three
Database Administration	Don't know
Introduction to SQL	Three
Physical Database Design	Don't know
Teradata Concepts	Three
V2R3 SQL Features	Two

Above is the full answer set.

When NO ELSE is present in CASE Statement

Sample_Table	
<u>Course_Name</u>	<u>Credits</u>
Tera-Tom on SQL	4

```
SELECT Course_Name  
      ,CASE Credits  
        WHEN 1 THEN 'One Credit'  
        WHEN 2 THEN 'Two Credits'  
        WHEN 3 THEN 'Three Credits'  
      END AS CreditAlias  
FROM Sample_Table ;
```

<u>Course_Name</u>	<u>CreditAlias</u>
Tera-Tom on SQL	

Fill in the Answer
for CreditAlias

Notice now that we have a 4 under the 'Credit' Column. However, in our CASE statement, we don't have instructions on what to do if the number is 4. What will occur?

When NO ELSE is present in CASE Statement

Sample_Table	
Course_Name	Credits
Tera-Tom on SQL	4

```
SELECT Course_Name  
      ,CASE Credits  
        WHEN 1 THEN 'One Credit'  
        WHEN 2 THEN 'Two Credits'  
        WHEN 3 THEN 'Three Credits'  
      END AS CreditAlias  
FROM Sample_Table ;
```

Course_Name	CreditAlias
Tera-Tom on SQL	?

A NULL Value is the result.

Since our value of 4 fell through the CASE statement without an ELSE statement a NULL value is returned in the Answer Set.

When an ELSE is present in CASE Statement

Sample_Table	
Course_Name	Credits
Tera-Tom on SQL	4

```
SELECT Course_Name  
      ,CASE Credits  
        WHEN 1 THEN 'One Credit'  
        WHEN 2 THEN 'Two Credits'  
        WHEN 3 THEN 'Three Credits'  
        ELSE 'Don''t Know'  
      END AS CreditAlias  
FROM Sample_Table ;
```

Course_Name	CreditAlias
Tera-Tom on SQL	

Fill in the Answer
for CreditAlias

Notice now that we have a 4 under the 'Credit' Column. However, in our CASE statement, we don't have instructions on what to do if the number is 4. What will occur?

When NO ELSE is present in CASE Statement

Sample_Table	
Course_Name	Credits
Tera-Tom on SQL	4

```
SELECT Course_Name  
      ,CASE Credits  
        WHEN 1 THEN 'One Credit'  
        WHEN 2 THEN 'Two Credits'  
        WHEN 3 THEN 'Three Credits'  
        ELSE 'Don''t Know'  
      END AS CreditAlias  
FROM Sample_Table ;
```

Course_Name	CreditAlias
Tera-Tom on SQL	Don't Know

Notice the pretty apostrophe!

Since our value of 4 fell through the CASE statement the ELSE statement kicked in and we delivered 'Don't Know'. Notice two single quotes that provided the word Don't.

When an Alias is NOT used in a CASE Statement

Sample_Table

Course_Name	Credits
Tera-Tom on SQL	4

```
SELECT Course_Name  
      ,CASE Credits  
        WHEN 1 THEN 'One Credit'  
        WHEN 2 THEN 'Two Credits'  
        WHEN 3 THEN 'Three Credits'  
        ELSE 'Don''t Know'  
      END  
FROM Sample_Table ;
```

NO Alias

Course_Name	
Tera-Tom on SQL	Don't Know

Fill in the Title if
you think you can

Notice now that we don't have an ALIAS for the CASE Statement. What will the system place in there for the Column Title.

When an Alias is NOT used in a CASE Statement

Sample_Table	
Course_Name	Credits
Tera-Tom on SQL	4

```
SELECT Course_Name  
      ,CASE Credits  
        WHEN 1 THEN 'One Credit'  
        WHEN 2 THEN 'Two Credits'  
        WHEN 3 THEN 'Three Credits'  
        ELSE 'Don''t Know'  
      END  
  FROM Sample_Table ;
```

NO Alias

Course_Name	< CASE expression >
Tera-Tom on SQL	Don't Know

Here is the title!

Notice now that we don't have an ALIAS for the CASE Statement. The title given by default is < CASE Expression >. That is why you should ALIAS your Case statements.

When NO ELSE is present in CASE Statement

Sample_Table

Course_Name	Credits
Tera-Tom on SQL	4

```
SELECT Course_Name  
      ,CASE Credits  
        WHEN 1 THEN 'One Credit'  
        WHEN 2 THEN 'Two Credits'  
        WHEN 3 THEN 'Three Credits'  
        ELSE 'Don''t Know'  
      END AS CreditAlias  
FROM Sample_Table ;
```

Course_Name	CreditAlias
Tera-Tom on SQL	Don't Know

Notice the pretty apostrophe!

Since our value of 4 fell through the CASE statement the ELSE statement kicked in and we delivered 'Don't Know'. Notice two single quotes that provided the word Don't.

Combining Searched Case and Valued Case

```
SELECT Last_Name
```

```
,CASE ← Searched Case
```

```
WHEN Grade_Pt IS NULL THEN 'Null'
```

```
WHEN Grade_Pt IN (1, 2, 3) THEN 'Integer GPA'
```

```
WHEN Grade_Pt Between 1 and 2 THEN 'Low GPA'
```

```
WHEN Grade_Pt < 4 THEN 'High GPA'
```

```
ELSE '4.0 GPA'
```

```
END AS "GPA Info"
```

```
,CASE Class_Code ← Valued Case
```

```
WHEN 'FR' THEN 'Year-1'
```

```
WHEN 'SO' THEN 'Year-2'
```

```
WHEN 'JR' THEN 'Year-3'
```

```
WHEN 'SR' THEN 'Year-4'
```

```
ELSE 'Null'
```

```
END As "Year"
```

```
FROM Student_Table
```

```
ORDER BY "Year";
```

Searched Case

NO Value follows CASE so
It's a **SEARCHED CASE!**

Searched Case Rules:

Check any way (=, <, >, IN, etc)

Check other values if you want

Last_Name	GPA Info	Year
Johnson	Null	Null
Thomas	4.0 GPA	Year-1
Hanson	High GPA	Year-1
Larkins	High GPA	Year-1
Smith	Integer GPA	Year-2
Wilson	High GPA	Year-2
McRoberts	Low GPA	Year-3
Bond	High GPA	Year-3
Phillips	Integer GPA	Year-4
Delaney	High GPA	Year-4

This Query above uses both a **Valued Case** and **Searched Case**. That's ALLOWED!

A Trick for getting a Horizontal Case

```
SELECT AVG(CASE Class_Code  
          WHEN 'FR' THEN Grade_pt  
          ELSE NULL END) (format 'Z.ZZ') AS Freshman_GPA  
      ,AVG(CASE Class_Code  
          WHEN 'SO' THEN Grade_pt  
          ELSE NULL END) (format 'Z.ZZ') AS Sophomore_GPA  
      ,AVG(CASE Class_Code  
          WHEN 'JR' THEN Grade_pt  
          ELSE NULL END) (format 'Z.ZZ') AS Junior_GPA  
      ,AVG(CASE Class_Code  
          WHEN 'SR' THEN Grade_pt  
          ELSE NULL END) (format 'Z.ZZ') AS Senior_GPA  
FROM Student_Table  
WHERE Class_Code IS NOT NULL ;
```

Freshman_GPA	Sophomore_GPA	Junior_GPA	Senior_GPA
2.29	2.90	2.92	3.18

Answer
Set

Aggregates ignore Nulls so knowing this trick has allowed for Horizontal Reporting

Nested Case

```
SELECT Last_Name  
,CASE Class_Code  
    WHEN 'JR' THEN 'Jr '  
        ||(CASE WHEN Grade_pt < 2 THEN 'Failing'  
            WHEN Grade_pt < 3.5 THEN 'Passing'  
            ELSE 'Exceeding'  
        END)  
        ELSE 'Sr '  
        ||(CASE WHEN Grade_pt < 2 THEN 'Failing'  
            WHEN Grade_pt < 3.5 THEN 'Passing'  
            ELSE 'Exceeding'  
        END)  
    END AS Status  
FROM Student_Table WHERE Class_Code IN ('JR','SR')  
ORDER BY Class_Code, Last_Name;
```

The diagram illustrates the execution flow of the nested CASE statement. It starts with a blue square labeled "Answer Set" at the top, which has two downward-pointing arrows leading to a blue triangle. From the triangle, two arrows point down to a table. The table has two columns: "Last_Name" and "Status". The data rows are: Bond (Jr Exceeding), McRoberts (Jr Failing), Delaney (Sr Passing), and Phillips (Sr Passing).

Last_Name	Status
Bond	Jr Exceeding
McRoberts	Jr Failing
Delaney	Sr Passing
Phillips	Sr Passing

A NESTED Case occurs when you have a Case Statement within another CASE Statement. Notice the Double Pipe symbols (||) that provide Concatenation.

Put a CASE in the ORDER BY

```
SELECT * FROM Student_Table  
ORDER BY CASE Class_Code
```

CASE in the
ORDER BY
Statement

```
WHEN 'FR' THEN 1  
WHEN 'SO' THEN 2  
WHEN 'JR' THEN 3  
WHEN 'SR' THEN 4  
ELSE 5
```

```
END ;
```

Student_ID	Last_Name	First_Name	Class_Code	Grade_Pt
234121	Thomas	Wendy	FR	4.00
125634	Hanson	Henry	FR	2.88
423400	Larkins	Michael	FR	0.00
333450	Smith	Andy	SO	2.00
231222	Wilson	Susie	SO	3.80
322133	Bond	Jimmy	JR	3.95
280023	McRoberts	Richard	JR	1.90
324652	Delaney	Danny	SR	3.35
123250	Phillips	Martin	SR	3.00
260000	Johnson	Stanley	?	?

Chapter 27

View Functions

“It is easier to go down a hill than up it but the view is
much better at the top.”

-Arnold Bennett

Table of Contents Chapter 27 - View Functions

- [Creating a Simple View](#)
- [Basic Rules for Views](#)
- [How to Modify a View](#)
- [Exceptions to the ORDER BY Rule inside a View](#)
- [How to Get HELP with a View](#)
- [Views sometimes CREATED for Formatting or Row Security](#)
- [Another Way to Alias Columns in a View CREATE](#)
- [Resolving Aliasing Problems in a View CREATE](#)
- [Resolving Aliasing Problems in a View CREATE](#)
- [Resolving Aliasing Problems in a View CREATE](#)
- [CREATING Views for Complex SQL such as Joins](#)
- [WHY certain columns need Aliasing in a View](#)
- [Aggregates on View Aggregates](#)
- [Locking Row for Access](#)
- [Altering A Table](#)
- [Altering A Table After a View has been Created](#)
- [A View that won't work after an ALTER](#)
- [Troubleshooting a View](#)
- [Updating Data in a Table through a View](#)
- [Maintenance Restrictions on a Table through a View](#)

Creating a Simple View

Employee_Table

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

```
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**

```
CREATE View Department_Salaries AS  
SELECT      Dept_No  
            ,SUM(Salary) as SumSal  
            ,SUM(Salary) / 12 as MonthSal  
FROM Employee_Table  
GROUP BY 1;
```

```
SEL *  
FROM Department_Salaries  
Order By 1 ;
```

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

1 /* CREATE the View */

```
CREATE View Employee_V2 AS  
SELECT Employee_No  
    ,First_Name  
    ,Last_Name  
    ,Dept_No  
    ,Salary  
FROM Employee_Table ;
```

2 /* SHOW View Command */

```
SHOW View Employee_V2 ;
```

Answer
Set

3

```
CREATE View Employee_V2 AS  
SELECT Employee_No  
    ,First_Name  
    ,Last_Name  
    ,Dept_No  
    ,Salary  
FROM Employee_Table ;
```

4

```
REPLACE View Employee_V2 AS  
SELECT      Employee_No  
            ,First_Name  
            ,Last_Name  
            ,Dept_No  
FROM Employee_Table ;
```

COPY

The Column Salary has been removed

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

Exceptions to the ORDER BY Rule inside a View

TOP
Command!
Oh, Okay!

/* CREATING A View */

Create view emp4v4 as
select **TOP 3** * from Employee_Table
ORDER BY Salary DESC;

ANSI OLAP
Command!
Oh, Okay!

/* CREATING A View with ANSI OLAP */

Create view Sales_Olap_V as
SELECT Product_ID, Sale_Date, Daily_Sales
,Sum(Daily_Sales) OVER (**ORDER BY** Daily_Sales
Rows Unbounded Preceding) as “CSUM”
FROM Sales_Table

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_View9 ;

<u>Column Name</u>	<u>Type</u>	<u>Comment</u>	<u>Nullable</u>	<u>Format</u>	<u>Title</u>
Employee_No	?		?	?	?
Dept_No	?		?	?	?
Last_Name	?		?	?	?
First_Name	?		?	?	?
Salary	?		?	?	?
<u>Max Length</u>	<u>Decimal Total Digits</u>	<u>Decimal Fractional Digits</u>	<u>Range Low</u>	<u>Range High</u>	
?	?	?	?	?	?
?	?	?	?	?	?
?	?	?	?	?	?
?	?	?	?	?	?
<u>UpperCase</u>	<u>Table/View?</u>		<u>Default value</u>	<u>Char Type</u>	<u>IdCol Type</u>
?	?	?	?	?	?
?	?	?	?	?	?
?	?	?	?	?	?
?	?	?	1	?	?

The Help View command do 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

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

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 ;
```

SELECTING from A View

```
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 ;
```

SELECTING from A View

```
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 ;
```

SELECTING from A View

```
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.

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 also 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 ;
```

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

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

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;
```

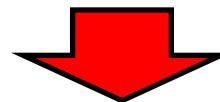
The Employee_Table used above will automatically use an ACCESS Lock, which allows ACCESS during UPDATES or table loads.

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?

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;
```



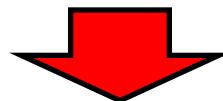
YES!

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 ;
```

```
-- Altering the actual Table  
Alter Table Employee_Table4  
Add Mgr_No Integer ;
```



Will the View STILL run?

Select from the View

```
SELECT * FROM Emp_HR_v4;
```



YES!

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

A View that won't work after an ALTER

```
CREATE VIEW Emp_HR_v5 AS  
SELECT Employee_No  
      ,Dept_No  
      ,Last_Name  
      ,First_Name  
FROM Employee_Table5 ;
```

```
-- Altering the actual Table  
Alter Table Employee_Table5  
Drop Dept_No;
```

Will the View STILL run?

```
-- Select from the View
```

```
SELECT * FROM Emp_HR_v5;
```

No! ERROR

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 ;
```

```
-- Altering the actual Table  
Alter Table Employee_Table6  
Drop Dept_No;
```

Will the View STILL run?

```
-- Select from the View
```

```
SELECT * FROM Emp_HR_v6;
```

No! Error

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 and 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  
WHERE Employee_No = 2000000;
```

-- SELECT from the actual Table

```
SEL * from Employee_Table8  
Where Employee_No = 2000000;
```

Employee_No	Dept_No	Last_Name	First_Name	Salary
2000000	?	Jones	Squiggy	8888.88

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

Chapter 28

Macro Functions

“Don’t count the days. Make the days count.”

-Muhammad Ali

Table of Contents Chapter 28 - Macro Functions

- [The 14 rules of Macros](#)
- [CREATING and EXECUTING a Simple Macro](#)
- [Multiple SQL Statements inside a Macro](#)
- [Complex Joins inside a Macro](#)
- [Passing an INPUT Parameter to a Macro](#)
- [Troubleshooting a Macro with INPUT Parameters](#)
- [Troubleshooting a Macro with INPUT Parameters](#)
- [An UPDATE Macro with Two Input Parameters](#)
- [Executing a Macro with Named \(Not Positional\) Parameters](#)
- [Troubleshooting a Macro](#)

The 14 rules of Macros

1. Macros can contain one or more SQL Statements
2. Macros are a database Extension in Teradata and not ANSI-Standard
3. Macros don't require PERM Space
4. Macros are stored in the Data Dictionary in the USER DBC
5. Every statement in the macro must end in a semi-colon ;
6. Only one DDL Statement can exist in a Macro and it must be the last statement
7. Input Parameters can be passed to the Macro
8. Macros run when users use the EXEC Macro Command
9. Macros can be executed from any viable front end, especially the Nexus
10. Macros can call views and even other Macros
11. All SQL Statements inside a Macro are considered **one Transaction**
12. Each Macro name must be a unique object within the database it resides
13. USERS need only the EXEC privilege to run a macro
14. No underlying privileges to tables inside the macro need be required

Macros are database extensions and reside in the **Data Dictionary in DBC**.

CREATING and EXECUTING a Simple Macro

```
CREATE Macro SQL01.Emp1_xyz AS  
(SELECT *  
FROM SQL_Class.Employee_Table  
WHERE Dept_No IN (300, 400) ; ) ;
```

Ends the SQL

Ends the CREATE

EXEC command
runs a Macro

-- Executing the Macro

```
EXEC sql01.Emp1_xyz;
```

Employee_No	Dept_No	Last_Name	First_Name	Salary
2312225	300	Larkins	Lorraine	40200.00
1256349	400	Harrison	Herbert	54500.00
1121334	400	Strickling	Cletus	54500.00
2341218	400	Reilly	William	36000.00

Only ONE transaction will run by executing any macro. Every Macro will have a semi-colon to end each SQL statement and an additional semi-colon to end the CREATE.

Multiple SQL Statements inside a Macro

```
CREATE Macro Emp2_mac AS  
(SELECT * FROM Employee_Table  
WHERE Dept_No = 400 ;
```

```
EXEC Emp2_mac ;
```

```
SELECT * FROM Department_Table  
WHERE Dept_No = 400 ; ) ;
```

Employee_No	Dept_No	Last_Name	First_Name	Salary
1256349	400	Harrison	Herbert	54500.00
1121334	400	Strickling	Cletus	54500.00
2341218	400	Reilly	William	36000.00

Dept_No	Department_Name	Mgr_No	Budget
400	Customer Support	1256349	500000.00

Still, only ONE transaction ran by executing the macro here. Everything in a macro is considered ONE transaction. Notice we have two SQL statements that end with a semi-colon, but we also have a semi-colon to end the CREATE MACRO Statements.

Complex Joins inside a Macro

```
CREATE Macro Emp3_mac AS  
(SELECT E.* , Mgr_No, Budget  
FROM Employee_Table as E  
INNER JOIN  
        Department_Table as D ON E.Dept_No = D.Dept_No ; ) ;
```

-- Executing the Macro

EXEC Emp3_mac ;

Complexity
made EASY!

Employee_No	Dept_No	Last_Name	First_Name	Salary	Mgr_No	Budget
1232578	100	Chambers	Mandee	48850.00	1232578	500000.00
1324657	200	Coffing	Billy	41888.88	1324657	550000.00
2312225	300	Larkins	Lorraine	40200.00	2312225	650000.00
1333454	200	Smith	John	48000.00	1333454	550000.00
1256349	400	Harrison	Herbert	54500.00	1256349	500000.00
2341218	400	Reilly	William	36000.00	2341218	500000.00
1121334	400	Strickling	Cletus	54500.00	1121334	500000.00

Users can create complex joins in macros and then **SHARE** them with other Users.

Passing an INPUT Parameter to a Macro

```
CREATE Macro Emp4_mac  
(IN_Dept_No INTEGER) ←  
AS ←  
(SELECT *  
FROM Employee_Table  
WHERE Dept_No = :IN_Dept_No ; ) ;
```

Input Parameters
are defined before
AS Keyword

```
EXEC Emp4_mac (400); ←
```

Input Parameters
Passed to Macro
in EXEC statement

Employee_No	Dept_No	Last_Name	First_Name	Salary
1256349	400	Harrison	Herbert	54500.00
1121334	400	Strickling	Cletus	54500.00
2341218	400	Reilly	William	36000.00

The IN_Dept_No represent INPUT Parameters in the above Macro. We must place something within the parentheses or the macro will not work, because it is expecting the USER to tell it the value of IN_Dept_No in the EXEC statement..

Troubleshooting a Macro with INPUT Parameters

```
CREATE Macro Emp5_mac  
(IN_Dept_No INTEGER)  
AS  
(SELECT *  
FROM Employee_Table  
WHERE Dept_No = :IN_Dept_No ; );
```

Input Parameters

Executing the Macro

```
EXEC Emp5_mac ;
```

Where are your
Input Parameters?

Error

This Macro expected an Input Parameter and errors unless it gets it.

Troubleshooting a Macro with INPUT Parameters

```
CREATE Macro Emp5_mac  
(IN_Dept_No INTEGER)  
AS  
(SELECT *  
FROM Employee_Table  
WHERE Dept_No = :IN_Dept_No ; );
```

Input Parameters

Executing the Macro
EXEC Emp5_mac (400);

Where are your
Input Parameters?

Success!

This Macro expected an Input Parameter and errors unless it gets it.

An UPDATE Macro with Two Input Parameters

```
CREATE Macro Emp6_mac  
(IN_Employee_No INTEGER ←  
,Salary Decimal(10,2)) ←  
AS ←  
(UPDATE Employee_Table  
SET Salary = :Salary  
WHERE Employee_No = :IN_Employee_No ; );
```

Two input parameters before AS Keyword

Colons tell system that these are Input Parameters

```
EXEC Emp6_mac (2000000, 44444.44) ;
```

These are Positional Parameters

This Macro expected two parameters so we placed them in the proper order (Position). Since the CREATE statement listed both and IN_Employee_No was listed first it will be in the first position for parameters in the EXEC statement inside brackets.

Executing a Macro with Named (Not Positional) Parameters

```
CREATE Macro Emp6_mac
```

```
(IN_Employee_No INTEGER ←
```

```
,Salary Decimal(10,2)) ←
```

```
AS ←
```

```
(UPDATE Employee_Table
```

```
SET Salary = :Salary
```

```
WHERE Employee_No = :IN_Employee_No ; ) ;
```

Two input parameters before AS Keyword

1

```
EXEC Emp6_mac (Salary = 44444.44, IN_Employee_No = 2000000) ;
```

Named Parameters can be in any order

2

```
EXEC Emp6_mac (2000000, 44444.44) ;
```

Positional Parameters

Both Exec statements are the same except how they pass the Macro Input Parameters.

Troubleshooting a Macro

```
CREATE Macro AnyTable_mac  
(IN_TableName Char(32))  
AS  
(SELECT *  
FROM :IN_TableName ; ) ;
```



The Input Placement is WRONG!

Error

This Macro failed because you can't pass an input parameter to a FROM CLAUSE.

Chapter 29

Set Operators

Functions

“Good advice is something a man gives when he is too old
to set a bad example”

- Francois de la Rouchefoucauld

Table of Contents Chapter 29 - Set Operators Functions

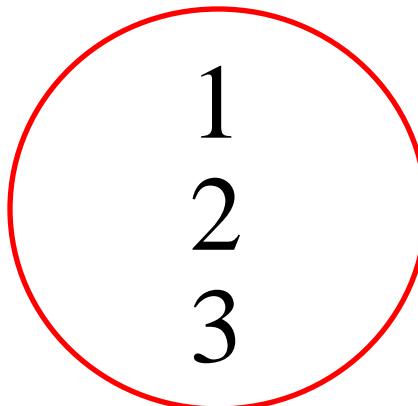
- [Rules of Set Operators](#)
- [INTERSECT Explained Logically](#)
- [INTERSECT Explained Logically](#)
- [UNION Explained Logically](#)
- [UNION Explained Logically](#)
- [UNION ALL Explained Logically](#)
- [UNION Explained Logically](#)
- [EXCEPT Explained Logically](#)
- [EXCEPT Explained Logically](#)
- [Minus Explained Logically](#)
- [Minus Explained Logically](#)
- [Testing Your Knowledge](#)
- [Testing Your Knowledge](#)
- [An Equal Amount of Columns in both SELECT List](#)
- [Columns in the SELECT list should be from the same Domain](#)
- [The Top Query handles all Aliases](#)
- [The Bottom Query does the ORDER BY \(a Number\)](#)
- [Great Trick: Place your Set Operator in a Derived Table](#)
- [UNION Vs UNION ALL](#)
- [Using UNION ALL and Literals](#)
- [A Great Example of how EXCEPT works](#)
- [USING Multiple SET Operators in a Single Request](#)
- [Changing the Order of Precedence with Parentheses](#)
- [Using UNION ALL for speed in Merging Data Sets](#)
- [Using UNION to be same as GROUP BY GROUPING SETS](#)
- [Using UNION to be same as GROUP BY ROLLUP](#)
- [Using UNION to be the same as GROUP BY Cube](#)
- [Using UNION to be same as GROUP BY Cube](#)

Rules of Set Operators

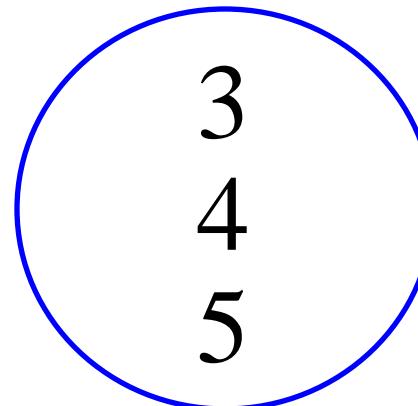
1. Each query will have two SELECT Statements separated by a SET Operator
2. SET Operators are UNION, INTERSECT, or EXCEPT/MINUS
3. Must specify the same number of columns from the same domain (data type/range)
4. If using Aggregates, both SELECTs must have their own GROUP BY
5. Both SELECTS must have a FROM Clause
6. The First SELECT is used for all ALIAS, TITLE, and FORMAT Statements
7. The Second SELECT will have the ORDER BY statement, which must be a number
8. When multiple operators the order of precedence is INTERSECT, UNION, and EXCEPT/MINUS
9. Parentheses can change the order of Precedence
10. Duplicate rows are eliminated in the spool, unless the ALL keyword is used

INTERSECT Explained Logically

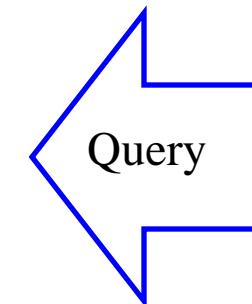
Table_Red



Table_Blue

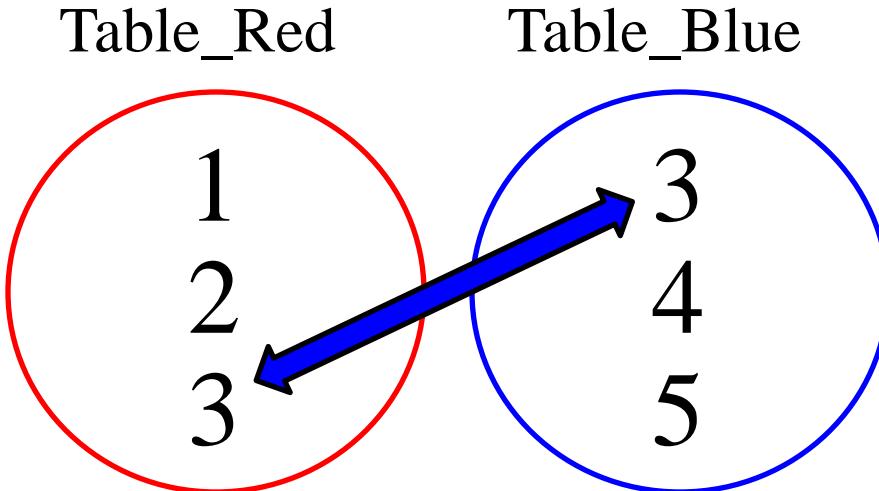


```
SELECT * FROM Table_Red  
INTERSECT  
SELECT * FROM Table_Blue ;
```

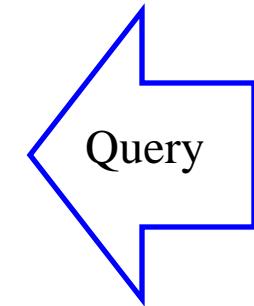


In this example, what numbers in the answer set would come from the query above ?

INTERSECT Explained Logically



```
SELECT * FROM Table_Red  
INTERSECT  
SELECT * FROM Table_Blue ;
```

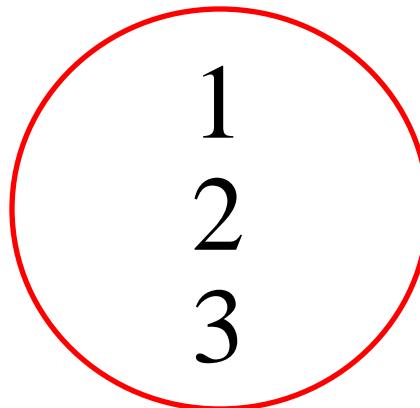


3

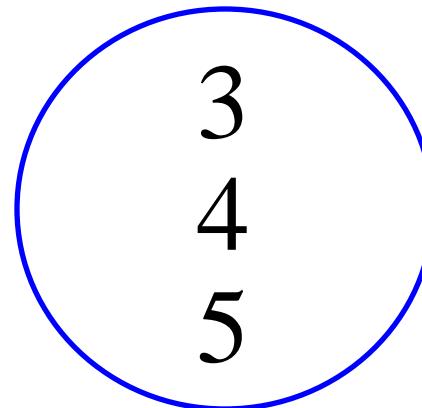
In this example, only the number 3 was in both tables so they INTERSECT.

UNION Explained Logically

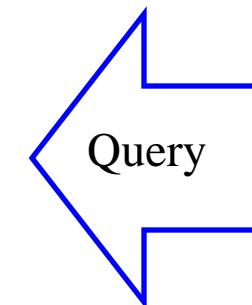
Table_Red



Table_Blue



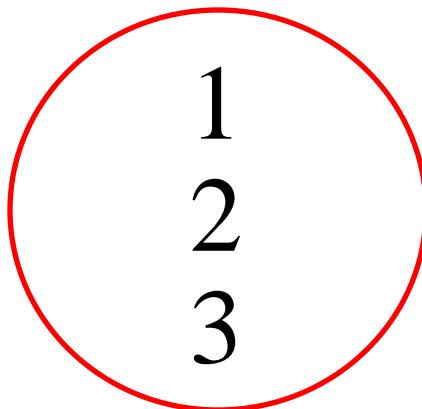
```
SELECT * FROM Table_Red  
UNION  
SELECT * FROM Table_Blue ;
```



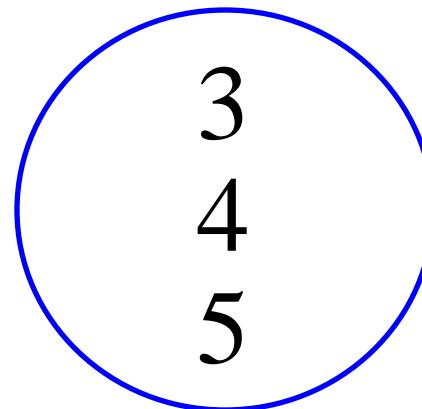
In this example, what numbers in the answer set would come from the query above ?

UNION Explained Logically

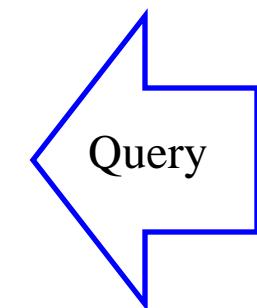
Table_Red



Table_Blue



```
SELECT * FROM Table_Red  
UNION  
SELECT * FROM Table_Blue ;
```

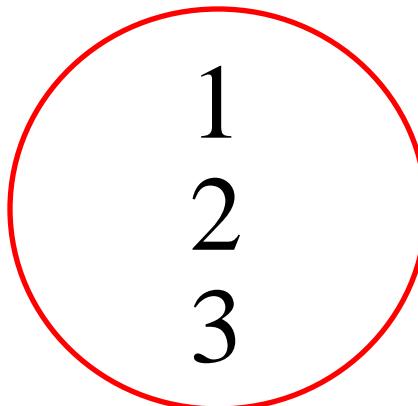


1 2 3 4 5

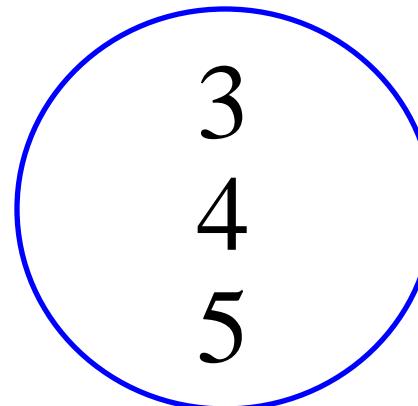
Both top and bottom queries run simultaneously, then the two different spools files are merged to eliminate duplicates and place the remaining numbers in the answer set.

UNION ALL Explained Logically

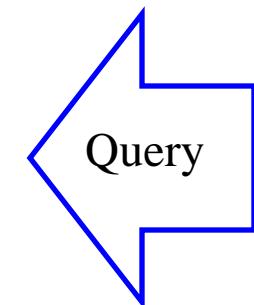
Table_Red



Table_Blue



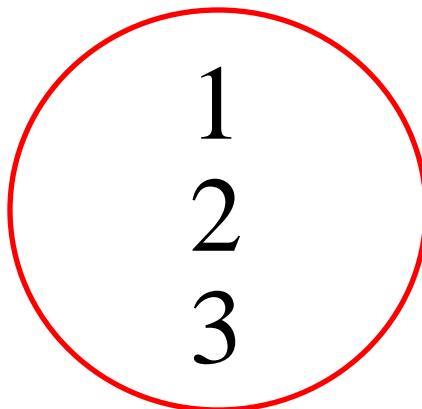
```
SELECT * FROM Table_Red  
UNION ALL  
SELECT * FROM Table_Blue ;
```



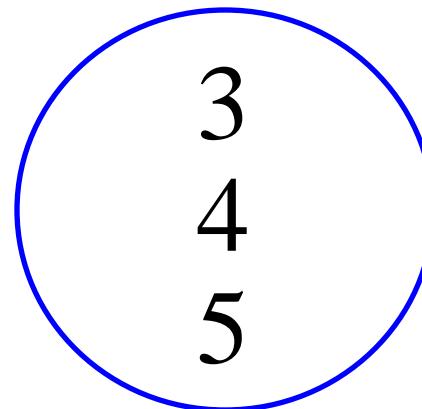
In this example, what numbers in the answer set would come from the query above ?

UNION Explained Logically

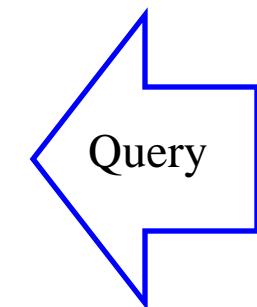
Table_Red



Table_Blue



```
SELECT * FROM Table_Red  
UNION ALL  
SELECT * FROM Table_Blue ;
```

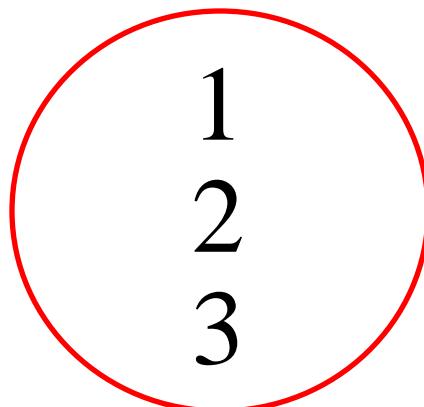


1 2 3 3 4 5

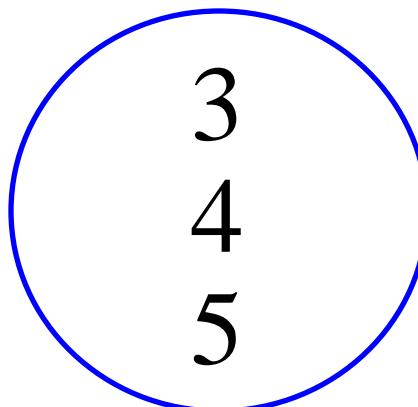
Both top and bottom queries run simultaneously, then the two different spools files are merged together to build the answer set. The **ALL** prevents eliminating **Duplicates**.

EXCEPT Explained Logically

Table_Red



Table_Blue



```
SELECT * FROM Table_Red  
EXCEPT  
SELECT * FROM Table_Blue ;
```

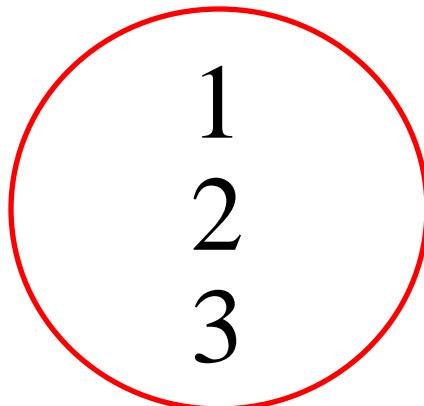
Query

EXCEPT and MINUS do the exact same thing so either word will work!

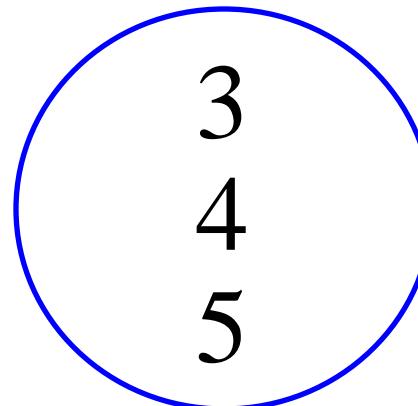
In this example, what numbers in the answer set would come from the query above ?

EXCEPT Explained Logically

Table_Red



Table_Blue



```
SELECT * FROM Table_Red  
EXCEPT  
SELECT * FROM Table_Blue ;
```

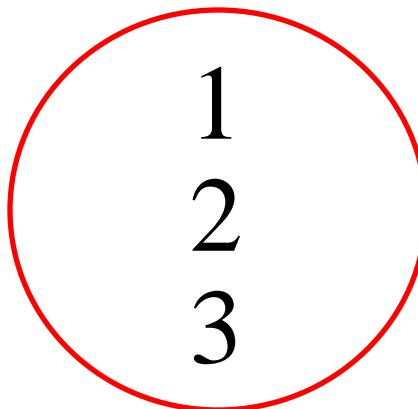
Query

1 2

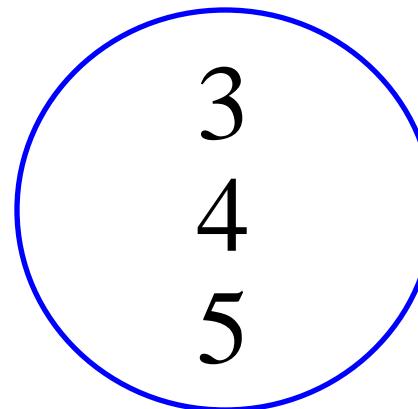
The Top query SELECTED 1, 2, 3 from Table_Red. From that point on only 1, 2, 3 at most could come back. The bottom query is run on Table_Blue and if there are any matches they are not ADDED to the 1, 2, 3 but instead take away either the 1, 2, or 3.

Minus Explained Logically

Table_Red



Table_Blue



```
SELECT * FROM Table_Blue  
MINUS  
SELECT * FROM Table_Red ;
```

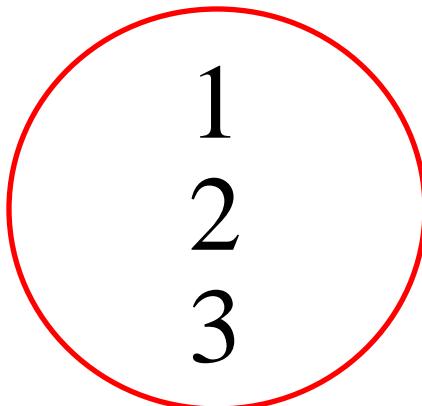
Query

EXCEPT and MINUS do the exact same thing so either word will work!

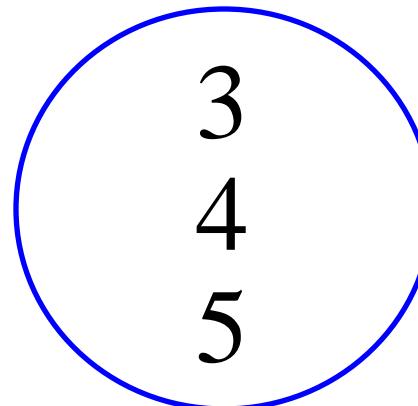
What will the answer set be? Notice I changed the order of the tables in the query!

Minus Explained Logically

Table_Red



Table_Blue



```
SELECT * FROM Table_Blue  
MINUS  
SELECT * FROM Table_Red ;
```

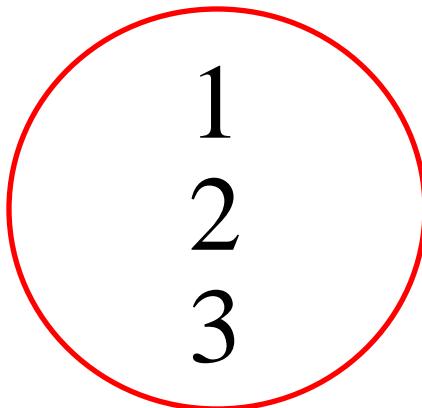
Query

4 5

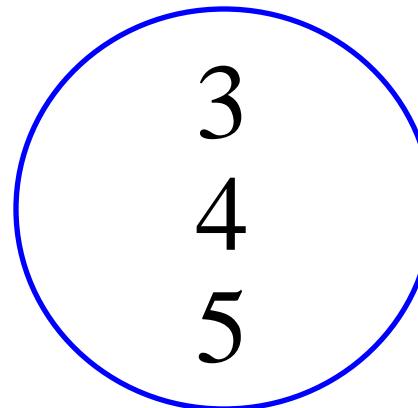
The Top query SELECTED 3, 4, 5 from Table_Blue. From that point on only 3, 4, 5 at most could come back. The bottom query is run on Table_Red and if there are any matches they are not ADDED to the 3, 4, 5 but instead take away either the 3, 4, or 5.

Testing Your Knowledge

Table_Red



Table_Blue



```
SELECT *  
FROM Table_Blue  
EXCEPT  
SELECT *  
FROM Table_Red ;
```

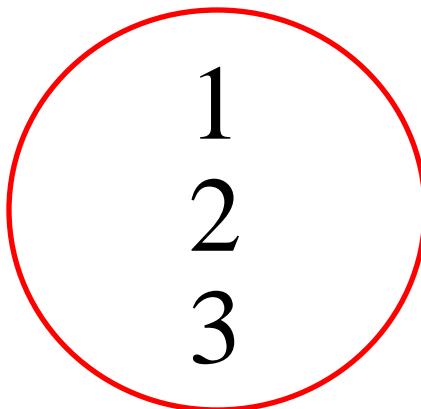
```
SELECT *  
FROM Table_Blue  
MINUS  
SELECT *  
FROM Table_Red ;
```

Will the result set be the same for both queries above?

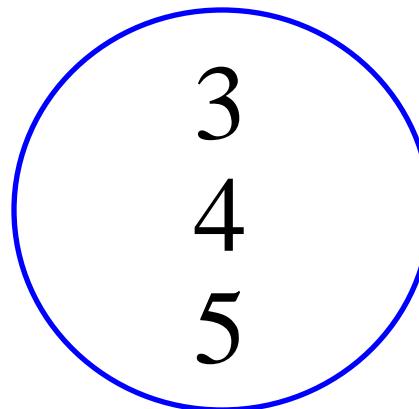
Both queries above are exactly the same to the system and produce the same result set.

Testing Your Knowledge

Table_Red



Table_Blue



```
SELECT *  
FROM Table_Blue  
EXCEPT  
SELECT *  
FROM Table_Red ;
```

```
SELECT *  
FROM Table_Red  
MINUS  
SELECT *  
FROM Table_Blue ;
```

Will the result set be the same for both queries above?

No! The first query returns 4, 5 and the query on the right returns 1, 2.

An Equal Amount of Columns in both SELECT List

```
SELECT Dept_No  
      ,Employee_No  
FROM Employee_Table  
INTERSECT  
SELECT Dept_No  
      ,Mgr_No  
FROM Department_Table;
```

Both queries have the same number of columns in the SELECT list.

Rule 1

<u>Dept_No</u>	<u>Employee_No</u>
400	1256349

Answer set

You must have an equal amount of columns in both SELECT lists. This is because data is compared from the two spool files and duplicates are eliminated. So, for comparison purposes there must be an equal amount of columns in both queries.

Columns in the SELECT list should be from the same Domain

```
SELECT First_Name  
FROM Employee_Table  
INTERSECT  
SELECT Department_Name  
FROM Department_Table;
```

You can't compare
First_Name with
Department_Name!
Different Domains!

Rule 2

First_Name

Answer
set

No rows returned

The above query works without error, but no data is returned. There are no First Names that are the same as Department Names. This is like comparing Apples to Oranges. That means they are NOT in the same Domain.

The Top Query handles all Aliases

```
SELECT Dept_No as Depty  
      ,Employee_No as "The Mgr"  
FROM Employee_Table  
INTERSECT  
SELECT Dept_No  
      ,Mgr_No  
FROM Department_Table;
```

Top query is responsible for The column ALIAS, Title and Formatting.

Rule 3

<u>Depty</u>	<u>The Mgr</u>
400	1256349

Answer set

The Top Query is responsible for ALIASING.

The Bottom Query does the ORDER BY (a Number)

```
SELECT Dept_No as Depty  
      ,Employee_No as "The Mgr"  
FROM Employee_Table  
INTERSECT  
SELECT Dept_No  
      ,Mgr_No  
FROM Department_Table  
ORDER BY 1;
```

Bottom query is responsible for the Sort with an ORDER BY

Rule 4

You must use a **number**.

ORDER BY Dept_No will error!



The Bottom Query is responsible for sorting, but the ORDER BY statement must be a number, which represents column1, column2, column3, etc.

Great Trick: Place your Set Operator in a Derived Table

```
SELECT Employee_No AS MANAGER  
      ,Trim(Last_Name) || ',' || First_Name as "Name"  
FROM   Employee_Table  
INNER JOIN  
        (SELECT Employee_No FROM Employee_Table  
         INTERSECT  
         SELECT Mgr_No      FROM Department_Table)  
          AS TeraTom (empno)  
ON Employee_No = empno  
ORDER BY "Name"
```

MANAGER	Name
1256349	Harrison, Herbert
1333454	Smith, John
1000234	Smythe, Richard
1121334	Strickling, Cletus

The Derived Table gave us the empno for all managers and we were able to join it.

UNION Vs UNION ALL

```
SELECT Department_Name, Dept_No from Department_Table  
UNION ALL
```

```
SELECT Department_Name, Dept_No from Department_Table  
ORDER BY 1;
```

UNION Answer Set

Department_Name	Dept_No
Customer Support	400
Human Resources	500
Marketing	100
Research and Development	200
Sales	300

UNION ALL Answer Set

Department_Name	Dept_No
Customer Support	400
Customer Support	400
Human Resources	500
Human Resources	500
Marketing	100
Marketing	100
Research and Development	200
Research and Development	200
Sales	300
Sales	300

UNION eliminates duplicates, but UNION ALL does not.

Using UNION ALL and Literals

```
SELECT Dept_No AS Dept  
'Employee '      (TITLE ' ')  
,First_Name || '' || Last_Name  
                           as "Name"  
FROM Employee_Table  
UNION ALL  
SELECT Dept_No  
'Department'  
,Department_Name  
FROM Department_Table  
ORDER BY 1, 2 ;
```

Dept		Name
?	Employee	Squiggy Jones
10	Employee	Richard Smythe
100	Department	Marketing
100	Employee	Mandee Chambers
200	Department	Research and Develop
200	Employee	Billy Coffing
200	Employee	John Smith
300	Department	Sales
300	Employee	Lorraine Larkins
400	Department	Customer Support
400	Employee	Cletus Strickling
400	Employee	Herbert Harrison
400	Employee	William Reilly
500	Department	Human Resources

Notice the 2nd SELECT column in that it is a literal 'Employee ' (with two spaces) and the other Literal is 'Department'. These literals match up because now they are both 10 characters long exactly. The UNION ALL brings back all Employees and all Departments and shows the employees in each valid department.

A Great Example of how EXCEPT works

```
SELECT Dept_No as Department_Number  
FROM Department_Table  
EXCEPT  
SELECT Dept_No  
FROM Employee_Table  
ORDER BY 1 ;
```

Department_Number

500

This query brought back all Departments without any employees.

USING Multiple SET Operators in a Single Request

```
SELECT Dept_No , Employee_No as empno  
FROM Employee_Table
```

UNION ALL

```
SELECT Dept_No, Employee_No  
FROM Employee_Table
```

INTERSECT ALL

```
SELECT Dept_No, Mgr_No  
FROM Department_Table
```

MINUS

```
SELECT Dept_No, Mgr_No  
FROM Department_Table  
WHERE Department_Name LIKE '%Sales%'  
ORDER BY 1, 2;
```

Dept_No	Empno
?	2000000
10	1000234
100	1232578
200	1324657
200	1333454
300	2312225
400	1121334
400	1256349
400	2341218

Above we use multiple SET Operators. They follow the natural Order of Precedence in that UNION is evaluated first, then INTERSECT, and finally MINUS.

Changing the Order of Precedence with Parentheses

```
SELECT Dept_No , Employee_No as empno  
FROM Employee_Table  
UNION ALL  
(SELECT Dept_No, Employee_No  
FROM Employee_Table  
INTERSECT ALL  
(SELECT Dept_No, Mgr_No  
FROM Department_Table  
MINUS  
SELECT Dept_No, Mgr_No  
FROM Department_Table  
WHERE Department_Name LIKE '%Sales%'))  
ORDER BY 1, 2;
```

Dept_No	Empno
?	2000000
10	1000234
100	1232578
200	1324657
200	1333454
300	2312225
400	1121334
400	1256349
400	1256349
400	2341218

Above we use multiple SET Operators and Parentheses to change the order of precedence. Above the EXCEPT runs first, then the INTERSECT and lastly, the UNION. The natural Order of Precedence without parentheses is UNION, INTERSECT, and finally EXCEPT or MINUS.

Using UNION ALL for speed in Merging Data Sets

Cust_Table_East

1,000,000 rows of
Eastern Customers

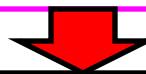
Cust_Table_West

1,000,000 rows of
Western Customers

Combined_Custs

Completely empty

```
INSERT INTO Combined_Custs
    SEL * FROM Cust_Table_East
    UNION ALL
    SEL * FROM Cust_Table_West ;
```



Combined_Custs

2,000,000 rows of
East and West Customers

Because the Combined_Custs table started empty there is no Transient Journal taking pictures for Rollback purposes so this dramatically increases the speed. This one transaction sees both SELECT statements run in parallel and then merge into one.

Using UNION to be same as GROUP BY GROUPING SETS

```
SELECT Product_ID as PROD_ID  
      ,NULL as Yr  
      ,NULL as Mth  
      ,SUM(Daily_Sales)
```

```
FROM Sales_Table  
GROUP BY 1, 2, 3
```

UNION

```
SELECT NULL
```

```
      ,EXTRACT(Year from Sale_Date)  
      ,NULL  
      ,SUM(Daily_Sales)
```

```
FROM Sales_Table  
GROUP BY 1, 2, 3
```

UNION

```
SELECT NULL
```

```
      ,NULL  
      ,EXTRACT(Month from Sale_Date)  
      ,SUM(Daily_Sales)
```

```
FROM Sales_Table  
GROUP BY 1, 2, 3
```

```
ORDER BY 1 DESC, 2, 3 ;
```

GROUP BY must be used
in all three SELECTs

Prod_ID	Yr	Mth	Sum(Daily_Sales)
3000	?	?	224587.82
2000	?	?	306611.81
1000	?	?	331204.72
?	?	9	418769.36
?	?	10	443634.99
?	2000	?	862404.35

Using UNION to be same as GROUP BY ROLLUP

```
SEL Product_ID as PROD_ID  
    ,EXTRACT(Year from Sale_Date) as Yr  
    ,EXTRACT (Month from Sale_Date) as Mth  
    ,SUM(Daily_Sales) as "Total" FROM Sales_Table  
GROUP BY 1, 2, 3  
  
UNION  
  
SEL Product_ID  
    ,EXTRACT(Year from Sale_Date)  
    ,NULL  
    ,SUM(Daily_Sales) FROM Sales_Table  
GROUP BY 1, 2, 3  
  
UNION  
  
SELECT Product_ID  
    ,NULL  
    ,NULL  
    ,SUM(Daily_Sales) FROM Sales_Table  
GROUP BY 1, 2, 3  
  
UNION  
  
SELECT NULL, NULL, NULL  
    ,SUM(Daily_Sales) FROM Sales_Table  
GROUP BY 1, 2, 3  
ORDER BY 1 DESC, 2, 3 ;
```

Prod_ID	Yr	Mth	Total
3000	?	?	224587.82
3000	2000	?	224587.82
3000	2000	9	139679.76
3000	2000	10	84908.06
2000	?	?	306611.81
2000	2000	?	306611.81
2000	2000	9	139738.91
2000	2000	10	166872.90
1000	?	?	331204.72
1000	2000	?	331204.72
1000	2000	9	139350.69
1000	2000	10	191854.03
?	?	?	862404.35

Using UNION to be the same as GROUP BY Cube

```
SEL Product_ID as PROD_ID  
    ,EXTRACT(Year from Sale_Date) as Yr  
    ,EXTRACT (Month from Sale_Date) as Mth  
    ,SUM(Daily_Sales) as "Total"  
FROM Sales_Table  
GROUP BY 1, 2, 3  
UNION  
SEL Product_ID  
    ,NULL  
    ,EXTRACT(Year from Sale_Date)  
    ,SUM(Daily_Sales) FROM Sales_Table  
GROUP BY 1, 2, 3  
UNION  
SELECT Product_ID  
    ,NULL  
    ,NULL  
    ,SUM(Daily_Sales) FROM Sales_Table  
GROUP BY 1, 2, 3  
UNION
```

```
SELECT NULL  
    ,EXTRACT(Year from Sale_Date)  
    ,EXTRACT (Month from Sale_Date)  
    ,SUM(Daily_Sales) FROM Sales_Table  
GROUP BY 1, 2, 3  
UNION  
SELECT NULL  
    ,EXTRACT(Year from Sale_Date)  
    ,NULL  
    ,SUM(Daily_Sales) FROM Sales_Table  
GROUP BY 1, 2, 3  
UNION  
SELECT NULL, NULL  
    ,EXTRACT (Month from Sale_Date)  
    ,SUM(Daily_Sales) FROM Sales_Table  
GROUP BY 1, 2, 3  
UNION  
SELECT NULL, NULL, NULL  
    ,SUM(Daily_Sales) FROM Sales_Table  
GROUP BY 1, 2, 3  
ORDER BY 1 DESC, 2 DESC, 3 DESC ;
```

CONTINUED on Right

Using UNION to be same as GROUP BY Cube

```
SEL Product_ID as PROD_ID  
    ,EXTRACT(Year from Sale_Date) as Yr  
    ,EXTRACT (Month from Sale_Date) as Mth  
    ,SUM(Daily_Sales) as "Total" FROM Sales_Table  
GROUP BY 1, 2, 3  
  
UNION  
  
SEL Product_ID  
    ,EXTRACT(Year from Sale_Date)  
    ,NULL  
    ,SUM(Daily_Sales) FROM Sales_Table  
GROUP BY 1, 2, 3  
  
UNION  
  
SELECT Product_ID  
    ,NULL  
    ,NULL  
    ,SUM(Daily_Sales) FROM Sales_Table  
GROUP BY 1, 2, 3  
  
UNION  
  
SELECT NULL, NULL, NULL  
    ,SUM(Daily_Sales) FROM Sales_Table  
GROUP BY 1, 2, 3  
ORDER BY 1 DESC, 2, 3 ;
```

Prod_ID	Yr	Mth	Total
3000	?	?	224587.82
3000	2000	?	224587.82
3000	2000	9	139679.76
3000	2000	10	84908.06
2000	?	?	306611.81
2000	2000	?	306611.81
2000	2000	9	139738.91
2000	2000	10	166872.90
1000	?	?	331204.72
1000	2000	?	331204.72
1000	2000	9	139350.69
1000	2000	10	191854.03
?	?	?	862404.35

Chapter 30

Table Create and Data Types

“Create a definite plan for carrying out your desire and begin at once, whether you are ready or not, to put this plan into action. “

-Napoleon Hill

Table of Contents Chapter 30 - Table Create and Data Types

- [Creating a Table](#)
- [Data Types](#)
- [Data Types Continued](#)
- [Data Types Continued](#)
- [Major Data Types and the number of Bytes they take up](#)
- [Making an exact copy a Table](#)
- [Making a NOT-So-Exact Copy a Table](#)
- [Copying a Table](#)
- [Troubleshooting Copying and Changing the Primary Index](#)
- [Copying only specific columns of a table](#)
- [Copying a Table and Keeping the Statistics](#)
- [Copying a Table with Statistics](#)
- [Copying a table Structure with NO Data but Statistics](#)
- [Fallback](#)

Continued on next page

Table of Contents Chapter 30 - Table Create and Data Types Continued

- [Fallback](#)
- [Before Journal](#)
- [Dual Before Journal](#)
- [After Journal](#)
- [Dual After Journal](#)
- [Journal Keyword Alone](#)
- [Why Use Journaling?](#)
- [Table Customization of the Data Block Size](#)
- [Table Customization with FREESPACE Percent](#)
- [Creating a QUEUE Table](#)
- [Example of how a Queue Table Works](#)
- [Example of how a Queue Table Works](#)

Creating a Table

```
CREATE Table Employee_Table1
(
    Employee_No      INTEGER
    ,Dept_No         INTEGER
    ,Last_Name       CHAR(20)
    ,First_Name      VARCHAR(20)
    ,Salary          DECIMAL(10,2)
    ,Hire_Date       DATE
    ,Social_Security Char(11)
)
UNIQUE PRIMARY INDEX(Employee_No);
```

UPI



This is a basic TABLE CREATE STATEMENT. You have Columns and the data types in them. As you see, the Primary Index is defined for a Table at TABLE CREATE time.

Creating a Table

```
CREATE Table Employee_Table2
(
    Employee_No          INTEGER
    ,Dept_No              INTEGER
    ,Last_Name            CHAR(20)
    ,First_Name           VARCHAR(20)
    ,Salary               DECIMAL(10,2)
    ,Hire_Date            DATE
    ,Social_Security       Char(11)
)
PRIMARY INDEX(Employee_No);
```

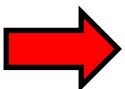
NUPI



A Non-Unique Primary Index (NUPI) is on the above table. Notice how you only need to put 'Primary Index'.

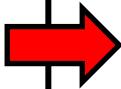
Creating a Table

NUPI



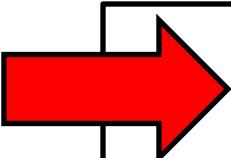
There is no Primary Index defined here

```
CREATE Table Employee_Table3
(
    Employee_No      INTEGER
    ,Dept_No         INTEGER
    ,Last_Name       CHAR(20)
    ,First_Name      VARCHAR(20)
    ,Salary          DECIMAL(10,2)
    ,Hire_Date       DATE
    ,Social_Security Char(11)
)
```



Here we have created a table without defining the Primary Index. This will **default** to the **first column** in the table here and make it a Non-Unique Primary Index (NUPI).

Creating a Table



```
CREATE SET Table Employee_Table4
(
    Employee_No      INTEGER
    ,Dept_No         INTEGER
    ,Last_Name       CHAR(20)
    ,First_Name      VARCHAR(20)
    ,Salary          DECIMAL(10,2)
    ,Hire_Date       DATE
    ,Social_Security Char(11)
)
PRIMARY INDEX(Employee_No);
```

SET TABLE means that **Duplicate ROWS** are rejected. If your system is in Teradata mode then SET tables will be the default.

Creating a Table



```
CREATE MULTISET Table Employee_Table5
(
    Employee_No      INTEGER
    ,Dept_No         INTEGER
    ,Last_Name       CHAR(20)
    ,First_Name      VARCHAR(20)
    ,Salary          DECIMAL(10,2)
    ,Hire_Date       DATE
    ,Social_Security Char(11)
)
PRIMARY INDEX(Employee_No);
```

A MULTISET Table means the table will ALLOW duplicate rows. If your system is in **ANSI** mode then **MULTISET** tables will be the default. In either Teradata mode or ANSI mode you can specifically state (SET or MULTISET) for the table type desired.

Creating a Table

```
CREATE SET Table Employee_Table6
(
    Employee_No      INTEGER
    ,Dept_No         INTEGER
    ,Last_Name       CHAR(20)
    ,First_Name      VARCHAR(20)
    ,Salary          DECIMAL(10,2)
    ,Hire_Date       DATE
    ,Social_Security Char(11)
)
UNIQUE PRIMARY INDEX(Employee_No);
```

It is very important when dealing with a SET table to have either a Unique Primary Index or a Unique Secondary Index. They eliminate the Duplicate Row Check. Because SET tables won't allow duplicate rows a "Duplicate Row Check" is done on all new INSERTS or UPDATES, but if any column has a UNIQUE constraint then the system knows that no row can be duplicate because the specific column is UNIQUE. This saves a lot of time. Do your best to stay away from the Duplicate Row Check.

Creating a Table

```
CREATE SET Table Employee_Table7  
(  
    Employee_No          INTEGER  
    ,Dept_No              INTEGER  
    ,Last_Name            CHAR(20)  
    ,First_Name           VARCHAR(20)  
    ,Salary                DECIMAL(10,2)  
    ,Hire_Date             DATE  
    ,Social_Security        Char(11)  
) Primary Index (Employee_No)  
Unique Index (Social_Security) ;
```

Unique
Secondary
Index (USI)



It is very important when dealing with a SET table to have either a Unique Primary Index or a Unique Secondary Index. They eliminate the Duplicate Row Check. Here we have created a UNIQUE Secondary Index (USI), and this will ensure no duplicate Social_Security values exist, so the system won't do the duplicate row check.

Creating a Table

```
CREATE SET Table Employee_Table8
(
    Employee_No      INTEGER
    ,Dept_No         INTEGER
    ,Last_Name       CHAR(20)
    ,First_Name      VARCHAR(20)
    ,Salary          DECIMAL(10,2)
    ,Hire_Date       DATE
    ,Social_Security Char(11)
) Unique Primary Index (Employee_No)
Unique Index (Social_Security) ;
```

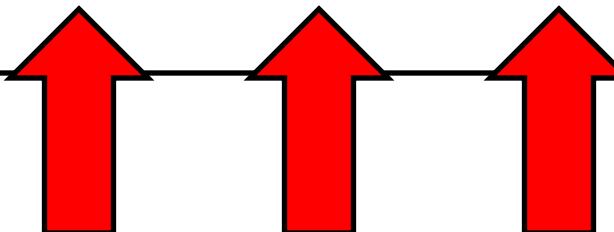
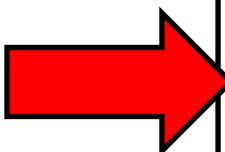
Unique
Secondary
Index (USI)



The Unique Secondary Index (USI) is an alternate path to the data. It allows for faster searches on a column, however it isn't as quick as the UPI or NUPI. When you utilize the USI column in the WHERE Clause it is always a 2-AMP operation. Fast!

Creating a Table

```
CREATE Table Employee_Table9
(
    Employee_No      INTEGER
    ,Dept_No         INTEGER
    ,Last_Name       CHAR(20)
    ,First_Name      VARCHAR(20)
    ,Salary          DECIMAL(10,2)
    ,Hire_Date       DATE
    ,Social_Security Char(11)
)
PRIMARY INDEX(Employee_No, Dept_No, Salary);
```



This is a **Multi-Column Primary Index** in this table. In this example, we have 3 Columns that make up our Primary Index.

Data Types

Data Type	Description	Data Value Range
INTEGER	Signed whole number	-2,147,483,648 to 2,147,483,647
SMALLINT	Signed smaller whole number	32,768 to 32,767
DECIMAL(X,Y) Where: X=1 thru 18, total number of digits in the number And Y=0 thru 18 digits to the right of the decimal	Signed decimal number 18 digits on either side of the decimal point	Largest value DEC(18,0) Smallest value DEC(18,18)
NUMERIC(X,Y) Same as DECIMAL	Synonym for DECIMAL	Same as DECIMAL
FLOAT REAL PRECISION DOUBLE PRECISION	Floating Point Format (IEEE)	<value>x10307 to <value>x10-308
CHARACTER(X) CHAR(X) Where: X=1 thru 64000	Fixed length character string, 1 byte of storage per character,	1 to 64,000 characters long, pads to length with space
VARCHAR(X) CHARACTER VARYING(X) CHAR VARYING(X) Where: X=1 thru 64000	Variable length character string, 1 byte of storage per character, plus 2 bytes to record length of actual data	1 to 64,000 characters as a maximum. The system only stores the characters presented to it.
CLOB (X { K M G }) CHARACTER LARGE OBJECT (X { K M G })	Large character object, for manipulating chunks. Can also specify character set and attribute.	Max for LATIN in K, M or G: 2097088000 bytes Max for UNICODE in K or M: 1048544000 bytes.
BLOB (X { K M G }) BINARY LARGE OBJECT (X { K M G })	Large binary object, for manipulating chunks.	Specified in Kilobytes, Megabytes or Gigabytes. Max is 2097088000 bytes.

Data Types Continued

Data Type	Description	Data Value Range
DATE	Signed internal representation of YYYYMMDD	See Date Chapter
TIME	Identifies a field as a TIME value with Hour, Minutes and Seconds	See Date Chapter
TIMESTAMP	Identifies a field as a TIMESTAMP value with Year, Month, Day, Hour, Minute, and Seconds	See Date Chapter
BYTEINT	Signed whole number	-128 to 127
BYTE (X) Where: X=1 thru 64000	Binary	1 to 64,000 bytes
VARBYTE (X) Where: X=1 thru 64000	Variable length binary	1 to 64,000 bytes
LONG VARCHAR	Variable length string	64,000 characters (maximum data length) The system only stores the characters provided, not trailing spaces.)
GRAPHIC (X) Where: X=1 thru 32000	Fixed length string of 16-bit bytes (2 bytes per character)	1 to 32,000 KANJI characters
VARGRAPHIC (X) Where: X=1 thru 32000	Variable length string of 16-bit bytes	1 to 32,000 characters as a maximum. The system only stores characters provided.

Data Types Continued

Data Type	Description	Data Value Range
BIGINT	Represents a signed, binary integer value	from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
PERIOD(DATE)	Two Dates that make up a Date Period	Beginning Date and Ending Date
PERIOD(TIME(n) [WITH TIME ZONE])	Two Times that make up a Time Period	Beginning Time and Ending Time
PERIOD(TIMESTAMP(n) [WITH TIME ZONE])	Two Timestamps that make up a Timestamp Period	Beginning Timestamp and Ending Timestamp

Major Data Types and the number of Bytes they take up

Bytes	Data Type	Comments
1	BYTEINT	
2	SMALLINT	
4	INTEGER	
8	BIGINT	
1	DECIMAL 1-2	
2	DECIMAL 3-4	
4	DECIMAL 5-9	
8	DECIMAL 10-18	
16	DECIMAL 19-38	
8	FLOAT	
4	DATE	
6/8	TIME	6 for 32-bit systems; 8 for 64-bit
8	TIME WITH TIME ZONE	
10/12	TIMESTAMP	10 for 32-bit systems; 12 for 64-bit
12	TIMESTAMP WITH TIME ZONE	

Making an exact copy a Table

-- This table already exists

```
CREATE SET Table Employee_Table8  
(  
    Employee_No      INTEGER  
    ,Dept_No         INTEGER  
    ,Last_Name       CHAR(20)  
    ,First_Name      VARCHAR(20)  
    ,Salary          DECIMAL(10,2)  
    ,Hire_Date       DATE  
    ,Social_Security Char(11)  
) Unique Primary Index (Employee_No)  
        Unique Index (Social_Security) ;
```

Make a Copy of the
Table with the Data.

```
CREATE Table Employee_Table10  
AS Employee_Table8 WITH DATA
```

Make a Copy of the
Table **without** the Data.

```
CREATE Table Employee_Table11  
AS Employee_Table8  
WITH NO DATA
```

When you want to make an exact copy of a table, by using the syntax at the top, it will make an **exact copy including the INDEXES!** You must include the WITH DATA or WITH NO DATA keywords, or it will error.

Making a NOT-So-Exact Copy a Table

This table already exists

```
CREATE SET Table Employee_Table8  
(  
    Employee_No      INTEGER  
    ,Dept_No         INTEGER  
    ,Last_Name       CHAR(20)  
    ,First_Name      VARCHAR(20)  
    ,Salary          DECIMAL(10,2)  
    ,Hire_Date       DATE  
    ,Social_Security Char(11)  
) Unique Primary Index (Employee_No)  
Unique Index (Social_Security) ;
```

Make a copy with some changes and keep the Data.

```
CREATE Table Employee_Table12 AS  
(SELECT * FROM Employee_Table8)  
WITH DATA  
PRIMARY INDEX (Dept_No)
```

Make a copy with some changes with **NO** Data.

```
CREATE Table Employee_Table13 AS  
(SELECT * FROM Employee_Table8)  
WITH NO DATA  
PRIMARY INDEX (Dept_No)
```

We made a copy of the table, but changed the Primary Index. The syntax above must be used to do this. You must include the WITH DATA or WITH NO DATA keywords, or it will error.

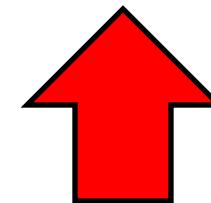
Copying a Table

-- This table already exists

```
CREATE SET Table Employee_Table8
(
    Employee_No      INTEGER
    ,Dept_No         INTEGER
    ,Last_Name       CHAR(20)
    ,First_Name      VARCHAR(20)
    ,Salary          DECIMAL(10,2)
    ,Hire_Date       DATE
    ,Social_Security Char(11)
) Unique Primary Index (Employee_No)
Unique Index (Social_Security);
```

-- Make a Copy of a Table

```
CREATE Table Employee_Table13
AS
(SELECT * FROM Employee_Table8)
WITH DATA ;
```



No Primary Index?
Make the first column
a NUPI

By using this way of getting the data, you can supply the table with a new Primary Index. In the example, they didn't take that opportunity so **the Primary Index will be the first column of the table and it will be a NUPI**. You must include the WITH DATA or WITH NO DATA keywords, or it will error.

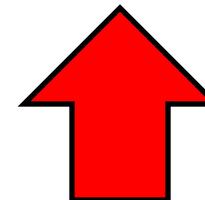
Troubleshooting Copying and Changing the Primary Index

This table already exists

```
CREATE SET Table Employee_Table8  
(  
    Employee_No      INTEGER  
    ,Dept_No         INTEGER  
    ,Last_Name       CHAR(20)  
    ,First_Name      VARCHAR(20)  
    ,Salary          DECIMAL(10,2)  
    ,Hire_Date       DATE  
    ,Social_Security Char(11)  
) Unique Primary Index (Employee_No)  
Unique Index (Social_Security) ;
```

Make a Copy of a Table

```
CREATE Table Employee_Table14  
AS Employee_Table8  
WITH DATA  
PRIMARY INDEX (Dept_No) ;
```



This isn't what
you want!

ERROR

Want a new Primary Index for your table? Well, this is **NOT** the copy syntax that will copy a table with a different Primary Index. See the previous slides.

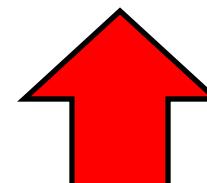
Copying only specific columns of a table

--This table already exists

```
CREATE SET Table Employee_Table8
(
    Employee_No      INTEGER
    ,Dept_No         INTEGER
    ,Last_Name       CHAR(20)
    ,First_Name      VARCHAR(20)
    ,Salary          DECIMAL(10,2)
    ,Hire_Date       DATE
    ,Social_Security Char(11)
) Unique Primary Index (Employee_No)
Unique Index (Social_Security);
```

-- Copying certain columns of a table

```
CREATE TABLE Employee_Table16
AS
(Select Employee_No
     ,Dept_No
     ,Last_Name
     ,First_Name
  FROM Employee_Table8)
WITH DATA
PRIMARY INDEX (Dept_No);
```



This copy statement will not error and actually copy only the first four columns and their data. It will also change the Primary Index to a NUPI on Dept_No.

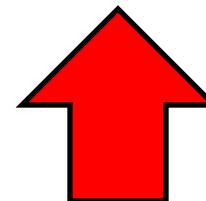
Copying a Table and Keeping the Statistics

--This table already exists

```
CREATE SET Table Employee_Table8  
(  
    Employee_No      INTEGER  
    ,Dept_No         INTEGER  
    ,Last_Name       CHAR(20)  
    ,First_Name      VARCHAR(20)  
    ,Salary          DECIMAL(10,2)  
    ,Hire_Date       DATE  
    ,Social_Security Char(11)  
) Unique Primary Index (Employee_No)  
        Unique Index (Social_Security) ;
```

--Make a Copy of a Table

```
CREATE TABLE Employee_Table16  
AS Employee_Table8 with DATA  
AND Statistics;
```



This copy statement will not error and will actually copy the table structure, the data, and have the exact same statistics.

Copying a Table with Statistics

--This Table already exists

```
CREATE SET Table Employee_Table  
(  
    Employee_No      INTEGER  
    ,Dept_No         INTEGER  
    ,Last_Name       CHAR(20)  
    ,First_Name      VARCHAR(20)  
    ,Salary          DECIMAL(10,2)  
    ,Hire_Date       DATE  
    ,Social_Security Char(11)  
) Unique Primary Index (Employee_No)  
        Unique Index (Social_Security) ;
```

--Make a Copy of a Table

```
CREATE TABLE Employee_Table7  
AS Employee_Table with NO DATA  
AND Statistics;
```



This copy statement will not error . It will actually copy the table structure with no data, but the **Statistics** will all be **ZEROED**.

Copying a table Structure with NO Data but Statistics

-- This table already exists

```
CREATE SET Table Employee_Table2  
(  
    Employee_No      INTEGER  
    ,Dept_No         INTEGER  
    ,Last_Name       CHAR(20)  
    ,First_Name      VARCHAR(20)  
    ,Salary          DECIMAL(10,2)  
    ,Hire_Date       DATE  
    ,Social_Security Char(11)  
) Unique Primary Index (Employee_No)  
Unique Index (Social_Security) ;
```

-- Make a Copy of a Table

```
CREATE TABLE Employee_Table7  
AS Employee_Table2 with NO DATA  
AND Statistics;
```



-- Once the Employee_Table7 is loaded
Recollect Statistics on all statistics
collected before by Employee_Table2

```
COLLECT STATISTICS  
ON Employee_Table7 ;
```

Once the Employee_Table7 is loaded with data, the user can Recollect Statistics. What is clever here is that originally the statistics on the new table are zeroed, which means they have the columns and indexes listed that were previously collected by the old table, but now actual statistic data. Once the new table is loaded and the COLLECT STATISTICS command is run again it will automatically update the true statistics of the new table, but only collect on the columns or indexes the old table had done in the past.

Fallback

```
CREATE SET Table Employee_Table, FALLBACK
(
    Employee_No          INTEGER
    ,Dept_No              INTEGER
    ,Last_Name            CHAR(20)
    ,First_Name           VARCHAR(20)
    ,Salary               DECIMAL(10,2)
    ,Hire_Date            DATE
    ,Social_Security       Char(11)
) Unique Primary Index (Employee_No)
Unique Index (Social_Security) ;
```

Fallback means a Duplicate copy of each row is stored on a different AMP. This effectively doubles the size of the table, but all is good if an AMP goes down. Fallback rows are stored on the AMPS, but Teradata never accesses the Fallback rows for a query unless the Primary data is inaccessible due to a hardware or software failure.

Fallback

```
CREATE SET Table Employee_Table, NO FALBACK
(
    Employee_No          INTEGER
    ,Dept_No              INTEGER
    ,Last_Name            CHAR(20)
    ,First_Name           VARCHAR(20)
    ,Salary               DECIMAL(10,2)
    ,Hire_Date            DATE
    ,Social_Security       Char(11)
) Unique Primary Index (Employee_No)
Unique Index (Social_Security) ;
```

Default
for
now

NO Fallback is the default. However, you should specify because FALBACK can be the default at the database level.

Before Journal

```
CREATE SET Table Employee_Table, NO FALBACK,  
Before Journal  
(  
    Employee_No          INTEGER  
    ,Dept_No              INTEGER  
    ,Last_Name            CHAR(20)  
    ,First_Name           VARCHAR(20)  
    ,Salary                DECIMAL(10,2)  
    ,Hire_Date             DATE  
    ,Social_Security        Char(11)  
 ) Unique Primary Index (Employee_No)  
 Unique Index (Social_Security) ;
```



Journaling helps with roll backing data if there is an error. A BEFORE Journal is a BEFORE Image of a row that will be stored for any new/changing row.

Dual Before Journal

```
CREATE SET Table Employee_Table, NO FALBACK,  
DUAL Before Journal
```



```
(  
Employee_No      INTEGER  
,Dept_No        INTEGER  
,Last_Name       CHAR(20)  
,First_Name      VARCHAR(20)  
,Salary          DECIMAL(10,2)  
,Hire_Date       DATE  
,Social_Security Char(11)  
) Unique Primary Index (Employee_No)  
Unique Index (Social_Security) ;
```

A DUAL BEFORE Journal takes Two BEFORE IMAGES of a row and these two copies are stored on two separate AMPS whenever any new/changing row occurs.

After Journal

```
CREATE SET Table Employee_Table, NO FALBACK,  
AFTER JOURNAL  
(  
    Employee_No          INTEGER  
    ,Dept_No              INTEGER  
    ,Last_Name            CHAR(20)  
    ,First_Name           VARCHAR(20)  
    ,Salary                DECIMAL(10,2)  
    ,Hire_Date             DATE  
    ,Social_Security        Char(11)  
 ) Unique Primary Index (Employee_No)  
 Unique Index (Social_Security) ;
```



An AFTER Journal is an AFTER Image of a row that will be stored for any new/changing row.

Dual After Journal

```
CREATE SET Table Employee_Table, NO FALBACK,  
DUEL AFTER JOURNAL ←  
(  
    Employee_No      INTEGER  
    ,Dept_No         INTEGER  
    ,Last_Name       CHAR(20)  
    ,First_Name      VARCHAR(20)  
    ,Salary          DECIMAL(10,2)  
    ,Hire_Date       DATE  
    ,Social_Security Char(11)  
 ) Unique Primary Index (Employee_No)  
     Unique Index (Social_Security) ;
```

A DUAL AFTER Journal is two AFTER IMAGES of a row that are stored for any new/changing row

Journal Keyword Alone



```
CREATE SET Table Employee_Table, NO FALBACK,  
JOURNAL  
(  
    Employee_No      INTEGER  
    ,Dept_No         INTEGER  
    ,Last_Name       CHAR(20)  
    ,First_Name      VARCHAR(20)  
    ,Salary          DECIMAL(10,2)  
    ,Hire_Date       DATE  
    ,Social_Security Char(11)  
 ) Unique Primary Index (Employee_No)  
 Unique Index (Social_Security) ;
```

A JOURNAL is a BEFORE and AFTER image that will be stored for any new/changing row

Why Use Journaling?



```
CREATE SET Table Employee_Table, NO FALBACK,  
BEFORE JOURNAL, AFTER JOURNAL
```

```
(  
    Employee_No          INTEGER  
    ,Dept_No              INTEGER  
    ,Last_Name            CHAR(20)  
    ,First_Name           VARCHAR(20)  
    ,Salary                DECIMAL(10,2)  
    ,Hire_Date             DATE  
    ,Social_Security        Char(11)  
) Unique Primary Index (Employee_No)  
      Unique Index (Social_Security) ;
```

A BEFORE Journal would be utilized to Rollback a programming error to the way things looked BEFORE (at a specific Point-In-Time) .

Why Use Journaling?

```
CREATE SET Table Employee_Table, NO FALBACK,  
BEFORE JOURNAL, AFTER JOURNAL ←
```

```
(  
    Employee_No      INTEGER  
    ,Dept_No         INTEGER  
    ,Last_Name       CHAR(20)  
    ,First_Name      VARCHAR(20)  
    ,Salary          DECIMAL(10,2)  
    ,Hire_Date       DATE  
    ,Social_Security Char(11)  
) Unique Primary Index (Employee_No)  
        Unique Index (Social_Security) ;
```

An AFTER Journal would be part of the Full System Backup strategy to recover how things looked AFTER (to a specific Point-In-Time). The above example uses both a BEFORE and AFTER Journal.

Table Customization of the Data Block Size

```
CREATE SET Table Employee_Table, Fallback,  
  DATABLOCKSIZE=16384 BYTES,  
  FREESPACE = 20 PERCENT  
(  
  Employee_No      INTEGER  
  ,Dept_No         INTEGER  
  ,Last_Name       CHAR(20)  
  ,First_Name      VARCHAR(20)  
  ,Salary          DECIMAL(10,2)  
  ,Hire_Date       DATE  
  ,Social_Security Char(11)  
 ) Unique Primary Index (Employee_No) ;
```

A user sometimes does not want to use the DEFAULT DATABLOCKSIZE. With this command, it allows them to alter the DATABLOCKSIZE. An OLTP Table is better tuned for smaller BLOCK SIZES.

Table Customization with FREESPACE Percent

```
CREATE SET Table Employee_Table, FALBACK,
DATABLOCKSIZE=16384 BYTES,
FREESPACE = 20 PERCENT
(
    Employee_No          INTEGER
    ,Dept_No              INTEGER
    ,Last_Name            CHAR(20)
    ,First_Name           VARCHAR(20)
    ,Salary               DECIMAL(10,2)
    ,Hire_Date            DATE
    ,Social_Security      Char(11)
) Unique Primary Index (Employee_No);
```

FREESPACE of 20 PERCENT means when using MULTILOAD or FASTLOAD, the table will only take up **80%** of the Cylinder leaving **20%** as free space.

Creating a QUEUE Table

```
CREATE SET Table Retail_Sales, QUEUE
(
    Ret_Qits  TIMESTAMP (6) NOT NULL DEFAULT CURRENT_TIMESTAMP(6)
    ,Product_ID  INTEGER
    ,Quantity  INTEGER
)
PRIMARY INDEX(Product_ID);
```

The first column of a QUEUE Table is **ALWAYS** a Queue Insertion Timestamp Column (QITS) defined as above. The first column **CANNOT** be defined as UNIQUE, an IDENTITY Column, or a UNIQUE SECONDARY INDEX OR PRIMARY KEY. The FIFO (First In First Out) is the principle is a Queue table built around. As rows come into the table they will be processes with the CONSUME statement on a first in first out basis. The major significance of having a Queue Table is Rows retrieved are processed only once and then deleted.

Example of how a Queue Table Works

```
CREATE SET Table Retail_Sales, QUEUE  
(  
    Ret_Qits  TIMESTAMP (6) NOT NULL  DEFAULT CURRENT_TIMESTAMP(6)  
    ,Product_ID  INTEGER  
    ,Quantity  INTEGER  
) PRIMARY INDEX(Product_ID);
```

1

2

```
INSERT INTO Retail_Sales  
(Current_Timestamp, 1000, 100);
```

3

```
SELECT * FROM Retail_Sales;
```

Ret_Qits

04/10/2011 10:22:23:19

Product ID

Quantity

1000 100

This row will **NOT** automatically be DELETED because of our SELECT.

Example of how a Queue Table Works

```
CREATE SET Table Retail_Sales, QUEUE  
(  
    Ret_Qits  TIMESTAMP (6) NOT NULL  DEFAULT CURRENT_TIMESTAMP(6)  
    ,Product_ID  INTEGER  
    ,Quantity  INTEGER  
) PRIMARY INDEX(Product_ID);
```

1

2

```
INSERT INTO Retail_Sales  
(Current_Timestamp, 1000, 100);
```

3

```
SELECT and CONSUME TOP 1 * from Retail_Sales;
```

Ret_Qits

04/10/2011 10:22:23:19

Product_ID

1000

Quantity

100

This row will automatically be **DELETED** because of our **SELECT and CONSUME TOP 1** Statement.

Chapter 31

Data Manipulation Language (DML)

“I tried to draw people more realistically, but the figure I neglected to update was myself.”

- Joe Sacco

Table of Contents Chapter 31 - Data Manipulation Language (DML)

- [INSERT Syntax # 1](#)
- [INSERT Example with Syntax 1](#)
- [INSERT Syntax # 2](#)
- [INSERT Example with Syntax 2](#)
- [INSERT Example with Syntax 3](#)
- [Using NULL for Default Values](#)
- [INSERT/SELECT Command](#)
- [INSERT/SELECT Example using All Columns \(*\)](#)
- [INSERT/SELECT Example with Less Columns](#)
- [INSERT/SELECT to Build a Data Mart](#)
- [Fast Path INSERT/SELECT](#)
- [NOT quite the Fast Path INSERT/SELECT](#)
- [UNION for the Fast Path INSERT/SELECT](#)
- [BTEQ for the Fast Path INSERT/SELECT](#)
- [The UPDATE Command Basic Syntax](#)
- [Two UPDATE Examples](#)
- [Subquery UPDATE Command Syntax](#)
- [Example of Subquery UPDATE Command](#)
- [Join UPDATE Command Syntax](#)
- [Example of an UPDATE Join Command](#)
- [Fast Path UPDATE](#)
- [The DELETE Command Basic Syntax](#)
- [Two DELETE Examples to DELETE ALL Rows in a Table](#)

Continued on next page

Table of Contents Chapter 31 - Data Manipulation Language (DML) Continued

- [A DELETE Example Deleting only Some of the Rows](#)
- [Subquery and Join DELETE Command Syntax](#)
- [Example of Subquery DELETE Command](#)
- [Example of Join DELETE Command](#)
- [Fast Path DELETE](#)
- [Fast Path DELETE Example # 1](#)
- [Fast Path DELETE Example # 2](#)
- [Fast Path DELETE Example # 3](#)
- [MERGE INTO](#)
- [MERGE INTO Example that Matches](#)
- [MERGE INTO Example that does NOT Match](#)
- [OReplace](#)

INSERT Syntax # 1

The following syntax of the INSERT does not use the column names as part of the command. Therefore, it requires that the VALUES portion of the INSERT match each and every column in the table with a data value or a NULL.

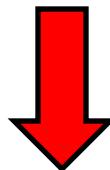
```
INS[ERT] [ INTO ] <table-name> VALUES  
( <literal-data-value> [ ...,<literal-data-value> ] ;
```

Note: Using INS instead of INSERT is not ANSI compliant.

The INSERT statement is used to put a new row into a table. A status is the only returned value from the database; no rows are returned to the user. It must account for all the columns in a table using either a data value or a NULL. When executed, the INSERT places a single new row into a table. Although it can run as a single row insert, primarily it is used in utilities like BTEQ, FastLoad, MultiLoad, TPump or some other application that reads a data record and uses the data to build a new row in a table.

INSERT Example with Syntax 1

```
INSERT INTO My_Table VALUES  
( 'My character data', 124.56, 102587, , NULL, '2000-12-31' );
```



My character data	124.56	102587	NULL	NULL	2000-12-31
-------------------	--------	--------	------	------	------------

After the execution of the above INSERT, there is a new row with the first character data value of 'My character data' going into Column1, the decimal value of 124.56 into Column2, the integer 102587 into Column3, NULL values into Column4 and Column5, and a date into Column6.

The NULL expressed in the VALUES list is the literal representation for no data. However, the two commas (,,) that follow the positional value for Column3 also represent missing data. The commas are placeholders or delimiters for the data values. When no data value is coded, the end result is a NULL.

INSERT Syntax # 2

The syntax of the second type of INSERT follows:

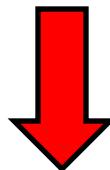
```
INS[ERT] [ INTO ] <table-name>
( <column-name> [,...<column-name> ]
VALUES
( <literal-data-value> [,...<literal-data-value> ] ;
```

Note: Using INS instead of INSERT is not ANSI compliant.

This is another form of the INSERT statement that can be used when some of the data is not available. It allows for the missing values (NULL) to be eliminated from the list in the VALUES clause. It is also the best format when the data is arranged in a different sequence than the CREATE TABLE, or when there are more nulls (unknown values) than available data values.

INSERT Example with Syntax 2

```
INSERT INTO My_Table (Column2, Column1, Column3, Column6)  
VALUES (124.56, 'My character data', 12587, '2000-12-31');
```

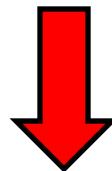


My character data	124.56	102587	NULL	NULL	2000-12-31
-------------------	--------	--------	------	------	------------

The above statement incorporates both of the reasons to use this syntax. First, notice that the column names Column2 and Column1 have been switched, to match the data values. Also, notice that Column4 and Column5 do not appear in the column list, therefore they are assumed to be NULL. This is a good format to use when the data is coming from a file and does not match the order of the table columns.

INSERT Example with Syntax 3

```
INSERT INTO My_Table  
(Column2=124.56, Column1='My character data', Column3=12587,  
Column6='2000-12-31') ;
```



My character data	124.56	102587	NULL	NULL	2000-12-31
-------------------	--------	--------	------	------	------------

The third form of the INSERT statement can be used to insert the same row as the previous INSERT. It might look like this:

Using NULL for Default Values

Either of the next two INSERT statements may be used to build a row with no data values in My_Table:

```
INSERT INTO My_Table VALUES (, , , , , );
```

```
INSERT INTO My_Table VALUES  
(NULL,NULL,NULL,NULL,NULL,NULL) ;
```



NULL	NULL	NULL	NULL	NULL	NULL
------	------	------	------	------	------

Teradata now has the ANSI DEFAULT VALUES functionality. Although an INSERT statement could easily put a null value into a table column, it requires it to use the NULL reserved word or by omitting a value for that column(s) between commas.

INSERT/SELECT Command

The syntax of the INSERT / SELECT is:

```
INS[ERT] [ INTO ] <table-name>
    SELECT <column-name> [...,<column-name> ]
    FROM <table-name> ;
```

Although the INSERT is great for adding a single row not currently present in the system, an INSERT/SELECT is even better when the data already exists within Teradata. In this case, the INSERT is combined with a SELECT. However, no rows are returned to the user. Instead, they go into the table as new rows.

The SELECT reads the data values from the one or more columns in one or more tables and uses them as the values to INSERT into another table. Simply put, the SELECT takes the place of the VALUES portion of the INSERT.

This is a common technique for building data marts, interim tables and temporary tables. It is normally a better and much faster alternative than extracting the rows to a data file, then reading the data file and inserting the rows using a utility.

INSERT/SELECT Example using All Columns (*)

When all columns are desired to make an exact copy of the second table and both tables have the exact same number of columns in the exact same order with the exact same data types; an * may be used in the SELECT to read all columns without a WHERE clause, as in the example below:

```
INSERT INTO My_Table  
SELECT * FROM My_Original_Table;
```

Like all SELECT operations without a WHERE clause, a full table scan occurs and all the rows of the second table are inserted into My_Table, using only the data values from the columns listed.

INSERT/SELECT Example with Less Columns

When fewer than all the columns are desired, either of the following INSERT / SELECT statements will do the job:

```
INSERT INTO My_Table  
    SELECT ( Column1, Column2, Column3 , , , '2010-01-01' )  
        FROM My_Original_Table ;
```

```
INSERT INTO My_Table ( Column2, Column1, Column3, Column6)  
    SELECT Column2, Column1, Column3 , CURRENT_DATE  
        FROM My_Original_Table ;
```

In both of the above examples, only the first three and the last columns are receiving data. In the first INSERT, the data is a literal date. The second INSERT uses the CURRENT_DATE. Both are acceptable, depending on what is needed.

Working with the same concept of a normal INSERT, when using the column names the only data values needed are for these columns and they must be in the same sequence as the column list, not the CREATE TABLE. Therefore, omitted data values or column names become a NULL data value.

INSERT/SELECT to Build a Data Mart

As an example of a data mart, it might be desirable to build a summary table using something like the following:

```
INSERT INTO My_Table
    SELECT ( Column1, SUM(Column2), AVG(Column3),
              COUNT(Column4), AVG(CHAR(Column5)),
              AVG(CHAR(Column6)) )
    FROM My_Original_Table
    GROUP BY 1 ;
```

However used, the INSERT / SELECT is a powerful tool for creating rows from the rows already contained in one or more other tables.

Fast Path INSERT/SELECT

```
INSERT INTO My_Table SELECT * FROM My_Original_Table ;
```

When the table being loaded is **empty**, the INSERT / SELECT is very fast. This is especially true when all columns and all rows are being copied. Remember, the table being loaded must be empty to attain the speed. If there is even one row already in the table, it negates the ability to take the Fast Path.

There are two reasons behind this speed. First, there is no need to Transient Journal an identifier for each inserted row. Recovery, if needed, is to empty the table. No other type of recovery can be easier or faster.

Second, when all columns and all rows are requested from the existing table and they exactly match the columns in the new table, there is no need to use spool. The rows go straight into the table being loaded. Additionally, when all rows are being selected Teradata does not bother to read the individual rows. Instead, each AMP literally copies the blocks of the original table to blocks for the new table.

These reasons are why it is called the Fast Path. To use this technique, the order of the columns in both tables must match exactly and so must the data types, otherwise spool must be used to rearrange the data values or translate from one data type to the other.

NOT quite the Fast Path INSERT/SELECT

What if it is necessary to retrieve the rows from multiple tables for the INSERT?

Multiple INSERT / SELECT operations could be performed as follows:

```
INSERT INTO My_Table      SELECT * FROM My_Original_Table_1 ;  
INSERT INTO My_Table      SELECT * FROM My_Original_Table_2 ;  
INSERT INTO My_Table      SELECT * FROM My_Original_Table_3 ;
```

The first INSERT/SELECT into My_Table loads the empty table extremely fast, even with millions of rows. However, the table is no longer empty and the subsequent INSERT is much slower because it cannot use the fast path. All inserted rows must be identified in the Transient Journal. It can more than double the processing time.

The real question is: How to make all of the individual SELECT operations act as one so that the table stays empty until all rows are available for the INSERT?

UNION for the Fast Path INSERT/SELECT

One way get the Fast Path is to use the UNION command to perform all SELECT operations in parallel before the first row is inserted into the new table. Therefore all rows are read from the various tables, combined into a single answer set in spool and then loaded into the empty table. All of this is done at high speed.

For instance, if all the rows from three different tables are needed to populate the new table, the applicable statement might look like the following:

```
INSERT INTO My_Table
    SELECT * FROM My_Original_Table_1
    UNION
    SELECT * FROM My_Original_Table_2
    UNION
    SELECT * FROM My_Original_Table_3 ;
```

Again, the above statement assumes that all four tables have exactly the same columns. Whether or not that would ever be the case in real life, this is used as an example. However, at this point we know the columns in the SELECT must match the columns in the table to be loaded, no matter how that is accomplished.

BTEQ for the Fast Path INSERT/SELECT

A second alternative method is available using BTEQ. The key here is that BTEQ can do multiple SQL statements as a single transaction for the SELECT and the INSERT operations. The only way to do that is to delay the actual INSERT, until all of the rows from all the select operations have completed. Then the INSERT is performed as a part of the same transaction into the empty table.

```
.logon localtd/dbc
```

```
Password: *****
```

```
Logon successfully completed
```

```
BTEQ – Enter your DBC/SQL request or BTEQ command:
```

```
INSERT INTO My_Table
```

```
    SELECT * FROM My_Original_Table_1
```

```
; INSERT INTO My_Table
```

```
    SELECT * FROM My_Original_Table_2
```

```
; INSERT INTO My_Table
```

```
    SELECT * FROM My_Original_Table_3 ;
```

By having another SQL command on the same line as the semi-colon (;), in BTEQ, they all become part of the same multi-statement transaction. Therefore, all are inserting into an empty table and it is much faster than doing each INSERT individually.

The UPDATE Command Basic Syntax

```
UPD[ATE] <table-name>
[ FROM <table-name> [AS <alias-name>] ]
    SET <column-name> = { <expression-or-data-value> | <data-value> }
        [..., <column-name> = <expression-or-data-value> | <data-value> ]
    [ WHERE <condition-test> ]
        [ AND <condition-test> ... ] [ OR <condition-test> ... ] [ALL] ;
```

The UPDATE statement is used to modify data values in one or more columns of one or more existing rows. A status is the only returned value from the database; no rows are returned to the user.

When business requirements call for a change to be made in the existing data, then the UPDATE is the SQL statement to use. In order for the UPDATE to work, it must know a few things about the data row(s) involved. Like all SQL, it must know which table to use for making the change, which column or columns to change and the change to make within the data.

Two UPDATE Examples

```
UPDATE My_Table  
    SET Column2 = 256  
        , Column4 = 'Mine'  
        , Column5 = 'Yours'  
    WHERE Column1 = 'My character data';
```

```
UPDATE My_Table  
    SET Column2 = Column2 + 256  
    WHERE Column1 = 'My character data'  
        AND Column4 = 'Mine'  
        AND Column5 = 'Yours';
```

The first UPDATE command modifies all rows that contain ‘My character data’ including the one that was inserted earlier in this chapter. It changes the values in three columns with new data values provided after the equal sign (=):

The next UPDATE uses the same table as the above statement. However, this time it modifies the value in a column based on its current value and adds 256 to it. The UPDATE determines which row(s) to modify with compound conditions written in the WHERE clause based on values stored in other columns:

Subquery UPDATE Command Syntax

```
UPD[ATE] <table-name>
[ FROM <table-name> [AS <alias-name>] ]
    SET <column-name> = { <expression-or-data-value> | <data-value> }
        [..., <column-name> = <expression-or-data-value> | <data-value> ]
WHERE <column-name> [..., <column-name>]
    IN ( SELECT <column-name> [...,<column-name>]
        FROM <table-name> [ AS <alias-name> ]
        [ WHERE <condition-test> ... ] ) [ ALL ];
;
```

Sometimes it is necessary to update rows in a table when they match rows in another table. To accomplish this, the tables must have one or more columns in the same domain. The matching process then involves either a subquery or join processing.

Notice in the above that creating an alias for the table being updated is not compatible with a FROM clause. This change was made in V2R4.

Example of Subquery UPDATE Command

To change rows in My_Table using another table called Ctl_Tbl, the following UPDATE uses a subquery operation to accomplish the operation:

```
UPDATE My_Table  
    SET Column3 = 20000000  
WHERE Column2 IN ( SELECT Column2 FROM Ctl_Tbl  
                      WHERE Column3 > 5000  
                      AND ctl_tbl.Column4 IS NOT NULL );
```

Sometimes it is necessary to update rows in a table when they match rows in another table. To accomplish this, the tables must have one or more columns in the same domain. The matching process then involves either a subquery or join processing.

Notice in the above that creating an alias for the table being updated is not compatible with a FROM clause. This change was made in V2R4.

Join UPDATE Command Syntax

```
UPD[ATE] <table-name>
  [ FROM <table-name> [ AS <alias-name> ] ]
    SET <column-name> = { <expression-or-data-value> | <data-value> }
      [..., <column-name> = <expression-or-data-value> | <data-value> ]
WHERE [<table-name>.]<column-name> = [<table-name>.]<column-name>
      [ AND <condition-test> ] [ OR <condition-test> ] [ ALL ] ;
```

When adding an alias to the UPDATE, the alias becomes the table name and MUST be used in the WHERE clause when qualifying columns.

Example of an UPDATE Join Command

To change rows in My_Table using another table called Ctl_Tbl the following UPDATE uses a join to accomplish the operation: Both examples are equivalent.

```
UPDATE My_Table  
  FROM Ctl_Tbl AS Ctbl  
    SET Column3 = 20000000  
      ,Column5 = 'A'  
 WHERE My_Table.Column2 = Ctbl.Column2  
 AND My_Table.Column3 > 5000  AND Ctl_Tbl.Column4 IS NOT NULL );
```

```
UPDATE My_Table  
    SET Column3 = 20000000  
      , Column5 = 'A'  
 WHERE mytable.Column2 = Ctl_tbl.Column2  
 AND mytable.Column3 > 5000 AND Ctl_tbl.Column4 IS NOT NULL );
```

In reality, the FROM is optional. This is because Teradata can dynamically include a table by qualifying the join column with the table name. The FROM is only needed to make an alias for the join tables. The second UPDATE is the same as the above without the FROM for Ctl_Tbl:

Fast Path UPDATE

The following INSERT/SELECT “updates” the values in Column3 and Column5 in every row of My_Table, using My_Table_Copy:

```
INSERT INTO My_Table_Copy
    SELECT  Column1
            ,Column2
            ,Column3*1.05
            ,Column4
            ,’A’
            ,Column6
    FROM My_Table ;
```

When the above command finishes, My_Table_Copy contains every row from My_Table with the needed update.

The DELETE Command Basic Syntax

```
DEL[ETE] [ FROM ] <table-name> [ AS <alias-name> ]  
[ WHERE condition ] [ ALL ] ;
```

The DELETE statement has one function and that is to remove rows from a table. A status is the only returned value from the database; no rows are returned to the user. One of the fastest things that Teradata does is to remove ALL rows from a table.

The reason for its speed is that it simply moves all of the sectors allocated to the table onto the free sector list in the AMP's Cylinder Index. It is the fast path and there is no OOPS command, unless the explicit transaction has not yet completed. In that case, a ROLLBACK statement can be issued to undo the delete operation before a COMMIT. Otherwise, the rows are gone and it will take either a backup tape or a BEFORE image in the Permanent Journal to perform a manual rollback. Be Very CAREFUL with DELETE. It can come back to bite you if your not careful.

Two DELETE Examples to DELETE ALL Rows in a Table

```
DELETE FROM My_Table ALL ;
```

```
DEL My_Table ;
```

Both examples will delete all the rows in the table.

Since the FROM and the ALL are optional, and the DELETE can be abbreviated, the second example still removes all rows from a table and executes exactly the same as the above statement:

A DELETE Example Deleting only Some of the Rows

```
DELETE FROM My_Table  
WHERE Column6 < 1001231 ;
```

The DELETE example above only removes the rows that contained a date value less than 1001231 (December 31, 2000) in Column6 (DATE, data type) and leaves all rows newer than or equal to the date:

Subquery and Join DELETE Command Syntax

The subquery syntax for the DELETE statement follows:

```
DEL[ETE] <table-name>
WHERE <column-name> [..., <column-name> ]
    IN ( SELECT <column-name> [...,<column-name> ]
        FROM <table-name> [ AS <alias-name> ]
        [ WHERE condition ... ] ) [ ALL ] ;
```

The join syntax for DELETE statement follows:

```
DEL[ETE] <table-name>
[ FROM <table-name> [ AS <alias-name> ] ]
WHERE <table-name>.<column-name>=<table-name>.<column-name>
    [ AND <condition> ]
    [ OR <condition> ] [ ALL ] ;
```

Sometimes it is desirable to delete rows from one table based on their existence in or by matching a value stored in another table. For example, you may be asked to give a raise to all people in the Awards Table. To access these rows from another table for comparison, a subquery or a join operation can be used.

Example of Subquery DELETE Command

To remove rows from My_Table using another table called Control_Del_Tbl the next DELETE uses a subquery operation to accomplish the operation:

```
DELETE FROM My_Table  
WHERE Column2 IN ( SELECT Column2 FROM Control_Del_Tbl  
                    WHERE Column3 > 5000 AND Column4 IS NULL );
```

The above uses a Subquery and the DELETE command.

Example of Join DELETE Command

To remove the same rows from My_Table using a join with the table called Control_Del_Tbl, the following is another technique to accomplish the same DELETE operation as the subquery example on the previous page.

```
DELETE My_Table  
      FROM Control_Del_Tbl AS Ctl_Tbl  
 WHERE My_Table.Column2 = Ctl_Tbl.Column2  
       AND My_Table.Column1 = Ctl_Tbl.Column1  
       AND Ctl_Tbl.Column4 IS NULL ;
```

The previous example could also be written using the format below. However, an alias cannot be used with this format:

```
DELETE My_Table  
 WHERE My_Table.Column2 = Control_Del_Tbl.Column2  
       AND My_Table.Column1 = Control_Del_Tbl.Column1  
       AND Control_Del_Tbl.Column4 IS NULL ;
```

The above uses a Join and the DELETE and is equivalent to the previous subquery example we saw on the previous page.

Fast Path DELETE

Fast Path DELETE always occur when the WHERE clause is omitted.

However, most of the time, it is not desirable to delete all of the rows. Instead, it is more practical to remove older rows to make room for newer rows or periodically purge data rows beyond the scope of business requirements.

For instance, the table is supposed to contain twelve months worth of data and it is now month number thirteen. It is now time to get rid of rows that are older than twelve months.

As soon as a WHERE clause is used in a DELETE, it must take the slow path to delete the rows. This simply means that it must log or journal a copy of each deleted row. This is to allow for the potential that the command might fail. If that should happen, Teradata can automatically put the deleted rows back into the table using a ROLLBACK. As slow as this additional processing makes the command, it is necessary to insure data integrity.

To use the Fast Path, a technique is needed that eliminates the journal logging. The trick is again to use a Fast Path INSERT / SELECT. This means, we insert the rows that need to be kept into an empty table.

Fast Path DELETE Example # 1

Normal Path Processing for the DELETE (uses the Transient Journal):

```
DELETE FROM My_Table  
WHERE Column6 < 1001231 ;
```

There are three different methods for using Fast Path Processing in BTEQ for a DELETE. The first method uses an INSERT/SELECT. It will be fast, but it does require privileges for using the appropriate DDL. It also requires that additional PERM space be available for temporarily holding both the rows to be kept and all of the original rows at the same time.

```
INSERT INTO My_Table_Copy  
        SELECT * FROM My_Table WHERE Column6 > 1001230  
; DROP TABLE My_Table  
; RENAME My_Table_Copy to My_Table ;
```

The first example does NOT use the Fast Path Delete, but the second example does.

Fast Path DELETE Example # 2

Normal Path Processing for the DELETE (uses the Transient Journal):

```
DELETE FROM My_Table  
WHERE Column6 < 1001231 ;
```

This next method also uses an INSERT/SELECT and will be fast. It does not require privileges for using any DDL. It probably will not be faster than the first method, since the rows must all be put back into the original table. However, the table is empty and the Fast Path will be used:

```
INSERT INTO My_Table_Copy  
    SELECT * FROM My_Table WHERE Column6 >= 1001230  
; DELETE My_Table  
; INSERT INTO My_Table  
    SELECT * FROM My_Table_Copy ;
```

The first example does NOT use the Fast Path Delete, but the second example does.

Fast Path DELETE Example # 3

This INSERT/SELECT uses a Global temporary table to prepare for the single transaction to copy My_Table in BTEQ:

```
INSERT INTO My_Global_Table_Copy  
    SELECT * FROM My_Table WHERE Column6 >= 1001230  
; DELETE My_Table  
; INSERT INTO My_Table  
    SELECT * FROM My_Global_Table_Copy ;
```

A Global Temporary Tables requires that TEMPORARY space be available for temporarily holding the rows to be kept and all of the original rows at the same time. A Volatile Temporary table could also be used. Its space comes from spool. However, it requires a CREATE statement to build the table, unlike Global Temporary tables. More information on Temporary tables is available in this book.

If you are not using BTEQ, these statements can be used in a macro. This works because macros always execute as a transaction.

MERGE INTO

Here are the V2R5 MERGE Rules:

- 1) The Source relation must be a single row.
- 2) The Primary Index of the target relation must use an equality condition to a numeric constant or a string literal in the ON Clause.
- 3) You cannot request an error log.

Here are the V12 and above MERGE Rules:

- 1) The Source relation can be either a single row or multiple rows.
- 2) The Primary Index of the target relation must be bound by use of an equality condition to an explicit term or to a column set of the source relation.
- 3) The Primary Index of the target table cannot be updated.
- 4) The INSERT if the WHEN NOT MATCHED Clause is specified, must have the value specified in the ON clause with the target table primary index of the target table, which causes the INSERT to be on the local AMP.

MERGE merges a source row set into a target table based on whether there is a MATCH or whether there is a NOT MATCH condition. If there is a MATCH then Teradata will UPDATE the row, but if there is a NOT MATCH condition it will INSERT the row. Teradata Extension pre V12 and ANSI Version on Teradata V12.

MERGE INTO Example that Matches

The first query shows Squiggy Jones in the Employee_Table with a salary of 32500.00.

```
SELECT * FROM Employee_Table WHERE Employee_No = 2000000;
```

Employee_No	Dept_No	Last_Name	First_Name	Salary
2000000	?	Jones	Squiggy	32500.00

I will now perform a MERGE that will have a MATCH.

```
MERGE INTO Employee_Table
USING VALUES (2000000, NULL, 'Jones', 'Squiggy', 40000.00)
AS E (Emp, Dept, Las, Fir, Sal)
ON Employee_No = Emp
WHEN MATCHED THEN
    UPDATE set salary = Sal
WHEN NOT MATCHED THEN
    INSERT VALUES (Emp, Dep, Las, Fir, Sal);
```

```
SELECT * FROM Employee_Table WHERE Employee_No = 2000000;
```

Employee_No	Dept_No	Last_Name	First_Name	Salary
2000000	?	Jones	Squiggy	40000.00

MERGE INTO Example that does NOT Match

I will now perform a MERGE that will have a NOT MATCH situation because Employee_No 3000000 does not exist in the Employee_Table.

```
MERGE INTO Employee_Table
    USING VALUES (3000000, 400, 'Coffing', 'TeraTom', 300000.00)
        AS E (Emp, Dep, Las, Fir, Sal)
    ON Employee_No = Emp
        WHEN MATCHED THEN
            UPDATE set salary = Sal
        WHEN NOT MATCHED THEN
            INSERT VALUES (Emp, Dep, Las, Fir, Sal);
```

```
SELECT * FROM Employee_Table WHERE Employee_No = 3000000;
```

Employee_No	Dept_No	Last_Name	First_Name	Salary
3000000	400	Coffing	TeraTom	300000.00

There is no employee 3000000 that exists before the MERGE INTO statement, but once the MERGE INTO statement runs (and doesn't find a Match) it INSERTS employee 3000000 into the table.

OReplace

OReplace will replace characters or eliminate them.

Just give the replace string as a zero length string and it removes the character.

So if col1 contains '123*45*67*89*' and you:

OReplace (col1,'*','')

the result is

123456789

The OReplace function is a UDF that will replace certain characters with others.

Chapter 32

Stored Procedure

Functions

“Freedom from effort in the present merely means that there has been effort stored up in the past.”

-Theodore Roosevelt

Table of Contents Chapter 32 - Stored Procedure Functions

- [Stored Procedures Vs. Macros](#)
- [Creating a Stored Procedure](#)
- [How you CALL a Stored Procedure](#)
- [Label all BEGIN and END statements except the first ones](#)
- [How to Declare a Variable](#)
- [How to Declare a Variable and then SET the Variable](#)
- [An IN Variable is passed to the Procedure during the CALL](#)
- [The IN, OUT and INOUT Parameters](#)
- [Using IF inside a Stored Procedure](#)
- [Example of two Stored Procedures with different techniques](#)
- [Using Loops in Stored Procedures](#)
- [You can Name the First Begin and End if you choose](#)
- [Using Keywords LEAVE vs. UNTIL for LEAVE vs. REPEAT](#)

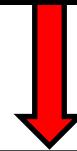
Stored Procedures Vs. Macros

Macros	Stored Procedures
<ul style="list-style-type: none">• Contains SQL• May contain BTEQ Dot commands• Parameter values can be passed• May retrieve 1 or more rows• Stored in DBC PERM Space• Returns rows to the client	<ul style="list-style-type: none">• Contains SQL• Contains comprehensive SPL• Parameter values can be passed to it• Must use a cursor to retrieve > than 1 row• Stored in DATABASE or USER PERM• May return 1 or more values to client as parameter

Stored Procedures are a lot like Macros. However, they **manipulate data a row** at a time. Stored Procedures take up PERM Space, unlike Views and Macros that do NOT. Stored Procedures are actually compiled, and will use a Cursor to retrieve or manipulate more than one row. Although, Stored Procedures utilize SQL they also utilize SPL, which stands for Stored Procedure Language, which provides loops, while, etc.

Creating a Stored Procedure

The Name of the Stored
Procedure here!



```
CREATE PROCEDURE First_Procedure ( )  
BEGIN  
    INSERT INTO Customer_Table DEFAULT VALUES;  
END ;
```

Begin and End
REQUIRED



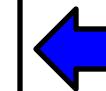
THE **BEGIN** and **END** statements are required in all Stored Procedures. Don't miss a semi-colon. They are everywhere. I have bolded them for your convenience.

How you CALL a Stored Procedure

1 CREATE PROCEDURE First_Procedure ()
BEGIN
 INSERT INTO Customer_Table DEFAULT VALUES;
END;

2

CALL First_Procedure ();



Parentheses
Needed

3

SELECT * FROM Customer_Table ORDER BY 1 ;

Customer_Number	Customer_Name	Phone_Number
?	?	?
11111111	Billy's Best Choice	555-1234
31313131	Acme Products	555-1111
31323134	Ace Consulting	555-1212
57896883	XYZ Plumbing	347-8954
87323456	Databases N-U	322-1012

Default
Values
row
placed
inside
Table

You SELECT from a View, EXECUTE a Macro, and you CALL a Stored Procedure.

Label all BEGIN and END statements except the first ones

1

```
CREATE PROCEDURE Second_Procedure( )  
BEGIN  
    INSERT INTO Customer_Table DEFAULT VALUES;  
    SecondSection:BEGIN  
        DELETE FROM Customer_Table WHERE Customer_Number is NULL;  
    END SecondSection; ←  
END;
```

2

```
CALL Second_Procedure();
```

When you have multiple BEGIN and END statements, you have to label them all (except for the first BEGIN and END statements). We have labeled our next set of BEGIN and END SecondSection.

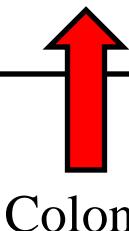
How to Declare a Variable

1

```
CREATE PROCEDURE Declare_Procedure( )  
BEGIN  
    DECLARE var1  INTEGER  DEFAULT 11111111;  
    DELETE FROM Customer_Table WHERE Customer_Number = :var1;  
END;
```

2

```
CALL Declare_Procedure();
```



Colon

When you **DECLARE** a variable and then reference that variable later, a colon is always in front of the Variable.

How to Declare a Variable and then SET the Variable

1

```
CREATE PROCEDURE SetVar_Procedure ( )  
BEGIN  
    DECLARE var1 INTEGER ;  
    SET var1 = 31313131 ;  
    DELETE FROM Customer_Table WHERE Customer_Number = :var1;  
END;
```

Semi-Colon

2

```
CALL SetVar_Procedure () ;
```

Colon

Once a variable and the data type is defined the value must be assigned. SET is the more flexible a method compared to DEFAULT.

An IN Variable is passed to the Procedure during the CALL

1

```
CREATE PROCEDURE PassInput_Procedure (IN var1 INTEGER )
BEGIN
    DELETE FROM Customer_Table WHERE Customer_Number = :var1;
END;
```



2

```
CALL PassInput_Procedure (31323134 );
```

The Variable Var1 was not assigned with the DEFAULT or the SET, but instead passed as a parameter. There are three types of parameters (IN, OUT, INOUT). In this example, an IN is being used.

The IN, OUT and INOUT Parameters

1

```
CREATE PROCEDURE Test_Proc  
    (IN var1 BYTEINT, IN var2 BYTEINT, OUT Msg CHAR(20) )  
BEGIN  
    CASE WHEN var1 = var2 THEN Set Msg = 'They are equal' ;  
        WHEN var1 < var2 THEN Set Msg = 'Variable 1 less' ;  
        ELSE Set Msg = 'Variable 1 greater' ;  
    END CASE;  
END;
```

2

```
CALL Test_Proc (1,2,Msg) ;
```

Msg

Variable 1 Less

There are three types of parameters (**IN, OUT, INOUT**). This is an example of an IN and an OUT parameter. What that means is this Stored Procedure will take a parameter in and then spit something out.

Using IF inside a Stored Procedure

1

```
CREATE PROCEDURE TestIF_Proc  
(IN var1 BYTEINT, IN var2 BYTEINT, OUT Msg CHAR(20) )  
BEGIN  
    IF var1 = var2 THEN SET Msg = 'They are equal';  
    END IF;  
    IF var1 < var2 THEN SET Msg = 'Variable 1 less';  
    END IF;  
    IF var1 > var2 THEN SET Msg = 'Variable 1 greater';  
    END IF;  
END;
```

2

```
CALL TestIF_Proc (2,2,Msg) ;
```

Msg

They are equal

Why did the Msg say “They are equal”?

Example of two Stored Procedures with different techniques

```
CREATE PROCEDURE TestELSE_Proc  
(IN var1 BYTEINT, IN var2 BYTEINT,  
 OUT Msg CHAR(20))  
BEGIN  
IF var1 = var2 THEN  
    SET Msg = 'They are equal';  
    ELSEIF var1 < var2 THEN  
        SET Msg = 'Variable 1 less';  
        ELSE  
            SET Msg = ' Variable 1 greater';  
        END IF;  
    END IF;  
END;
```

1

```
CREATE PROCEDURE TestIF_Proc  
(IN var1 BYTEINT, IN var2 BYTEINT,  
 OUT Msg CHAR(20))  
BEGIN  
    IF var1 = var2 THEN  
        SET Msg = 'They are equal';  
    END IF;  
    IF var1 < var2 THEN  
        SET Msg = 'Variable 1 less';  
    END IF;  
    IF var1 > var2 THEN  
        SET Msg = 'Variable 1 greater';  
    END IF;  
END;
```

2

These queries do the SAME thing. However, the first one is more efficient because it only does TWO calculations instead of three.

Using Loops in Stored Procedures

```
CREATE Table My_Log_Tbl  
(  
    Cntr      Integer  
,TheTime  Time  
) Primary Index (Cntr);
```

1

```
CREATE PROCEDURE Inserter_Five ( )  
LOOPER:BEGIN  
    DECLARE Cntr INTEGER          DEFAULT 0;  
    Loopit:LOOP  
        SET Cntr = Cntr + 1;  
        IF Cntr > 5 THEN LEAVE Loopit; ←  
        END IF;  
        INSERT INTO My_Log_Tbl  
            VALUES (:Cntr, TIME);  
    END LOOP Loopit ;  
END LOOPER;
```

3

```
CALL Inserter_Five () ;
```

4

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

Cntr	TheTime
1	09:48:43
2	09:48:43
3	09:48:43
4	09:48:43
5	09:48:43

LOOPS require Labeling. Much like when you have more than one BEGIN/END.

You can Name the First Begin and End if you choose

```
CREATE Table My_Log_Tbl  
(  
    Cntr      Integer  
,TheTime  Time  
) Primary Index (Cntr);
```

1

First we have to CREATE the Table.

```
CREATE PROCEDURE Inserter_Five ( )  
LOOPER:BEGIN  
    DECLARE Cntr INTEGER      DEFAULT 0;  
    Loopit:LOOP  
        SET Cntr = Cntr + 1;  
        IF Cntr > 5 THEN LEAVE Loopit;  
        END IF;  
        INSERT INTO My_Log_Tbl  
            VALUES (:Cntr, TIME);  
    END LOOP Loopit ;  
END LOOPER;
```

2

Then we build the Stored Procedure

3

```
CALL Inserter_Five () ;
```

Now we call the Stored Procedure

4

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

Cntr	TheTime
1	09:48:43
2	09:48:43
3	09:48:43
4	09:48:43
5	09:48:43

This loops 5 times! We didn't have to label Looper because it's the first Begin and End. The LEAVE statement is how the LOOP is told it is done looping.

Using Keywords LEAVE vs. UNTIL for LEAVE vs. REPEAT

--Procedure One

```
CREATE PROCEDURE Ins5( )
LOOPER:BEGIN
DECLARE Cntr INTEGER
    DEFAULT 0;
Loopit:LOOP
    SET Cntr = Cntr + 1;
    IF Cntr > 5 THEN LEAVE Loopit;
    END IF;
    INSERT INTO My_Log_Tbl
        VALUES (:Cntr, TIME);
END LOOP Loopit ;
END LOOPER;
```

--Procedure Two

```
CREATE PROCEDURE Ins5A( )
LOOPER:BEGIN
DECLARE Cntr INTEGER
    DEFAULT 0;
Loopit:REPEAT
    SET Cntr = Cntr + 1;
    INSERT INTO My_Log_Tbl
        VALUES (:Cntr, TIME);
    UNTIL Cntr > 4
    END REPEAT Loopit ;
END LOOPER;
```



Both Procedures above do the same thing. The **UNTIL** keyword in Procedure Two jumps it out of the REPEAT Loop. There are some differences in the above. The first example (Procedure One) tests Cntr before the INSERT, but Procedure two does not so Procedure two will always do at least one INSERT no matter what Cntr is set at.

Chapter 33

Trigger Functions

“Ideas pull the trigger, but instinct loads the gun.”

-Don Maquis

Table of Contents Chapter 33 – Trigger Functions

- [The Fundamentals of Triggers](#)
- [CREATING A Trigger](#)
- [FOR EACH STATEMENT vs. FOR EACH ROW](#)
- [Using ORDER when Similar Triggers Exist](#)

The Fundamentals of Triggers

- 1) A Trigger is an Event driven maintenance operation (Object Type ‘G’)
- 2) The event is caused (**Trigger Fired**) by one or more modifications to a table
- 3) The Original Modification, the TRIGGER, and all subsequent triggers constitute a Single Transaction
- 4) The SQL and all Triggers associated must each work or all are Rolled Back
- 5) The user’s initial SQL maintenance request that causes a row to change in a table and then causes a trigger to fire (execute) can be:
INSERT, UPDATE, DELETE or INSERT/SELECT
(SELECT will NOT work to fire a Trigger)
- 6) Once a Trigger fires it can perform SQL such as INSERT, UPDATE, DELETE, INSERT/SELECT, ABORT/ROLLBACK, EXEC

Triggers **fire automatically** when a Triggering Event occurs. A Trigger can even fire and **EXEC** a Macro.

CREATING A Trigger

```
CREATE TABLE EmpLog_Table  
  (UserUpdating Char(30)  
 ,DateTime Timestamp  
 ,Employee_No Integer  
 ,Oldsal decimal(10,2)  
 ,newsal decimal(10,2)  
 )Primary index(Employee_No);
```

```
CREATE TRIGGER EmpRaise  
 AFTER UPDATE OF (Salary) ON Employee_Table  
 REFERENCING OLD AS oldrow NEW AS newrow  
 FOR EACH ROW  
 (INSERT INTO EmpLog_Table  
   VALUES (USER, CURRENT_TIMESTAMP,  
   oldrow.employee_no, oldrow.salary, newrow.salary));
```

The Table EmpLog_Table needs to exist before the Trigger EmpRaise can successfully be created. The great thing about a trigger is that users can see the OLD Row and the New Row after the UPDATE and both can be recorded for auditing purposes. Above is an AFTER Trigger because it fires AFTER an UPDATE, vs. some triggers that fire just BEFORE a change happens.

You can also have a Trigger fire once for an event or fire for every Row that changes. The statement FOR EACH ROW specifies to fire for every ROW change.

FOR EACH STATEMENT vs. FOR EACH ROW

```
CREATE TABLE Audit_Table  
  (UserUpdating Char(30)  
   ,DateTime Timestamp  
  )Primary index(UserUpdating);
```

```
CREATE TRIGGER AuditCheck  
  AFTER UPDATE OF (Salary) ON Employee_Table  
  FOR EACH statement  
  (INSERT INTO Audit_Table  
    VALUES(USER, CURRENT_TIMESTAMP));
```

```
UPDATE Employee_Table  
SET salary = Salary * 1.1 ;
```

Does AuditCheck Fire for each row updated or once per statement? For each statement.

Using ORDER when Similar Triggers Exist

```
CREATE TRIGGER INStrig1
AFTER INSERT ON CasTbl_1
ORDER 100
REFERENCING NEW AS afterrow
FOR EACH ROW
( INSERT INTO CasTbl_2 values
(afterrow.col1+1, beforerow.col2*2) );
```

```
CREATE TRIGGER INStrig2
AFTER INSERT ON CasTbl_1
ORDER 200
REFERENCING NEW AS afterrow
FOR EACH ROW
( INSERT INTO CasTbl_3 values
(afterrow.col1+2, beforerow.col2*4) );
```

When two different triggers implemented on the same table (that are similar) and they don't have an ORDER statement you can't tell which will fire first (random). In the example above which Trigger will fire first? The one with ORDER 100, not 200.

Chapter 34

Math Functions

“Ideas pull the trigger, but instinct loads the gun.”

-Don Maquis

Table of Contents Chapter 34 – Math Functions

- [What is the Order of Precedents?](#)
- [What is the Answer to this Math Question?](#)
- [What is the Answer to this Math Question?](#)
- [What is the Answer to this Math Question?](#)

What is the Order of Precedents?

Order of Precedents for Math

() Parentheses

** Exponentiation

* Multiplication

/ Division

+ Addition

- Subtraction

Mathematics can be used within your Syntax. But, like any math problems, it has an Order of Precedents.

What is the Answer to this Math Question?

SELECT $(2 + 4) * 5$

What is the Answer?

30

This is an example of mathematics taking place with the Order of Precedents being respected.

What is the Answer to this Math Question?

SELECT $2 + 4 / 5$

What is the Answer?

2

The answer to this example is 2. This is because when a decimal isn't used within the equation, the system will NOT allow for a decimal to be part of the answer set.

Instead of 2.8, it drops the decimal altogether and just gives you a 2.

What is the Answer to this Math Question?

SELECT $2 + 4.0 / 5$

What is the Answer?

2.8

In order to see the decimal in your answer set, you must have at least one decimal in your equation.

Chapter 35

The SAMPLE

Function

“The universe extends beyond the mind of man, and is more complex than the small sample one can study.”

- Kenneth L. Pike

Table of Contents Chapter 35 - The SAMPLE Function

- [The SAMPLE Function and Syntax](#)
- [SAMPLE Function Examples](#)
- [A SAMPLE Example that asks for Multiple Samples](#)
- [A SAMPLE Example with the SAMPLEID](#)
- [A SAMPLE Example WITH REPLACEMENT](#)
- [A SAMPLE Example with Four 10% Samples](#)
- [A Randomized SAMPLE](#)
- [A SAMPLE with Conditional Logic](#)
- [Aggregates and A SAMPLE using a Derived Table](#)
- [Random Number Generator](#)
- [Using Random to SELECT a Percentage of Rows](#)
- [Using Random and Aggregations](#)

The SAMPLE Function and Syntax

The syntax for the SAMPLE function:

```
SAMPLE [ WITH REPLACEMENT ] [  
RANDOMIZED ALLOCATION ]  
[ WHEN <condition> THEN ]  
{<number-of-rows> | <percentage>}  
[...,<number-of-rows> |  
<percentage>]  
[ ELSE {<number-of-rows> |  
<percentage>} END ]
```

The Sampling function (SAMPLE) permits a SELECT to randomly return rows from a Teradata database table. It allows the request to specify either an absolute number of rows or a percentage of rows to return. Additionally, it provides an ability to return rows from multiple samples.

SAMPLE Function Examples

Bring back 5 rows

```
SELECT *  
FROM Student_Course_Table  
SAMPLE 5 ;
```

5 Rows Returned

Student_ID	Course_ID
280023	210
260000	400
125634	100
125634	220
333450	500

Bring back 25% of the rows

```
SELECT *  
FROM Student_Course_Table  
SAMPLE .25 ;
```

4 Rows Returned

Student_ID	Course_ID
125634	100
231222	210
234121	100
322133	300

The above example uses the SAMPLE to get a random sample of the sales table. Notice that five rows came back because we asked for a SAMPLE 5 in the first example. In the second example we got a percentage of 25% of the rows.

A SAMPLE Example that asks for Multiple Samples

Bring back two samples
of 25%, but no duplicates!

```
SELECT *  
FROM Student_Course_Table  
SAMPLE .25, .25  
ORDER BY 1,2 ;
```

8 Rows Returned

<u>Student_ID</u>	<u>Course_ID</u>
123250	100
125634	100
125634	220
231222	220
260000	400
280023	210
322133	300
333450	500

Sometimes, a single sampling of the data is not sufficient. The SAMPLE function can be used to request more than one sample by listing either the number of rows or the percentage of the rows to be returned. The above example uses the SAMPLE function to request multiple samples:

A SAMPLE Example with the SAMPLEID

```
SELECT Student_ID  
      ,Course_ID  
      ,SAMPLEID  
  FROM Student_Course_Table  
 SAMPLE 5,5,5  
 ORDER BY 3,1,2 ;
```

Bring back 3 Samples with each sample having 5 rows.

Student_ID	Course_ID	SAMPLEID
125634	100	1
125634	220	1
322133	300	1
324652	200	1
333450	400	1
123250	100	2
125634	200	2
231222	220	2
280023	210	2
322133	220	2
231222	210	3
234121	100	3
260000	400	3
333450	500	3

Although multiple samples were taken, the rows came back as a single answer set consisting of 5 rows, 5 rows and then 4 rows of the data. The SAMPLEID column name can be used to distinguish between each sample. The last sample only brought back 4 rows because there are only 14 rows in the table and there will be no duplicates.

A SAMPLE Example WITH REPLACEMENT

```
SELECT Student_ID  
      ,Course_ID  
      ,SAMPLEID  
  FROM Student_Course_Table  
 SAMPLE WITH REPLACEMENT 5, 5, 5  
ORDER BY 3, 1, 2 ;
```

Bring back 3 Samples with each sample having 5 rows.

You can have **duplicates** now!

Student_ID	Course_ID	SAMPLEID
125634	100	1
231222	210	1
231222	210	1
280023	210	1
280023	210	1
125634	100	2
125634	220	2
231222	220	2
322133	220	2
324652	200	2
234121	100	3
260000	400	3
322133	220	3
324652	200	3
333450	500	3

At the same time, you may wish for rows to be available for all samples. The above example uses the SAMPLE WITH REPLACEMENT function with the SAMPLEID to request multiple samples and denote which sample each row came from.

A SAMPLE Example with Four 10% Samples

```
SELECT Student_ID  
      ,Course_ID  
      ,SAMPLEID  
  FROM Student_Course_Table  
 SAMPLE .1, .1, .1, .1  
ORDER BY SAMPLEID ;
```

Student_ID	Course_ID	SAMPLEID
125634	220	1
123250	100	2
324652	200	3
280023	210	4

Bring back 4 Samples with each sample having 10% of the rows.

The above example uses the SAMPLE function with the SAMPLEID to request multiple samples as a percentage and denote which sample each row came from. Although 10% of 14 rows is 1.4, it can only return a whole row and therefore, 1 row is returned per sample. Also, since SAMPLEID is a column, it can be used as the sort key.

A Randomized SAMPLE

```
SELECT Student_ID  
      ,Course_ID  
      ,SAMPLEID  
  FROM Student_Course_Table  
 SAMPLE RANDOMIZED ALLOCATION .1, .1, .1, .1;
```

Bring back 4 Samples with each sample having 10% of the rows and do a random sample across the entire population.

Student_ID	Course_ID	SAMPLEID
125634	100	2
125634	200	3
125634	220	1
234121	100	4

By default the SAMPLE function does a proportional sampling across all AMPs in the system. Therefore it is not a simple random sample across the entire population of rows. If you wish a random sample across the entire population, use the RANDOMIZED ALLOCATION as seen above.

A SAMPLE with Conditional Logic

```
SELECT Student_ID  
      ,Course_ID  
      ,SAMPLEID  
  FROM Student_Course_Table  
 SAMPLE RANDOMIZED ALLOCATION  
        WHEN Course_ID >200 THEN .1, .1 ELSE .2, .2  
        END  
 ORDER BY 3;
```

Bring back two Samples with one row per sample if the Course_ID is ≤ 200 .

Bring back two Samples with two rows each if the Course_ID > 200 .

Student_ID	Course_ID	SAMPLEID
234121	100	1
125634	100	2
333450	500	3
231222	220	3
125634	220	4
322133	220	4

The above query brings back two Samples with one row per sample if the Course_ID is ≤ 200 . Else it bring back two Samples with two rows each if the Course_ID > 200 . This means it will attempt to bring back six records total in four different samples.

Aggregates and A SAMPLE using a Derived Table

```
SELECT count(distinct(Course_ID))
      FROM (SEL Course_ID FROM Student_Course_Table
SAMPLE 5) DT ;
```

COUNT(Distinct(Course_ID))

4

A second run of the same SELECT might very well yield these results:

COUNT(Distinct(Course_ID))

5

Although they look like Aggregates, they are not normally compatible with them in the same SELECT list. As demonstrated here, aggregation can be performed, however, they must be calculated in a temporary or derived table.

The above example uses the SAMPLE function to request multiple samples to create a derived table (See Temporary Tables chapter). Then, the unique rows will be counted to show the random quality of the SAMPLE function:

Random Number Generator

The syntax for RANDOM is:

RANDOM(<low-literal-value>, <high-literal-value>)

The example below uses the RANDOM function to return a random number between 1 and 20:

SELECT RANDOM(1, 20);

Random(1,20)

14

The RANDOM function generates a random number that is inclusive for the numbers specified in the SQL that is greater than or equal to the first argument and less than or equal to the second argument.

The RANDOM function may be used in the SELECT list, in a CASE, in a WHERE clause, in a QUALIFY, in a HAVING and in an ORDER BY. The RANSOM function can be used creatively to provide some powerful functionality within SQL.

Using Random to SELECT a Percentage of Rows

The next SELECT uses RANDOM to randomly select 5% of the rows from the table:

```
SELECT *
  FROM Sales_Table
 WHERE RANDOM(1, 100) = 5;
```

Product_ID	Sale_Date	Daily_Sales
1000	09/28/2000	48850.40

There is roughly a 5% (1 out of 100) chance that a row will be returned using RANDOM in the WHERE clause, completely at random. Since SAMPLE randomly selects rows out of spool, currently RANDOM will be faster than SAMPLE. However, SAMPLE will be more accurate regarding the number of rows being returned with both the percent and row count.

Using Random and Aggregations

This example uses RANDOM to randomly generate a number that will determine which rows from the aggregation will be returned:

```
SELECT Product_ID, COUNT(Daily_Sales)
FROM Sales_Table
GROUP BY 1
HAVING COUNT(Daily_Sales) > RANDOM(1, 10) ;
```

Product_ID	Count(Daily_Sales)
2000	7
3000	7

This last example uses RANDOM to randomly generate a number that will determine which rows from the aggregation will be returned. Whenever a random number is needed within the SQL, RANDOM is a great tool.

Chapter 36

Statistical Aggregate

Functions

“Its not that figures lie, its that liars figure.”

-Anonymous

Table of Contents Chapter 36 - Statistical Aggregate Functions

- [The Stats Table](#)
- [The KURTOSIS Function](#)
- [A Kurtosis Example](#)
- [The SKEW Function](#)
- [A SKEW Example](#)
- [The STDDEV_POP Function](#)
- [A STDDEV_POP Example](#)
- [The STDDEV_SAMP Function](#)
- [A STDDEV_SAMP Example](#)
- [The VAR_POP Function](#)
- [A VAR_POP Example](#)
- [The VAR_SAMP Function](#)
- [A VAR_SAMP Example](#)
- [The CORR Function](#)
- [A CORR Example](#)
- [Another CORR Example so you can Compare](#)
- [The COVAR_POP Function](#)
- [A COVAR_POP Example](#)
- [Another COVAR_POP Example so you can Compare](#)
- [The REGR_INTERCEPT Function](#)
- [A REGR_INTERCEPT Example](#)
- [Another REGR_INTERCEPT Example so you can Compare](#)
- [A REGR_SLOPE Example](#)
- [Another REGR_SLOPE Example so you can Compare](#)
- [Using GROUP BY](#)
- [No Having Clause Vs Use of HAVING](#)

The Stats Table

Col1	Col2	Col3	Col4	Col5	Col6
1	1	1	30	1	0
2	1	1	29	2	5
3	3	10	28	3	10
4	3	10	27	4	15
5	3	10	26	5	20
6	4	10	25	6	30
7	5	10	24	7	30
8	5	10	23	8	30
9	5	10	22	9	35
10	5	20	21	10	35
11	7	20	20	22	40
12	7	20	19	12	40
13	9	20	18	13	45
14	9	20	17	14	45
15	9	20	16	15	50
16	9	20	15	14	55
17	10	20	14	13	55
18	10	20	13	12	60
19	10	20	12	11	60
20	10	20	11	9	65
21	10	20	10	8	65
22	10	20	9	7	65
23	13	20	8	6	70
24	13	30	7	5	70
25	13	30	6	4	80
26	14	40	5	3	85
27	15	40	4	2	90
28	15	50	3	1	90
29	16	50	2	1	95
30	16	60	1	1	100

Above is the Stats_Table data in which we will use in our statistical examples.

The KURTOSIS Function

```
SELECT KURTOSIS(col1) AS KofColl  
FROM Stats_Table;
```

The KURTOSIS function is used to return a number that represents the sharpness of a peak on a plotted curve of a probability function for a distribution compared with the normal distribution.

A high value result is referred to as leptokurtic. While a medium result is referred to as mesokurtic and a low result is referred to as platykurtic.

A positive value indicates a sharp or peaked distribution and a negative number represents a flat distribution. A peaked distribution means that one value exists more often than the other values. A flat distribution means there is the same quantity values exist for each number.

If you compare this to the row distribution associated within Teradata, most of the time a flat distribution is best, with the same number of rows stored on each AMP. Having skewed data represents more of a lumpy distribution.

A Kurtosis Example

Stats_Table

Col1	Col2	Col3	Col4	Col5	Col6
1	1	1	30	1	0
2	1	1	29	2	5
3	3	10	28	3	10
4	3	10	27	4	15
5	3	10	26	5	20
6	4	10	25	6	30
7	5	10	24	7	30
8	5	10	23	8	30
9	5	10	22	9	35
10	5	20	21	10	35
11	7	20	20	22	40
12	7	20	19	12	40
13	9	20	18	13	45
14	9	20	17	14	45
15	9	20	16	15	50
16	9	20	15	14	55
17	10	20	14	13	55
18	10	20	13	12	60
19	10	20	12	11	60
20	10	20	11	9	65
21	10	20	10	8	65
22	10	20	9	7	65
23	13	20	8	6	70
24	13	30	7	5	70
25	13	30	6	4	80
26	14	40	5	3	85
27	15	40	4	2	90
28	15	50	3	1	90
29	16	50	2	1	95
30	16	60	1	1	100

```
SELECT KURTOSIS(col1) AS KofCol1  
      ,KURTOSIS(col2) AS KofCol2  
      ,KURTOSIS(col3) AS KofCol3  
      ,KURTOSIS(col4) AS KofCol4  
      ,KURTOSIS(col5) AS KofCol5  
      ,KURTOSIS(col6) AS KofCol6  
FROM   Stats_Table;
```

KofCol1	KofCol2	KofCol3	KofCol4	KofCol5	KofCol6
-1	-1	1	-1	-1	-1

The SKEW Function

```
SELECT SKEW(col1) AS SKofCol1  
FROM Stats_Table;
```

The Skew indicates that a distribution does not have equal probabilities above and below the mean (average). In a skew distribution, the median and the mean are not coincident, or equal.

Where:

- a median value < mean value = a positive skew
- a median value > mean value = a negative skew
- a median value = mean value = no skew

Syntax for using SKEW:

`SKEW(<column-name>)`

A SKEW Example

Stats_Table

Col1	Col2	Col3	Col4	Col5	Col6
1	1	1	30	1	0
2	1	1	29	2	5
3	3	10	28	3	10
4	3	10	27	4	15
5	3	10	26	5	20
6	4	10	25	6	30
7	5	10	24	7	30
8	5	10	23	8	30
9	5	10	22	9	35
10	5	20	21	10	35
11	7	20	20	22	40
12	7	20	19	12	40
13	9	20	18	13	45
14	9	20	17	14	45
15	9	20	16	15	50
16	9	20	15	14	55
17	10	20	14	13	55
18	10	20	13	12	60
19	10	20	12	11	60
20	10	20	11	9	65
21	10	20	10	8	65
22	10	20	9	7	65
23	13	20	8	6	70
24	13	30	7	5	70
25	13	30	6	4	80
26	14	40	5	3	85
27	15	40	4	2	90
28	15	50	3	1	90
29	16	50	2	1	95
30	16	60	1	1	100

```
SELECT SKEW(col1) AS SKofCol1  
      ,SKEW(col2) AS SKofCol2  
      ,SKEW(col3) AS SKofCol3  
      ,SKEW(col4) AS SKofCol4  
      ,SKEW(col5) AS SKofCol5  
      ,SKEW(col6) AS SKofCol6  
FROM Stats_Table;
```

SKofCol1	SKofCol2	SKofCol3	SKofCol4	SKofCol5	SKofCol6
0	-0	1	0	0	-0

A median value < mean value = a positive skew
A median value > mean value = a negative skew
A median value = mean value = no skew

The STDDEV_POP Function

```
SELECT STDDEV_POP(col1) AS SDPCol1  
FROM Stats_Table;
```

The standard deviation function is a statistical measure of spread or dispersion of values. It is the root's square of the difference of the mean (average). This measure is to compare the amount by which a set of values differs from the arithmetical mean.

The STDDEV_POP function is one of two that calculates the standard deviation. The population is of all the rows included based on the comparison in the WHERE clause.

Syntax for using STDDEV_POP:

STDDEV_POP(<column-name>)

A STDDEV_POP Example

Stats_Table

Col1	Col2	Col3	Col4	Col5	Col6
1	1	1	30	1	0
2	1	1	29	2	5
3	3	10	28	3	10
4	3	10	27	4	15
5	3	10	26	5	20
6	4	10	25	6	30
7	5	10	24	7	30
8	5	10	23	8	30
9	5	10	22	9	35
10	5	20	21	10	35
11	7	20	20	22	40
12	7	20	19	12	40
13	9	20	18	13	45
14	9	20	17	14	45
15	9	20	16	15	50
16	9	20	15	14	55
17	10	20	14	13	55
18	10	20	13	12	60
19	10	20	12	11	60
20	10	20	11	9	65
21	10	20	10	8	65
22	10	20	9	7	65
23	13	20	8	6	70
24	13	30	7	5	70
25	13	30	6	4	80
26	14	40	5	3	85
27	15	40	4	2	90
28	15	50	3	1	90
29	16	50	2	1	95
30	16	60	1	1	100

```
SELECT STDDEV_POP(col1) AS SDPCol1  
      ,STDDEV_POP(col2) AS SDPCol2  
      ,STDDEV_POP(col3) AS SDPCol3  
      ,STDDEV_POP(col4) AS SDPCol4  
      ,STDDEV_POP(col5) AS SDPCol5  
      ,STDDEV_POP(col6) AS SDPCol6  
FROM Stats_Table;
```

SDPCol1	SDPCol2	SDPCol3	SDPCol4	SDPCol5	SDPCol6
9	4	14	9	4	27

The standard deviation function is a statistical measure of spread or dispersion of values. It is the root's square of the difference of the mean (average). This measure is to compare the amount by which a set of values differs from the arithmetical mean.

The STDDEV_SAMP Function

```
SELECT STDDEV_SAMP(col1) AS SDSCol1  
FROM Stats_Table;
```

The standard deviation function is a statistical measure of spread or dispersion of values. It is the root's square of the difference of the mean (average). This measure is to compare the amount by which a set of values differs from the arithmetical mean.

The STDDEV_SAMP function is one of two that calculates the standard deviation. The sample is a random selection of all rows returned based on the comparisons in the WHERE clause. The population is for all of the rows based on the WHERE clause.

Syntax for using STDDEV_SAMP:

STDDEV_SAMP(<column-name>)

A STDDEV_SAMP Example

Stats_Table

Col1	Col2	Col3	Col4	Col5	Col6
1	1	1	30	1	0
2	1	1	29	2	5
3	3	10	28	3	10
4	3	10	27	4	15
5	3	10	26	5	20
6	4	10	25	6	30
7	5	10	24	7	30
8	5	10	23	8	30
9	5	10	22	9	35
10	5	20	21	10	35
11	7	20	20	22	40
12	7	20	19	12	40
13	9	20	18	13	45
14	9	20	17	14	45
15	9	20	16	15	50
16	9	20	15	14	55
17	10	20	14	13	55
18	10	20	13	12	60
19	10	20	12	11	60
20	10	20	11	9	65
21	10	20	10	8	65
22	10	20	9	7	65
23	13	20	8	6	70
24	13	30	7	5	70
25	13	30	6	4	80
26	14	40	5	3	85
27	15	40	4	2	90
28	15	50	3	1	90
29	16	50	2	1	95
30	16	60	1	1	100

```
SELECT STDDEV_POP(col1) AS SDSCol1  
,STDDEV_POP(col2) AS SDSCol2  
,STDDEV_POP(col3) AS SDSCol3  
,STDDEV_POP(col4) AS SDSCol4  
,STDDEV_POP(col5) AS SDSCol5  
,STDDEV_POP(col6) AS SDSCol6  
FROM Stats_Table;
```

SDSCol1	SDSCol2	SDSCol3	SDSCol4	SDSCol5	SDSCol6
9	4	14	9	5	27

The STDDEV_SAMP function is one of two that calculates the standard deviation. The sample is a random selection of all rows returned based on the comparisons in the WHERE clause. The population is for all of the rows based on the WHERE clause.

The VAR_POP Function

```
SELECT VAR_POP(col1) AS VPColl  
FROM Stats_Table;
```

The Variance function is a measure of dispersion (spread of the distribution) as the square of the standard deviation. There are two forms of Variance in Teradata, VAR_POP is for the entire population of data rows allowed by the WHERE clause.

Although standard deviation and variance are regularly used in statistical calculations, the meaning of variance is not easy to elaborate. Most often variance is used in theoretical work where a variance of the sample is needed.

There are two methods for using variance. These are the Kruskal-Wallis one-way Analysis of Variance and Friedman two-way Analysis of Variance by rank.

Syntax for using VAR_POP:

VAR_POP(<column-name>)

A VAR_POP Example

Stats_Table

Col1	Col2	Col3	Col4	Col5	Col6
1	1	1	30	1	0
2	1	1	29	2	5
3	3	10	28	3	10
4	3	10	27	4	15
5	3	10	26	5	20
6	4	10	25	6	30
7	5	10	24	7	30
8	5	10	23	8	30
9	5	10	22	9	35
10	5	20	21	10	35
11	7	20	20	22	40
12	7	20	19	12	40
13	9	20	18	13	45
14	9	20	17	14	45
15	9	20	16	15	50
16	9	20	15	14	55
17	10	20	14	13	55
18	10	20	13	12	60
19	10	20	12	11	60
20	10	20	11	9	65
21	10	20	10	8	65
22	10	20	9	7	65
23	13	20	8	6	70
24	13	30	7	5	70
25	13	30	6	4	80
26	14	40	5	3	85
27	15	40	4	2	90
28	15	50	3	1	90
29	16	50	2	1	95
30	16	60	1	1	100

```
SELECT VAR_POP(col1) AS VPCol1  
      ,VAR_POP(col2) AS VPCol2  
      ,VAR_POP(col3) AS VPCol3  
      ,VAR_POP(col4) AS VPCol4  
      ,VAR_POP(col5) AS VPCol5  
      ,VAR_POP(col6) AS VPCol6  
FROM Stats_Table;
```

VPCol1	VPCol2	VPCol3	VPCol4	VPCol5	VPCol6
75	19	191	75	20	723

The Variance function is a measure of dispersion (spread of the distribution) as the square of the standard deviation. There are two forms of Variance in Teradata, VAR_POP is for the entire population of data rows allowed by the WHERE clause.

The VAR_SAMP Function

```
SELECT VAR_SAMP(col1) AS VSCol1  
FROM Stats_Table;
```

The Variance function is a measure of dispersion (spread of the distribution) as the square of the standard deviation. There are two forms of Variance in Teradata, VAR_SAMP is used for a random sampling of the data rows allowed through by the WHERE clause.

Although standard deviation and variance are regularly used in statistical calculations, the meaning of variance is not easy to elaborate. Most often variance is used in theoretical work where a variance of the sample is needed to look for consistency.

There are two methods for using variance. These are the Kruskal-Wallis one-way Analysis of Variance and Friedman two-way Analysis of Variance by rank.

Syntax for using VAR_SAMP:

VAR_SAMP(<column-name>)

A VAR_SAMP Example

Stats_Table

Col1	Col2	Col3	Col4	Col5	Col6
1	1	1	30	1	0
2	1	1	29	2	5
3	3	10	28	3	10
4	3	10	27	4	15
5	3	10	26	5	20
6	4	10	25	6	30
7	5	10	24	7	30
8	5	10	23	8	30
9	5	10	22	9	35
10	5	20	21	10	35
11	7	20	20	22	40
12	7	20	19	12	40
13	9	20	18	13	45
14	9	20	17	14	45
15	9	20	16	15	50
16	9	20	15	14	55
17	10	20	14	13	55
18	10	20	13	12	60
19	10	20	12	11	60
20	10	20	11	9	65
21	10	20	10	8	65
22	10	20	9	7	65
23	13	20	8	6	70
24	13	30	7	5	70
25	13	30	6	4	80
26	14	40	5	3	85
27	15	40	4	2	90
28	15	50	3	1	90
29	16	50	2	1	95
30	16	60	1	1	100

```
SELECT VAR_SAMP(col1) AS VSCol1  
      ,VAR_SAMP(col2) AS VSCol2  
      ,VAR_SAMP(col3) AS VSCol3  
      ,VAR_SAMP(col4) AS VSCol4  
      ,VAR_SAMP(col5) AS VSCol5  
      ,VAR_SAMP(col6) AS VSCol6  
FROM   Stats_Table ;
```

VSCol1	VSCol2	VSCol3	VSCol4	VSCol5	VSCol6
78	20	198	78	20	748

The Variance function is a measure of dispersion (spread of the distribution) as the square of the standard deviation. There are two forms of Variance in Teradata, VAR_SAMP is used for a random sampling of the data rows allowed through by the WHERE clause.

The CORR Function

```
SELECT CORR(col1, col2) AS CCol1#2  
FROM Stats_Table;
```

The CORR function is a binary function, meaning that two variables are used as input to it. It measures the association between 2 random variables. If the variables are such that when one changes the other does so in a related manner, they are correlated. Independent variables are not correlated because the change in one does not necessarily cause the other to change.

The correlation coefficient is a number between -1 and 1. It is calculated from a number of pairs of observations or linear points (X,Y).

Where:

1 = perfect positive correlation

0 = no correlation

-1 = perfect negative correlation

Syntax for using CORR:

CORR(<column-name>, <column-name>)

A CORR Example

Stats_Table

Col1	Col2	Col3	Col4	Col5	Col6
1	1	1	30	1	0
2	1	1	29	2	5
3	3	10	28	3	10
4	3	10	27	4	15
5	3	10	26	5	20
6	4	10	25	6	30
7	5	10	24	7	30
8	5	10	23	8	30
9	5	10	22	9	35
10	5	20	21	10	35
11	7	20	20	22	40
12	7	20	19	12	40
13	9	20	18	13	45
14	9	20	17	14	45
15	9	20	16	15	50
16	9	20	15	14	55
17	10	20	14	13	55
18	10	20	13	12	60
19	10	20	12	11	60
20	10	20	11	9	65
21	10	20	10	8	65
22	10	20	9	7	65
23	13	20	8	6	70
24	13	30	7	5	70
25	13	30	6	4	80
26	14	40	5	3	85
27	15	40	4	2	90
28	15	50	3	1	90
29	16	50	2	1	95
30	16	60	1	1	100

```
SELECT CORR(col1, col2) AS CCol1#2  
      ,CORR(col1, col3) AS CCol1#3  
      ,CORR(col1, col4) AS CCol1#4  
      ,CORR(col1, col5) AS CCol1#5  
      ,CORR(col1, col6) AS CCol1#6  
FROM   Stats_Table ;
```

CCol1#2	CCol1#3	CCol1#4	CCol1#5	CCol1#6
0.986480	0.885155	-1.000000	-0.151877	0.991612

Where:

1 = perfect positive correlation

0 = no correlation

-1 = perfect negative correlation

Another CORR Example so you can Compare

Stats_Table

Col1	Col2	Col3	Col4	Col5	Col6
1	1	1	30	1	0
2	1	1	29	2	5
3	3	10	28	3	10
4	3	10	27	4	15
5	3	10	26	5	20
6	4	10	25	6	30
7	5	10	24	7	30
8	5	10	23	8	30
9	5	10	22	9	35
10	5	20	21	10	35
11	7	20	20	22	40
12	7	20	19	12	40
13	9	20	18	13	45
14	9	20	17	14	45
15	9	20	16	15	50
16	9	20	15	14	55
17	10	20	14	13	55
18	10	20	13	12	60
19	10	20	12	11	60
20	10	20	11	9	65
21	10	20	10	8	65
22	10	20	9	7	65
23	13	20	8	6	70
24	13	30	7	5	70
25	13	30	6	4	80
26	14	40	5	3	85
27	15	40	4	2	90
28	15	50	3	1	90
29	16	50	2	1	95
30	16	60	1	1	100

```

SELECT CORR(col4, col2) AS CCol4#2
      ,CORR(col4, col3) AS CCol4#3
      ,CORR(col4, col1) AS CCol4#1
      ,CORR(col4, col5) AS CCol4#5
      ,CORR(col4, col6) AS CCol4#6
FROM   Stats_Table ;
    
```

CCol4#2	CCol4#3	CCol4#1	CCol4#5	CCol4#6
-0.986480	-0.885155	-1.000000	0.151877	-0.991612

Where:

1 = perfect positive correlation

0 = no correlation

-1 = perfect negative correlation

The COVAR_POP Function

```
SELECT COVAR_POP(col1, col2) AS CCol1#2  
FROM Stats_Table;
```

The covariance is a statistical measure of the tendency of two variables to change in conjunction with each other. It is equal to the product of their standard deviations and correlation coefficients.

The covariance is a statistic used for bivariate samples or bivariate distribution. It is used for working out the equations for regression lines and the product-moment correlation coefficient.

Syntax:

COVAR(<column-name>, <column-name>)

A COVAR_POP Example

Stats_Table

Col1	Col2	Col3	Col4	Col5	Col6
1	1	1	30	1	0
2	1	1	29	2	5
3	3	10	28	3	10
4	3	10	27	4	15
5	3	10	26	5	20
6	4	10	25	6	30
7	5	10	24	7	30
8	5	10	23	8	30
9	5	10	22	9	35
10	5	20	21	10	35
11	7	20	20	22	40
12	7	20	19	12	40
13	9	20	18	13	45
14	9	20	17	14	45
15	9	20	16	15	50
16	9	20	15	14	55
17	10	20	14	13	55
18	10	20	13	12	60
19	10	20	12	11	60
20	10	20	11	9	65
21	10	20	10	8	65
22	10	20	9	7	65
23	13	20	8	6	70
24	13	30	7	5	70
25	13	30	6	4	80
26	14	40	5	3	85
27	15	40	4	2	90
28	15	50	3	1	90
29	16	50	2	1	95
30	16	60	1	1	100

SELECT

COVAR_POP(col1, col2) AS CPCol1#2
,COVAR_POP(col1, col3) AS CPCol1#3
,COVAR_POP(col1, col4) AS CPCol1#4
,COVAR_POP(col1, col5) AS CPCol1#5
,COVAR_POP(col1, col6) AS CPCol1#6

FROM Stats_Table ;

CPCol1#2	CPCol1#3	CPCol1#4	CPCol1#5	CPCol1#6
37.50	105.90	-74.92	-5.82	230.75

The covariance is a statistical measure of the tendency of two variables to change in conjunction with each other. It is equal to the product of their standard deviations and correlation coefficients.

Another COVAR_POP Example so you can Compare

Stats_Table

Col1	Col2	Col3	Col4	Col5	Col6
1	1	1	30	1	0
2	1	1	29	2	5
3	3	10	28	3	10
4	3	10	27	4	15
5	3	10	26	5	20
6	4	10	25	6	30
7	5	10	24	7	30
8	5	10	23	8	30
9	5	10	22	9	35
10	5	20	21	10	35
11	7	20	20	22	40
12	7	20	19	12	40
13	9	20	18	13	45
14	9	20	17	14	45
15	9	20	16	15	50
16	9	20	15	14	55
17	10	20	14	13	55
18	10	20	13	12	60
19	10	20	12	11	60
20	10	20	11	9	65
21	10	20	10	8	65
22	10	20	9	7	65
23	13	20	8	6	70
24	13	30	7	5	70
25	13	30	6	4	80
26	14	40	5	3	85
27	15	40	4	2	90
28	15	50	3	1	90
29	16	50	2	1	95
30	16	60	1	1	100

SELECT

COVAR_POP(col4, col2) AS CPCol4#2
,COVAR_POP(col4, col3) AS CPCol4#3
,COVAR_POP(col4, col1) AS CPCol4#1
,COVAR_POP(col4, col5) AS CPCol4#5
,COVAR_POP(col4, col6) AS CPCol4#6

FROM Stats_Table ;

CPCol4#2	CPCol4#3	CPCol4#1	CPCol4#5	CPCol4#6
-37.50	-105.90	-74.92	5.82	-230.75

The covariance is a statistical measure of the tendency of two variables to change in conjunction with each other. It is equal to the product of their standard deviations and correlation coefficients.

The REGR_INTERCEPT Function

```
SELECT REGR_INTERCEPT(col1, col2) AS RIofCol1#2  
FROM Stats_Table;
```

A regression line is a line of best fit, drawn through a set of points on a graph for X and Y coordinates. It uses the Y coordinate as the Dependent Variable and the X value as the Independent Variable.

Two regression lines always meet or intercept at the mean of the data points(x,y), where x=AVG(x) and y=AVG(y) and is not usually one of the original data points.

Syntax for using REGR_INTERCEPT:

REGR_INTERCEPT(dependent-expression, independent-expression)

A REGR_INTERCEPT Example

Stats_Table

Col1	Col2	Col3	Col4	Col5	Col6
1	1	1	30	1	0
2	1	1	29	2	5
3	3	10	28	3	10
4	3	10	27	4	15
5	3	10	26	5	20
6	4	10	25	6	30
7	5	10	24	7	30
8	5	10	23	8	30
9	5	10	22	9	35
10	5	20	21	10	35
11	7	20	20	22	40
12	7	20	19	12	40
13	9	20	18	13	45
14	9	20	17	14	45
15	9	20	16	15	50
16	9	20	15	14	55
17	10	20	14	13	55
18	10	20	13	12	60
19	10	20	12	11	60
20	10	20	11	9	65
21	10	20	10	8	65
22	10	20	9	7	65
23	13	20	8	6	70
24	13	30	7	5	70
25	13	30	6	4	80
26	14	40	5	3	85
27	15	40	4	2	90
28	15	50	3	1	90
29	16	50	2	1	95
30	16	60	1	1	100

SELECT

```
REGR_INTERCEPT(col1, col2) AS RICol1#2  
,REGR_INTERCEPT(col1, col3) AS RICol1#3  
,REGR_INTERCEPT(col1, col4) AS RICol1#4  
,REGR_INTERCEPT(col1, col5) AS RICol1#5  
,REGR_INTERCEPT(col1, col6) AS RICol1#6  
FROM Stats_Table ;
```

RICol1#2	RICol1#3	RICol1#4	RICol1#5	RICol1#6
-1	3	31	18	-1

A regression line is a line of best fit, drawn through a set of points on a graph for X and Y coordinates. It uses the Y coordinate as the Dependent Variable and the X value as the Independent Variable.

Two regression lines always meet or intercept at the mean of the data points(x,y), where x=AVG(x) and y=AVG(y) and is not usually one of the original data points.

Another REGR_INTERCEPT Example so you can Compare

Stats_Table

Col1	Col2	Col3	Col4	Col5	Col6
1	1	1	30	1	0
2	1	1	29	2	5
3	3	10	28	3	10
4	3	10	27	4	15
5	3	10	26	5	20
6	4	10	25	6	30
7	5	10	24	7	30
8	5	10	23	8	30
9	5	10	22	9	35
10	5	20	21	10	35
11	7	20	20	22	40
12	7	20	19	12	40
13	9	20	18	13	45
14	9	20	17	14	45
15	9	20	16	15	50
16	9	20	15	14	55
17	10	20	14	13	55
18	10	20	13	12	60
19	10	20	12	11	60
20	10	20	11	9	65
21	10	20	10	8	65
22	10	20	9	7	65
23	13	20	8	6	70
24	13	30	7	5	70
25	13	30	6	4	80
26	14	40	5	3	85
27	15	40	4	2	90
28	15	50	3	1	90
29	16	50	2	1	95
30	16	60	1	1	100

SELECT

```
REGR_INTERCEPT(col4, col2) AS RICol4#2  
,REGR_INTERCEPT(col4, col3) AS RICol4#3  
,REGR_INTERCEPT(col4, col1) AS RICol4#1  
,REGR_INTERCEPT(col4, col5) AS RICol4#5  
,REGR_INTERCEPT(col4, col6) AS RICol4#6  
FROM Stats_Table ;
```

RICol4#2	RICol4#3	RICol4#1	RICol4#5	RICol4#6
32	28	0	13	32

A regression line is a line of best fit, drawn through a set of points on a graph for X and Y coordinates. It uses the Y coordinate as the Dependent Variable and the X value as the Independent Variable.

Two regression lines always meet or intercept at the mean of the data points(x,y), where x=AVG(x) and y=AVG(y) and is not usually one of the original data points.

The REGR_SLOPE Function

```
SELECT REGR_SLOPE(col1, col2) AS RSCol1#2  
FROM Stats_Table;
```

A regression line is a line of best fit, drawn through a set of points on a graph of X and Y coordinates. It uses the Y coordinate as the Dependent Variable and the X value as the Independent Variable.

The slope of the line is the angle at which it moves on the X and Y coordinates. The vertical slope is Y on X and the horizontal slope is X on Y.

Syntax for using REGR_SLOPE:

REGR_SLOPE(dependent-expression, independent-expression)

A REGR_SLOPE Example

Stats_Table

Col1	Col2	Col3	Col4	Col5	Col6
1	1	1	30	1	0
2	1	1	29	2	5
3	3	10	28	3	10
4	3	10	27	4	15
5	3	10	26	5	20
6	4	10	25	6	30
7	5	10	24	7	30
8	5	10	23	8	30
9	5	10	22	9	35
10	5	20	21	10	35
11	7	20	20	22	40
12	7	20	19	12	40
13	9	20	18	13	45
14	9	20	17	14	45
15	9	20	16	15	50
16	9	20	15	14	55
17	10	20	14	13	55
18	10	20	13	12	60
19	10	20	12	11	60
20	10	20	11	9	65
21	10	20	10	8	65
22	10	20	9	7	65
23	13	20	8	6	70
24	13	30	7	5	70
25	13	30	6	4	80
26	14	40	5	3	85
27	15	40	4	2	90
28	15	50	3	1	90
29	16	50	2	1	95
30	16	60	1	1	100

SELECT

REGR_SLOPE(col1, col2) AS RSCol1#2
,REGR_SLOPE(col1, col3) AS RSCol1#3
,REGR_SLOPE(col1, col4) AS RSCol1#4
,REGR_SLOPE(col1, col5) AS RSCol1#5
,REGR_SLOPE(col1, col6) AS RSCol1#6

FROM Stats_Table ;

RSCol1#2	RSCol1#3	RSCol1#4	RSCol1#5	RSCol1#6
2	1	-1	-0	0

A regression line is a line of best fit, drawn through a set of points on a graph of X and Y coordinates. It uses the Y coordinate as the Dependent Variable and the X value as the Independent Variable.

The slope of the line is the angle at which it moves on the X and Y coordinates. The vertical slope is Y on X and the horizontal slope is X on Y.

Another REGR_SLOPE Example so you can Compare

Stats_Table

Col1	Col2	Col3	Col4	Col5	Col6
1	1	1	30	1	0
2	1	1	29	2	5
3	3	10	28	3	10
4	3	10	27	4	15
5	3	10	26	5	20
6	4	10	25	6	30
7	5	10	24	7	30
8	5	10	23	8	30
9	5	10	22	9	35
10	5	20	21	10	35
11	7	20	20	22	40
12	7	20	19	12	40
13	9	20	18	13	45
14	9	20	17	14	45
15	9	20	16	15	50
16	9	20	15	14	55
17	10	20	14	13	55
18	10	20	13	12	60
19	10	20	12	11	60
20	10	20	11	9	65
21	10	20	10	8	65
22	10	20	9	7	65
23	13	20	8	6	70
24	13	30	7	5	70
25	13	30	6	4	80
26	14	40	5	3	85
27	15	40	4	2	90
28	15	50	3	1	90
29	16	50	2	1	95
30	16	60	1	1	100

SELECT

```
REGR_SLOPE(col4, col2) AS RSofCol1#2
,REGR_SLOPE(col4, col3) AS RSofCol1#3
,REGR_SLOPE(col4, col1) AS RSofCol1#4
,REGR_SLOPE(col4, col5) AS RSofCol1#5
,REGR_SLOPE(col4, col6) AS RSofCol1#6
FROM Stats_Table ;
```

RSCol4#2	RSCol4#3	RSCol4#1	RSCol4#5	RSCol4#6
-2	-1	1	0	-0

A regression line is a line of best fit, drawn through a set of points on a graph of X and Y coordinates. It uses the Y coordinate as the Dependent Variable and the X value as the Independent Variable.

The slope of the line is the angle at which it moves on the X and Y coordinates. The vertical slope is Y on X and the horizontal slope is X on Y.

Using GROUP BY

Stats_Table

Col1	Col2	Col3	Col4	Col5	Col6
1	1	1	30	1	0
2	1	1	29	2	5
3	3	10	28	3	10
4	3	10	27	4	15
5	3	10	26	5	20
6	4	10	25	6	30
7	5	10	24	7	30
8	5	10	23	8	30
9	5	10	22	9	35
10	5	20	21	10	35
11	7	20	20	22	40
12	7	20	19	12	40
13	9	20	18	13	45
14	9	20	17	14	45
15	9	20	16	15	50
16	9	20	15	14	55
17	10	20	14	13	55
18	10	20	13	12	60
19	10	20	12	11	60
20	10	20	11	9	65
21	10	20	10	8	65
22	10	20	9	7	65
23	13	20	8	6	70
24	13	30	7	5	70
25	13	30	6	4	80
26	14	40	5	3	85
27	15	40	4	2	90
28	15	50	3	1	90
29	16	50	2	1	95
30	16	60	1	1	100

SELECT

col3	,count(*)	AS Cnt
	,avg(col1)	AS Avg1
	,stddev_pop(col1)	AS SD1
	,var_pop(col1)	AS VP1
	,avg(col4)	AS Avg4
	,stddev_pop(col4)	AS SD4
	,var_pop(col4)	AS VP4
	,avg(col6)	AS Avg6
	,stddev_pop(col6)	AS SD6
	,var_pop(col6)	AS VP6

FROM Stats_Table

GROUP BY 1

ORDER BY 1;

Col3	Cnt	Avg1	SD1	VP1	Avg4	SD4	VP4	Avg6	SD6	VP6
1	2	2	0	0	30	0	0	2	2	6
10	7	6	2	4	25	2	4	24	9	74
20	14	16	4	16	14	4	16	54	11	116
30	2	24	0	0	6	0	0	75	5	25
40	2	26	0	0	4	0	0	88	2	6
50	2	28	0	0	2	0	0	92	2	6
60	1	30	0	0	1	0	0	100	0	0

No Having Clause Vs Use of HAVING

```

SELECT col3
      ,count(*) AS Cnt
      ,avg(col1) AS Avg1
      ,stddev_pop(col1) AS SD1
      ,var_pop(col1) AS VP1
      ,avg(col4) AS Avg4
      ,stddev_pop(col4) AS SD4
      ,var_pop(col4) AS VP4
      ,avg(col6) AS Avg6
      ,stddev_pop(col6) AS SD6
      ,var_pop(col6) AS VP6
FROM Stats_Table
GROUP BY 1
ORDER BY 1;
  
```

```

SELECT col3
      ,count(*) AS Cnt
      ,avg(col1) AS Avg1
      ,stddev_pop(col1) AS SD1
      ,var_pop(col1) AS VP1
FROM Stats_Table
GROUP BY 1
ORDER BY 1
HAVING Cnt > 2 and VP1 < 20;
  
```

Col3	Cnt	Avg1	SD1	VP1	Avg4	SD4	VP4	Avg6	SD6	VP6
1	2	2	0	0	30	0	0	2	2	6
10	7	6	2	4	25	2	4	24	9	74
20	14	16	4	16	14	4	16	54	11	116
30	2	24	0	0	6	0	0	75	5	25
40	2	26	0	0	4	0	0	88	2	6
50	2	28	0	0	2	0	0	92	2	6
60	1	30	0	0	1	0	0	100	0	0

col3	Cnt	Avg1	SD1	VP1
10	7	6	2	4
20	14	16	4	16

The example (above right) uses HAVING to perform a compound comparison on both the count and the covariance

Chapter 37

Explain

A rule to live by: I won't use anything I can't explain in five minutes.

- Phil Crosby

Table of Contents Chapter 37 - Explain

- [EXPLAIN Keywords](#)
- [EXPLAIN Keywords Continued](#)
- [Explain Example – Full Table Scan](#)
- [Explain Example – Unique Primary Index \(UPI\)](#)
- [Explain Example – Non-Unique Primary Index \(NUPI\)](#)
- [Explain Example – Unique Secondary Index \(USI\)](#)
- [Explain Example – Repartitioned to All-AMPs](#)
- [Explain Example – Row Hash Match Scan](#)
- [Explain Example – Duplicated on All-AMPs](#)
- [Explain Example – Low Confidence](#)
- [Explain Example – High Confidence](#)
- [Explain Example – Product Join](#)
- [Explain Example – BMSMS](#)
- [Explain Terminology for Partitioned Primary Index Tables](#)
- [Explain Example – From a Single Partition](#)
- [Explain Example – From N Partitions](#)
- [Explain Example – Partitions and Current Date](#)

EXPLAIN Keywords

Locking Pseudo Table	Serial lock on a symbolic table. Every table has one. Used to prevent deadlocks situations between users.
Locking table for	Indicates that an ACCESS, READ, WRITE, or EXCLUSIVE lock has been placed on the table
Locking rows for <type>	Indicates that an ACCESS, READ, or WRITE, lock is placed on rows read or written
Do an ABORT test	Guarantees a transaction is not in progress for this user
All AMPs retrieve	All AMPs are receiving the AMP steps and are involved in providing the answer set
By way of an all rows scan	Rows are read sequentially on all AMPs
By way of primary index	Rows are read using the Primary index column(s)
By way of index number	Rows are read using the Secondary index – number from HELP INDEX
BMSMS	Bit Map Set Manipulation Step, alternative direct access technique when multiple NUSI columns are referenced in the WHERE clause
Residual conditions	WHERE clause conditions, other than those of a join
Eliminating duplicate rows	Providing unique values, normally result of DISTINCT, GROUP BY or subquery
Where unknown comparison will be ignored	Indicates that NULL values will not compare to a TRUE or FALSE. Seen in a subquery using NOT IN or NOT = ALL because no rows will be returned on ignored comparison.
Nested join	The fastest join possible. It uses a UPI to retrieve a single row after using a UPI or a USI in the WHERE to reduce the join to a single row.

EXPLAIN Keywords Continued

Merge join	Rows of one table are matched to the other table on common domain columns after being sorted into the same sequence, normally Row Hash
Product join	Rows of one table are matched to all rows of another table with no concern for domain match
ROWID join	A very fast join. It uses the ROWID of a UPI to retrieve a single row after using a UPI or a USI in the WHERE to reduce the join to a single row.
Duplicated on all AMPS	Participating rows for the table (normally smaller table) of a join are duplicated on all AMPS
Hash redistributed on all AMPs	Participating rows of a join are hashed on the join column and sent to the same AMP that stores the matching row of the table to join
SMS	Set Manipulation Step, result of an INTERSECT, UNION, EXCEPT or MINUS operation
Last use	SPOOL file is no longer needed after the step and space is released
Built locally on the AMPs	As rows are read, they are put into SPOOL on the same AMP
Aggregate Intermediate Results computed locally	The aggregation values are all on the same AMP and therefore no need to redistribute them to work with rows on other AMPs
Aggregate Intermediate Results computed globally	The aggregation values are not all on the same AMP and must be redistributed on one AMP, to accompany the same value with from the other AMPs

Explain Example – Full Table Scan

```
EXPLAIN SELECT * FROM Employee_Table ;
```

- 1) First, we lock a distinct SQL_CLASS."pseudo table" for read on a RowHash to prevent global deadlock for SQL_CLASS.Employee_Table.
- 2) Next, we lock SQL_CLASS.Employee_Table for read.
- 3) We do an **all-AMPs RETRIEVE** step from SQL_CLASS.Employee_Table by way of an **all-rows scan** with no residual conditions into Spool 1 (group_amps), which is built locally on the AMPs. The size of Spool 1 is estimated with low confidence to be 6 rows (342 bytes).
The estimated time for this step is 0.03 seconds.
- 4) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.
-> The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 0.03 seconds.

When you see an all-AMPs RETRIEVE by way of an all-rows scan that means that Teradata is doing a Full Table Scan. All AMPs are retrieving all the rows they own.

Explain Example – Unique Primary Index (UPI)

```
EXPLAIN SELECT * FROM Employee_Table  
WHERE Employee_No = 2000000;
```

- 1) First, we do a **single-AMP RETRIEVE** step from
SQL_CLASS.Employee_Table by way of the **unique primary index**
"SQL_CLASS.Employee_Table.Employee_No = 2000000" with no residual
conditions. The estimated time for this step is 0.01 seconds.
-> The row is sent directly back to the user as the result of
statement 1. The total estimated time is 0.01 seconds.

If you use the Primary Index column in the WHERE clause you will most likely get a Single-AMP retrieve by way of the Unique Primary Index. This is the fastest query!

Explain Example – Non-Unique Primary Index (NUPI)

```
EXPLAIN SELECT * FROM Sales_Table  
WHERE Product_ID = 1000 ;
```

- 1) First, we do a **single-AMP RETRIEVE** step from SQL_CLASS.Sales_Table by way of the **primary index** "SQL_CLASS.Sales_Table.Product_ID = 1000" with no residual conditions into Spool 1 (one-amp), which is built locally on that AMP. The size of Spool 1 is estimated with low confidence to be 2 rows (66 bytes). The estimated time for this step is 0.02 seconds.
-> The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 0.02 seconds.

If you use the Primary Index column in the WHERE clause you will most likely get a Single-AMP retrieve. This example utilized a Non-Unique Primary Index (NUPI). It doesn't get much faster than this unless it utilizes a Unique Primary Index (UPI).

Explain Example – Unique Secondary Index (USI)

```
CREATE UNIQUE INDEX (First_Name, Last_Name) on Employee_Table ;
```

```
EXPLAIN SELECT * FROM Employee_Table  
    WHERE First_Name = 'Lorraine'  
        AND Last_Name = 'Larkins' ;
```

Created a Unique
Secondary Index

- 1) First, we do a **two-AMP RETRIEVE** step from SQL_CLASS.Employee_Table by way of **unique index # 12 "SQL_CLASS.Employee_Table.Last_name = 'Larkins'**, SQL_CLASS.Employee_Table.First_name = 'Lorraine'" with no residual conditions. The estimated time for this step is 0.01 seconds.
-> The row is sent directly back to the user as the result of statement 1. The total estimated time is 0.01 seconds.

Using a **UNIQUE Secondary Index (USI)** in the **WHERE** clause will result in a two-AMP Retrieve every time. This is very fast.

Explain Example – Redistributed to All-AMPs

```
EXPLAIN SELECT E.* , D.*  
          FROM Employee_Table as E  
          INNER JOIN  
          Department_Table as D  
          ON E.Dept_No = D.Dept_No ;
```

- 4) We do an all-AMPs RETRIEVE step from SQL_CLASS.E by way of an all-rows scan with a condition of ("NOT (SQL_CLASS.E.Dept_No IS NULL)") into Spool 2 (all_amps), which is redistributed by the hash code of (SQL_CLASS.E.Dept_No) to all AMPs. Then we do a SORT to order Spool 2 by row hash. The size of Spool 2 is estimated with low confidence to be 6 rows (294 bytes). The estimated time for this step is 0.01 seconds.
- 5) We do an all-AMPs JOIN step from SQL_CLASS.D by way of a RowHash match scan, which is joined to Spool 2 (Last Use) by way of a RowHash match scan. SQL_CLASS.D and Spool 2 are joined using a merge join, with a join condition of ("Dept_No = SQL_CLASS.D.Dept_No").

Data is often redistributed on a Join to ensure that the matching rows are on the same physical AMPs in Spool. If you see the word redistributed then data is being moved!

Explain Example – Row Hash Match Scan

```
EXPLAIN SELECT E2.* , D.*  
          FROM Employee_Table2 as E2  
          INNER JOIN  
          Department_Table as D  
          ON E2.Dept_No = D.Dept_No ;
```

- 4) We do an **all-AMPs JOIN** step from SQL_CLASS.D by way of a **RowHash match scan**, which is joined to SQL_CLASS.E2 by way of a **RowHash match scan**. SQL_CLASS.D and SQL_CLASS.E2 are joined using a merge join, with a join condition of ("SQL_CLASS.E2.Dept_No = SQL_CLASS.D.Dept_No"). The result goes into Spool 1 (group_amps), which is built locally on the AMPs. The size of Spool 1 is estimated with low confidence to be 8 rows (728 bytes). The estimated time for this step is 0.04 seconds.
- 5) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.

When you see the words Row Hash Match Scan then the matching rows are on the same physical AMP in Spool and the join takes place. This is what you want to see for Joins, and it is even better when data isn't moved before hand.

Explain Example – Duplicated on All-AMPs

```
EXPLAIN SELECT E.*, D.*  
          FROM Employee_Table      as E,  
                Department_Table2  as D  
         WHERE E.Dept_No = D.Dept_No;
```

4) We execute the following steps in parallel.

1) We do an **all-AMPs RETRIEVE** step from SQL_CLASS.d by way of an all-rows scan with a condition of ("NOT (SQL_CLASS.d.Dept_No IS NULL)") into Spool 2 (all_amps), which is **duplicated on all AMPs**. Then we do a SORT to order Spool 2 by the hash code of (SQL_CLASS.d.Dept_No). The size of Spool 2 is estimated with low confidence to be 16 rows (752 bytes). The estimated time for this step is 0.01 seconds.

Data is often redistributed on a Join to ensure that the matching rows are on the same physical AMPS in Spool. If you see the word redistributed then data is being moved! This means that the smaller table (usually) is duplicated on each AMP. The Parsing Engine decided that this was a less costly approach than redistributing the data. This is sometimes called a Big Table/Small Table Join.

Explain Example –Low Confidence

```
EXPLAIN SELECT * FROM Addresses;
```

- 1) First, we lock a distinct SQL_CLASS."pseudo table" for read on a RowHash to prevent global deadlock for SQL_CLASS.Addresses.
- 2) Next, we lock SQL_CLASS.Addresses for read.
- 3) We do an all-AMPs RETRIEVE step from SQL_CLASS.Addresses by way of an all-rows scan with no residual conditions into Spool 1 (group_amps), which is built locally on the AMPs. The size of Spool 1 is estimated with low confidence to be 2 rows (114 bytes). The estimated time for this step is 0.03 seconds.
- 4) Finally, we send out an END TRANSACTION step to all AMPS involved in processing the request.
-> The contents of Spool 1 are sent back to the user as the result of statement 1. The total estimated time is 0.03 seconds.

The EXPLAIN plan estimates 2 rows, but there are actually 5 rows coming back. When the Explain plan shows low confidence it is often because there are no COLLECT STATISTICS on the table, so the PE estimates. Statistics collection is often done by the DBA. The next slide will COLLECT STATISTICS and retry the query.

Explain Example – High Confidence

1

COLLECT STATISTICS on Addresses
COLUMN Subscriber_No ;

2

EXPLAIN SELECT * FROM Addresses;

- 3) We do an all-AMPs RETRIEVE step from SQL_CLASS.Addresses by way of an all-rows scan with no residual conditions into Spool 1 (group_amps), which is built locally on the AMPs. The size of Spool 1 is estimated with **high confidence** to be **5 rows** (285 bytes). The estimated time for this step is 0.03 seconds.
- 4) Finally, we send out an END TRANSACTION step to all AMPS involved in processing the request.

The EXPLAIN plan now estimates 5 rows and shows High confidence because there are COLLECT STATISTICS on the table. If No Statistics are on the table the Parsing Engine will make an estimate after sampling a Random AMP.

Explain Example – Product Join

```
EXPLAIN SELECT *
  FROM Student_Table S, Course_Table C, Student_Course_Table SC
 WHERE s.Student_Id = sc.Student_Id ;
```

- 6) We do an all-AMPs JOIN step from SQL_CLASS.C by way of an all-rows scan with no residual conditions, which is joined to Spool 2 (Last Use) by way of an all-rows scan. SQL_CLASS.C and Spool 2 are joined using a **product join**, with a join condition of **"(1=1)"**. The result goes into Spool 1 (group_amps), which is built locally on the AMPs. The size of Spool 1 is estimated with low confidence to be 96 rows (7,584 bytes). The estimated time for this step is 0.05 seconds.

The product join in step 6 is using **(1=1)** as the join condition where it should be a merge join. Therefore, this is a **Cartesian product join**. A careful analysis of the SELECT shows a single join condition in the WHERE clause. However, this is a three-table join and should have two join conditions. The WHERE clause needs to be fixed and by using the EXPLAIN we have saved valuable time.

Explain Example – BMSMS

```
EXPLAIN SELECT * From Employee_Table as E  
WHERE Salary = 36000.00  
AND Dept_No = 400;
```

- 3) We do a **BMSMS** (bit map set manipulation) step that builds a bit map for Employee_Table by way of index # 4 Salary = 360000.00" which is placed in Spool 2. The estimated time for this step is 0.01 seconds.
- 4) We do an all-AMPs RETRIEVE step from E by way of index # 8 E.Dept_No= 400" and the bit map in Spool 2 (Last Use) with a residual condition of ("E.Salary = 36000.00") into Spool 1 (group_amps), which is built locally on the AMPS. The size of Spool 1 is estimated with low confidence to be 50 rows (4620 bytes). The estimated time for this step is 0.02 seconds.

BMSMS (Bit Map Set Manipulation Step) is an excellent way to process large tables. This basically occurs when multiple columns are ANDed together with each Column being a Non-Unique Secondary Index (NUSI). This usually won't happen unless STATISTICS were collected on the table.

Explain Terminology for Partitioned Primary Index Tables

“A single partition of“ means that an AMP will access a single partition of a table. This is called Partition Elimination. Only a small slice of the table is read.

“N partition of“ means that an AMP will access N partitions of a table. This is called also considered Partition Elimination. The smaller the number N is the faster the query will be compared to the usual Full Table Scan.

“SORT to partition Spool m by RowKey“ means the spool is to be sorted by RowKey, which constitutes the Partition Number and the Hash of the Primary Index. This is done usually for a Join.

“A RowKey-based“ means an equality join on the RowKey, which is similar to the Row Hash Match Scan for Non-PPI tables. The Join is taking place.

“Enhanced by dynamic partition“ means a join condition where dynamic partition elimination has been used.

Partitioned Primary Index (PPI) tables are tables that are not sorted by Row-Hash on each AMP, but instead sorted first by the Partition. This is designed to eliminate full table scans on Range Queries. When the explain shows Partition Elimination then a Full Table Scan is NOT being performed, which is the entire purpose of PPI Tables.

Explain Example – From a Single Partition

```
EXPLAIN SELECT * FROM Order_PPI  
WHERE Order_Date = '1998-05-04'
```

- 3) We do an all-AMPs RETRIEVE step from a single partition of SQL_CLASS.Order_PPI with a condition of ("SQL_CLASS.Order_PPI.Order_Date = DATE '1998-05-04'") with a residual condition of ("SQL_CLASS.Order_PPI.Order_Date = DATE '1998-05-04'") into Spool 1 (group_amps), which is built locally on the AMPs. The size of Spool 1 is estimated with no confidence to be 1 row (41 bytes). The estimated time for this step is 0.02 seconds.

The above example uses all-AMPs, but each AMP only reads a single partition. This is as good as it gets with a Partition Primary Index(PPI) Table.

Explain Example – From N Partitions

```
EXPLAIN SELECT * FROM Order_PPI  
WHERE Order_Date BETWEEN '1998-05-01' and '1998-05-31' ;
```

- 3) We do an all-AMPs RETRIEVE step from 31 partitions of SQL_CLASS.Order_PPI with a condition of ("SQL_CLASS.Order_PPI.Order_Date <= DATE '1998-05-31') AND (SQL_CLASS.Order_PPI.Order_Date >= DATE '1998-05-01')") into Spool 1 (group_amps), which is built locally on the AMPs. The size of Spool 1 is estimated with no confidence to be 2 rows (82 bytes). The estimated time for this step is 0.03 seconds.
- 4) Finally, we send out an END TRANSACTION step to all AMPs involved in processing the request.

The above example uses all-AMPs, but each AMP only 31 partitions. This has eliminated reading the entire table, and dramatically improved the performance on Range Queries like the example above.

Explain Example – Partitions and Current_Date

```
EXPLAIN SELECT * FROM Order_PPI  
    WHERE Order_Date = Current_Date ;
```

- 3) We do an all-AMPs RETRIEVE step from a single partition of SQL_CLASS.Order_PPI with a condition of ("SQL_CLASS.Order_PPI.Order_Date = DATE '2012-06-18'") with a residual condition of ("SQL_CLASS.Order_PPI.Order_Date = DATE '2012-06-18'") into Spool 1 (group_amps), which is built locally on the AMPs. The size of Spool 1 is estimated with no confidence to be 1 row (41 bytes). The estimated time for this step is 0.02 seconds.

The above example uses all-AMPs, but each AMP reads only one partition. What is different about this is the improvement that Teradata has made to Current_Date. This happened in Teradata V12 and is a welcomed addition.

Chapter 38

Collect Statistics

“The non permanent appearance of happiness and distress, and their disappearance in due course, are like the appearance and disappearance of summer and winter seasons.”

- Bhagavad Gita

Table of Contents Chapter 38 - Collect Statistics

- [The Purpose of Collect Statistics](#)
- [What to COLLECT STATISTICS On?](#)
- [How do you know if Statistics were Collected on a Table?](#)
- [The Basic Syntax for COLLECT STATISTICS](#)
- [COLLECT STATISTICS Examples for a better Understanding](#)
- [The Official Syntax for COLLECT STATISTICS](#)
- [How to Re-COLLECT STATISTICS on a Table](#)
- [How does the PE Plan if No Statistics were Collected?](#)
- [How to Copy a Table with Data and the Statistics?](#)
- [How to Copy a Table with NO Data and the Statistics?](#)
- [When to COLLECT STATISTICS Using only a SAMPLE](#)
- [Examples of COLLECT STATISTICS Using only a SAMPLE](#)
- [How to Collect Statistics on a PPI Table on the Partition](#)
- [Teradata V12 and V13 Statistics Enhancements](#)

The Purpose of Collect Statistics

The Teradata Parsing Engine (PE) also called the Optimizer is in charge of creating the PLAN for the AMPs to follow. The PE works best when Statistics have been collected on a table. Then it knows:

1. The **number of rows** in the table
 2. The **average row size**
 3. Information on all **Indexes** in which statistics were collected
 4. The **range of values** for the column(s) in which statistics were collected
 5. The **number of rows per value** for the column(s) in which statistics were collected
 6. The number of **NULLs** for the column(s) in which statistics were collected
-

The purpose of the COLLECT STATISTICS command is to gather and store demographic data for one or more columns or indices of a table or join index. This process computes a statistical profile of the collected data, and stores the synopsis in the Data Dictionary (DD) inside USER DBC for use during the PE's optimizing phase of SQL statement parsing. The optimizer uses this synopsis data to generate efficient table access and join plans. Do NOT COLLECT Statistics on all columns in the table.

What to COLLECT STATISTICS On?

You don't COLLECT STATISTICS on all columns and indexes because it takes up too much space for unnecessary reasons, but you do collect on:

- All Non-Unique indices
- Non-index join columns
- The Primary Index of small tables
- Primary Index of a Join Index
- Secondary Indices defined on any join index
- Join index columns that frequently appear on any additional join index
- Columns that frequently appear in WHERE search conditions
- Columns that frequently appear in WHERE search conditions or in the WHERE clause of joins.

The first time you collect statistics you collect them at the index or column level. After that you just collect statistics at the table level and all previous columns collected previously are collected again. It is a mistake to collect statistics only once and then never do it again. COLLECT STATISTICS each time a table's data changes by 10%.

How do you know if Statistics were Collected on a Table?

Syntax: HELP Statistics <Table Name>

1

Help Statistics Hierarchy_Table ;

ERROR [HY000] [Teradata][ODBC Teradata Driver][Teradata Database]

There are no statistics defined for the table.
HELP STATISTICS Command Failed.



Careful: This looks like an error, but it is merely stating No Statistics Collected!

2

Help Statistics Employee_Table ;

Date	Time	Unique Values	Column Names
12/06/19	20:50:29	9	Employee_No
12/06/19	20:50:30	6	Dept_No
12/06/19	20:50:30	9	Last_Name, First_Name

The HELP Statistics command will show you what statistics have been collected.

The Basic Syntax for COLLECT STATISTICS

Here is the syntax for collecting on columns and indexes.

1

COLLECT STATISTICS on <Tablename>
COLUMN <Column Name>;

2

COLLECT STATISTICS on <Tablename>
INDEX (<Column Name(s)>);

Below are three actual examples

1

COLLECT STATISTICS on Employee_Table
COLUMN Employee_No ;

2

COLLECT STATISTICS on Employee_Table
COLUMN Dept_No ;

3

COLLECT STATISTICS on Employee_Table
INDEX(First_Name, Last_Name);

The COLLECT STATISTICS commands above are excellent examples.

COLLECT STATISTICS Examples for a better Understanding

```
COLLECT STATISTICS on Employee_Table  
COLUMN Dept_No ;
```

Here is how you COLLECT STATISTICS on a single **column**.

```
COLLECT STATISTICS on Employee_Table  
COLUMN Employee_No ;
```

Here is how you COLLECT STATISTICS on a single **Index**.

```
COLLECT STATISTICS on Employee_Table  
INDEX(First_Name, Last_Name);
```

Here is how to COLLECT STATISTICS on a **Multi-column Index**.

```
COLLECT STATISTICS on Employee_Table  
Column (Employee_No, Dept_No)
```

Here is how to COLLECT STATISTICS on **multiple columns** that are often used together in the **SQL WHERE Clause**.

```
COLLECT STATISTICS on Employee_Table
```

Here is how to Refresh STATISTICS on **columns and indexes** previously collected.

The Official Syntax for COLLECT STATISTICS

1

Collect Statistics on a Table

```
COLLECT STATISTICS [ USING SAMPLE ]
  ON [ TEMPORARY ] { <table-name> | <join-index-name> | <hash-index-name>
}
[ COLUMN { <column-name> | (<column-list>) }
| [ UNIQUE ] INDEX { <index-name> [ ALL ] | (<column-list>) }
  [ ORDER BY { HASH | VALUES } [ <column-name> ] ] ] ;
```

2

Collect Statistics on an Index

```
COLLECT STATISTICS [ USING SAMPLE ]
[ COLUMN { <column-name> | (<column-list>) }
| [ UNIQUE ] INDEX { <index-name> [ ALL ] | (<column-list>) }
  [ ORDER BY { HASH | VALUES } [ <column-name> ] ] ]
ON [ TEMPORARY ] { <table-name> | <join-index-name> | <hash-index-name> } ;
```

How to Re-COLLECT STATISTICS on a Table

Here is the syntax for re-collecting statistics on a table.

COLLECT STATISTICS on <Tablename> ;

Below is an actual example

COLLECT STATISTICS on Employee_Table;



This will **NOT** Collect on
all Columns in the table, but
only on the Columns and
Indexes previously collected.

The first time you collect statistics you do it for each individual column or index that you want to collect on. When a table changes its data by 10% due to Inserts, Updates, or Deletes you merely use the command above and it re-collects on the same columns and indexes previously collected on.

How does the PE Plan if No Statistics were Collected?

If the Parsing Engine finds there are NO Statistics on a table being queried it runs a **Random AMP Sample** and then Guesstimates!

A Random AMP is selected for a Random Sample. Two things happen:

- 1) Indexes are sampled on the Random AMP and the PE estimates based on the total number of AMPS in the system.
- 2) If a column in the WHERE clause of the SQL is not an Index the PE assumes that 10% of the rows will come back. If two columns are in the WHERE clause then it assumes 7.5% of the rows will come back. If three columns are in the WHERE Clause it assumes 5%.

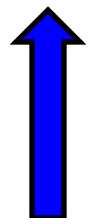
Today's Teradata systems always perform a Random AMP Sample even if tables have Statistics. Then they compare the Random AMP Sample with the statistics to determine if the statistics are **stale**.

A Random AMP sample is selected by the PE to perform a Random AMP Sample if the PE finds there are no statistics on the table. Bad distribution could be bad here!

How to Copy a Table with Data and the Statistics?

This next example is pretty amazing. Assume that the original Employee_Table had COLLECT STATISTICS on the columns Employee_No and Dept_No. The New table we have called Employee_Table_New will have DDL exactly like the Employee_Table plus data plus the statistics. Yes, the exact same statistics will be copied to the new table. Below is the actual example!

```
CREATE TABLE Employee_Table_New AS Employee_Table  
with DATA  
AND Statistics;
```



The example above will CREATE a new table called Employee_Table_New and it will have the exact same DDL as the Employee_Table, the exact same data, and the exact same statistics.

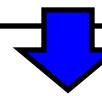
How to Copy a Table with NO Data and the Statistics?

This next example is clever. Assume that the original Employee_Table had COLLECT STATISTICS on the columns Employee_No and Dept_No. The New table we have called Employee_Table_99 will have DDL exactly like the Employee_Table but NO data. It will have the Statistics, but they will be Zeroed Statistics.

```
CREATE TABLE Employee_Table_99 AS Employee_Table  
with NO DATA  
AND Statistics;
```

Table DDL copied with NO Data and Zeroed Statistics
on the columns Employee_No and Dept_No.

Once you have loaded Employee_Table_99 with the data you want by using BTEQ, FastLoad, an INSERT/SELECT, Nexus, or the ETL tool of your choice you can Recollect Statistics!



COLLECT STATISTICS ON Employee_Table_99 ;

You have just Re-Collected Statistics on Employee_Table_99 for the columns Employee_No and Dept_No. The easy re-collection on the columns previously collected on (after the data is loaded) was the entire purpose of getting the Zeroed Statistics in the first place. Make sure you recollect after your data is loaded though!

When to COLLECT STATISTICS Using only a SAMPLE

You might consider Collecting Statistics with SAMPLE if:

- 1 You are collecting statistics on a very large table.
- 2 When collecting statistics becomes a problem with system performance or cost because the system is so busy.

Don't consider Collecting Statistics with SAMPLE if:

- 1 The tables are small.
- 2 To replace all existing full scan Collect Statistics.
- 3 If the column's data is skewed badly.

COLLECT STATISTICS can be very time consuming because it performs a full table scan and then performs a lot of statistical calculations. Because Collect Statistics runs infrequently and benefits query optimization it is considered a necessary task. Without statistics, query performance will suffer. The bad news about sampled statistics is that they may not be as accurate, which could negatively affect the PE's plans. In most cases sampled statistics are better than no statistics. Don't use Sample unless necessary!

Examples of COLLECT STATISTICS Using only a SAMPLE

COLLECT STATISTICS USING SAMPLE on Employee_Table COLUMN Dept_No ;

Here is how you COLLECT STATISTICS on a single column.

COLLECT STATISTICS USING SAMPLE on Employee_Table COLUMN Employee_No ;

Here is how you COLLECT STATISTICS on a single Index.

COLLECT STATISTICS USING SAMPLE on Employee_Table INDEX(First_Name, Last_Name);

Here is how to COLLECT STATISTICS on a Multi-column Index.

COLLECT STATISTICS on Employee_Table ;

Here is how to Refresh STATISTICS on a Table.

If you recollect statistics on a sample it recollects with the same sample!

Sampled statistics are generally more accurate for data that is not skewed. For example, columns or indexes that are unique or nearly unique are not skewed. Because the PE needs to be aware of skewed data you should not collect with sample on skewed data. That is why sampling is generally more appropriate for indexes than non-indexed column(s). If you recollect statistics on a Sample it Recollects with the same Sample!

How to Collect Statistics on a PPI Table on the Partition

Here is the syntax for collecting statistics on a PPI table.

`COLLECT STATISTICS on <Tablename> COLUMN PARTITION;`

Below is an actual example

`COLLECT STATISTICS on Order_Table_PPI COLUMN PARTITION;`

Three reasons to Collect on the Partition:

- 1 The Parsing Engine will have a better plan for PPI Tables.
- 2 This helps the most with Partition Elimination on Range Queries.
- 3 This is especially helpful when a table has a lot of empty partitions.

The Parsing Engine can use this information to better estimate the query cost when there are a significant number of empty partitions. If PARTITION statistics are not collected, empty partitions may cause the Parsing Engine to underestimate the number of rows in a partition. You shouldn't use WITH SAMPLE to collect on Partitions.

Teradata V12 and V13 Statistics Enhancements

In V12 Extrapolate Statistics is designed to more accurately provide for a statistical estimate for date range-based queries that specify a “future” date that is outside the bounds of the current statistics. This results in less re-collections.

In V12 Stale Statistics Detection compares the Random AMP Sample with the statistics collected and determines if they are stale, and should not be used.

In V13 Statistics can now be collected on Volatile Tables.

In V13 PARTITION statistic capabilities have been added to Global Temporary Tables.

In V13 Multi-Column statistics are now available on Hash Indexes and Join Indexes.

In V13 Sample Statistics are available on Tables, Volatile Tables, Global Temporary Tables, Hash Indexes and Join Indexes, including the Partition Columns.

Chapter 39

Hashing Functions

“Love, like a chicken salad or restaurant hash, must be taken with blind faith or it loses its flavor. “

-Helen Rowland

Table of Contents Chapter 39 - Hashing Functions

- [Hashing Functions on Teradata](#)
- [The HASHROW Function](#)
- [The HASHROW Function in a real-world Example](#)
- [The HASHBUCKET Function](#)
- [The HASHBUCKET Function in a real-world Example](#)
- [The HASHAMP Function](#)
- [The HASHAMP Function in a real-world Example](#)
- [A Great HASHAMP Function for Large Tables](#)
- [The HASHBAKAMP Function](#)
- [A Real-World HASBAKHAMP Function Example](#)
- [A Great way to see distribution for Primary and Fallback rows](#)

Hashing Functions on Teradata

Teradata uses Parallel Processing and the most important aspect of this is to spread the rows of a table equally among the AMPs, who read and write data.

Below are the four Hashing Functions available to see the distribution of data:

HASHROW

HASHAMP

HASHBUCKET

HASHBAKAMP

Teradata uses parallel processing with its architecture of AMPs and PEs. The Primary Index (PI) must exist whenever you create a table because it is the sole determinant of which AMPs owns which rows. This concept pertains to data storage and data retrieval. Picking the proper column(s) for the PI is extremely important for distribution and therefore, performance. The hashing functions introduced in this section provide information pertaining to the selection of the AMP where each individual row is stored.

There are hashing functions, which can be incorporated into SQL, to produce and use the same hash value result for testing current or future distribution levels. In other words, these functions can be used to evaluate the distribution of the rows within any or all tables or determine the acceptability of other columns as a potential primary index.

The HASHROW Function

The basic syntax for using the HASHROW function follows:

```
HASHROW( [ <data-column-value> [..., <data-column-value> ... ] ] )
```

Below is an actual example

```
SELECT HASHROW (NULL) AS NULL_HASH  
      ,HASHROW('Coffing') AS Name_HASH  
      ,HASHROW() AS NO_HASH ;
```

NULL_HASH	NAME_HASH	NO_HASH
00000000	4B1075E1	FFFFFFF

If you ran this query over and over again you would get the same result.

Notice that you can Hash a Null and understand that the hash is consistent so running this query with those values will return the same answer time and time again. The HASHROW function is used to produce the 32-bit binary (BYTE(4) data type) Row Hash that is stored as part of the data row. It can return a maximum of 4,294,967,295 unique values. The values produced range from 0 to FFFFFFFF.

The HASHROW Function in a real-world Example

Now that the functionality has been demonstrated on the previous page, a more realistic use might be the following to examine the data distribution and determine the average number of rows per value:

```
SELECT  
COUNT(*)/COUNT(DISTINCT(HASHROW(Student_Id))) AS  
AVG_ROW_CT  
FROM Student_Table;
```

AVG_ROW_CT

1

From the above answer of 1 it is a great sign that the data is perfectly distributed or even Unique. As good as this is, the HASHROW function does not provide a lot more help in the evaluation process. However, when combined with the other Hashing Functions, it yields some very helpful data demographics.

The HASHBUCKET Function

The basic syntax for using the HASHBUCKET function follows:

```
HASHBUCKET( [ <row-hash-value> ] )
```

Below is an actual example

```
SELECT HASHBUCKET(NULL) AS NULL_BUCKET  
      ,HASHBUCKET() AS NO_BUCKET;
```

<u>NULL_BUCKET</u>	<u>NO_BUCKET</u>
?	1048575

If you ran this query over and over again you would get the same result.

The HASHBUCKET function is used to produce the 16-bit binary Hash Bucket (the DSW) that is used with the Hash Map to determine the AMP that should store and retrieve the data row. It can return a maximum of just over 1,000,000 unique values. The values range from 0 to 1,048,575, not counting the NULL as a potential result. The input to the HASHBUCKET is the 32-bit Row Hash value.

The HASHBUCKET Function in a real-world Example

```
SELECT COUNT(*) AS NBR_ROWS  
      ,HASHBUCKET(HASHROW (Student_ID)) AS Bucket_No  
FROM Student_Table  
GROUP BY 2 ;
```

NBR_ROWS	Bucket_No
1	1007154
1	499675
1	1020598
1	1019019
1	1017675
1	579440
1	466041
1	975629
1	729631
1	542497

Now that the functionality has been demonstrated on the previous page, a more realistic use might be to see the number of rows in each Hash Bucket for a column in a table:

The HASHAMP Function

The basic syntax for using the HASHAMP function follows:

```
HASHAMP( <hash-bucket> )
```

Below is an actual example

```
SELECT HASHAMP(NULL) AS NULL_BUCKET  
      ,HASHAMP() AS NO_BUCKET;
```

NULL_BUCKET	NO_BUCKET
?	1

The HASHAMP function returns the identification number of the primary AMP for any Hash Bucket number. The input to the HASHAMP function is an integer value. When no value is passed to the HASHAMP function, it returns a number that is one less than the number of AMPs in the current system configuration. If any other data type is passed to it, a run-time error occurs.

The HASHAMP Function in a real-world Example

```
SELECT Student_ID  
      ,HASHBUCKET(HASHROW(Student_ID)) AS Bucket_No  
      ,HASHAMP(HASHBUCKET(HASHROW(Student_ID))) AS AMP_No  
FROM Student_Table  
ORDER BY 1;
```

Student_ID	Bucket_No	AMP_No
123250	1019019	0
125634	542497	1
231222	579440	2
234121	1017675	3
260000	975629	4
280023	499675	3
322133	1007154	2
324652	729631	1
333450	1020598	2
423400	466041	4

Now that the functionality has been demonstrated on the previous page, a more realistic use might be to see the AMP Number for each Student to check for distribution.

A Great HASHAMP Function for Large Tables

```
SELECT COUNT(*) "Count"  
      ,HASHAMP(HASHBUCKET(HASHROW(Product_ID))) AS AMP_No  
FROM Sales_Table  
GROUP BY 2  
ORDER BY 2 ;
```

Count	AMP_No
14	0
7	1

The example above is shown using a smaller table on a small system, but the query will brilliantly show the row counts for each AMP. This is a very valuable report.

The HASHBAKAMP Function

The basic syntax for using the HASHBAKAMP function follows:

```
HASHBAKAMP( <hash-bucket> )
```

Below is an actual example

```
SELECT HASHBAKAMP(NULL) AS NULL_BUCKET  
      ,HASHBAKAMP() AS NO_BUCKET;
```

<u>NULL_BUCKET</u>	<u>NO_BUCKET</u>
?	1

The HASHBAKAMP function returns the identification number of the Fallback AMP for any Hash Bucket number. The input to the HASHBAKAMP function is an integer. When no value is passed to the HASHAMP function, it returns a number that is one less than the number of AMPs in the current system configuration. If any other data type is passed to it, a run-time error occurs.

A Real-World HASBAKHAMP Function Example

```
SELECT Student_ID  
      ,HASHBUCKET(HASHROW(Student_ID)) AS Bucket_No  
      ,HASHBAKAMP(HASHBUCKET(HASHROW(Student_ID)))  
                           AS BAK_AMP_No  
FROM Student_Table ORDER BY 3;
```

Student_ID	Bucket_No	BAK_AMP_No
333450	1020598	0
322133	1007154	0
231222	579440	0
125634	542497	1
260000	975629	1
234121	1017675	1
123250	1019019	1
324652	729631	1
280023	499675	1
423400	466041	1

The example above shows the Student_ID, the Bucket_No and the AMP that contains the fallback rows for each Student_ID.

A Great way to see distribution for Primary and Fallback rows

```
SELECT SUM(NbrRows) AS "Rows Per AMP" ,AMP_Nbr
FROM (SEL COUNT(*),
      HASHBAKAMP(HASHBUCKET(HASHROW(Student_ID)))
      FROM Student_Table GROUP BY 2
UNION ALL
SEL COUNT(*),HASHAMP(HASHBUCKET(HASHROW(Student_ID)))
FROM Student_Table GROUP BY 2) DT (NbrRows, AMP_Nbr)
GROUP BY 2
ORDER BY 2 ;
```

Rows Per AMP	AMP_Nbr
10	0
10	1

The following SELECT can help determine that situation by finding all the Primary rows with their AMP and all the FALBACK rows with their AMPs and than adding them together for the total (notice it uses a derived table to consolidate the rows counts):

Chapter 40

BTEQ – Batch

Teradata Query

“In old age we are like a **batch** of letters that someone has sent. We are no longer in the past, we have arrived.“

- Knut Hamsun

Table of Contents Chapter 40 - BTEQ – Batch Teradata Query

- [BTEQ- Batch Teradata Query Tool](#)
- [How to Logon to BTEQ in Interactive Mode](#)
- [Running Queries in BTEQ in Interactive Mode](#)
- [BTEQ Commands Vs BTEQ SQL Statements](#)
- [WITH BY Command for Subtotals](#)
- [WITH Command for a Grand Total](#)
- [WITH and WITH BY Together for Subtotals and Grand Totals](#)
- [How to Logon to BTEQ in a SCRIPT](#)
- [Running Queries in BTEQ through a Batch Script](#)
- [Running a BTEQ Batch Script through the Command Prompt](#)
- [Running a BTEQ Batch Script through the Run Command](#)
- [Using BTEQ Scripts to IMPORT Data](#)
- [Creating a BTEQ IMPORT for a Comma Separated Value File](#)
- [Four Great Examples/Ways to Run a Teradata BTEQ Script](#)
- [BTEQ Export – Four types of Export Variations](#)
- [Creating a BTEQ Export Script in Record Mode](#)
- [Creating a BTEQ Export Script in Record Mode](#)
- [The Appearance of Record Mode Vs Report Mode Data](#)

BTEQ – Batch TEradata Query Tool

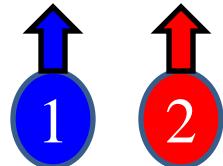
- 1 The first Teradata Query tool delivered back in the late 1980's.
- 2 BTEQ is a report writer, unlike SQL Assistant which is more of a spreadsheet.
- 3 Exports Teradata data off of Teradata a row at a time.
- 4 Imports data onto Teradata a row at a time.
- 5 Queries can be run **interactively** or in a **Batch Script!**

Why is BTEQ available on every Teradata system ever built? Because the Batch TEradata Query (BTEQ) tool was the original way that SQL was submitted to Teradata as a means of getting an answer set in a desired format. Here is what is excellent about BTEQ: It is a cool report writer and can do things SQL Assistant cannot do. It is also great at Export and Import of data at a row level. Although somewhat outdated BTEQ can be a valuable asset (on a rainy day)!

How to Logon to BTEQ in Interactive Mode

Enter your logon or BTEQ Command:

.logon localtd/**dbc**



1 This is your TDPID –
Which is your System ID.

2 This is your Username

Password: *****



3 You will then be prompted
for your password.

Logon successfully completed

BTEQ – Enter your DBC/SQL request or BTEQ command:

When logging onto BTEQ in interactive mode you will type .LOGON followed by your TDP-ID and then a forward slash. The TDP-ID identifies your system. You could have multiple Teradata systems, such as a production system and a test system. The TDP is the name of the system you are logging onto. Then you enter your User-ID. You will then be prompted for your password, which is hidden from view as you type it in.

Running Queries in BTEQ in Interactive Mode

Enter your logon or **BTEQ** Command:

.logon localtddbc

Password: *****

Logon successfully completed

BTEQ – Enter your DBC/SQL request or BTEQ command:

```
SELECT *  
FROM SQL_Class.Employee_Table ;
```



The SEMI-COLON is what
signals BTEQ the query is
ready to be executed.

Remember that BTEQ commands begin with a period (.) and do not require a semi-colon (;) to end the statement. SQL commands do not ever start with a period and they must always be terminated with a semi-colon.

BTEQ Commands Vs BTEQ SQL Statements

BTEQ – Enter your DBC/SQL request or BTEQ command:

 .SET Sidetitles ON

BTEQ Commands will
always start with a
period and
NOT end with a
semi-colon!

SELECT *
FROM SQL_Class.Employee_Table
WHERE Employee_No = 1000234;

 SQL will **NOT** Start with
a period and
always end with a
semi-colon!

Employee_No	1000234
Dept_No	10
Last_Name	Smythe
First_Name	Richard
Salary	64300.00

 A Report that uses
Sidetitles

 .SET Sidetitles OFF

Remember that BTEQ commands begin with a period (.) and do not require a semi-colon (;) to end the statement. SQL commands do not ever start with a period and they must always be terminated with a semi-colon.

WITH BY Command for Subtotals

BTEQ – Enter your DBC/SQL request or BTEQ command:

```
SELECT *
FROM SQL_Class.Employee_Table
WITH SUM(Salary) BY Dept_No ;
```

Employee_No	Dept_No	Last_Name	First_Name	Salary
1333454	200	Smith	John	48000.00
1325457	200	Coffing	Billy	41888.88
→ SUM(Salary)				89888.88
1256349	400	Harrison	Herbert	54500.00
1121334	400	Strickling	Cletus	54500.00
2341218	400	Reilly	William	36000.00
→ SUM(Salary)				145000.00

We have
Subtotals
on
all
Dept_No
breaks

The WITH BY command shows each detail line about a particular row in a particular department number. Then when there is a new department number the report will break and show the SUM (Salary) for that particular department. This shows subtotals!

WITH Command for a Grand Total

BTEQ – Enter your DBC/SQL request or BTEQ command:

```
SELECT *
FROM SQL_Class.Employee_Table
WITH SUM(Salary) ;
```

Employee_No	Dept_No	Last_Name	First_Name	Salary
1333454	200	Smith	John	48000.00
1325457	200	Coffing	Billy	41888.88
1256349	400	Harrison	Herbert	54500.00
1121334	400	Strickling	Cletus	54500.00
2341218	400	Reilly	William	36000.00

SUM(Salary) 234888.88

The example above uses the WITH, but it doesn't use the BY statement. If you just use the WITH statement you can get aggregate grand totals. Both the WITH and the WITH BY statements can be used together to get both subtotals and grand totals.

WITH and WITH BY Together for Subtotals and Grand Totals

BTEQ – Enter your DBC/SQL request or BTEQ command:

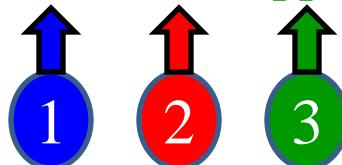
```
SELECT *
FROM SQL_Class.Employee_Table
WITH SUM(Salary) BY Dept_No
WITH SUM(Salary) ;
```

Employee_No	Dept_No	Last_Name	First_Name	Salary
1333454	200	Smith	John	48000.00
1325457	200	Coffing	Billy	41888.88
				SUM(Salary) 89888.88
1256349	400	Harrison	Herbert	54500.00
1121334	400	Strickling	Cletus	54500.00
2341218	400	Reilly	William	36000.00
				SUM(Salary) 145000.00
				SUM(Salary) 234888.88

This example shows both subtotals and grand totals!

How to Logon to BTEQ in a SCRIPT

.logon localtd/**dbc**, apple123



1

This is your TDPID –
Which is your System ID.

2

This is your Username

3

Your password is placed
after the comma

When logging onto BTEQ in a script you will type .LOGON followed by your TDP-ID and then a forward slash. The TDP-ID identifies your system. You could have multiple Teradata systems, such as a production system and a test system. The TDP is the name of the system you are logging onto. Then you enter your User-ID. Then you separate your password from your User-ID by a comma.

Running Queries in BTEQ through a Batch Script

```
.logon localtd/dbc, apple123
```

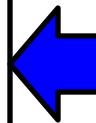
```
SELECT *  
FROM SQL_Class.Employee_Table ;
```

```
.QUIT  
.LOGOFF
```

We created this script in Notepad.

We saved the SCRIPT and called it:

C:\Temp\Script1.txt



When you want to run your SQL or Import or Export jobs in a script you create the script in something like notepad and save it. We saved the above script that we named Script1.txt in our C:\Temp directory on our local PC hard drive.

Running a BTEQ Batch Script through the Command Prompt

```
.logon localtd/dbc, apple123
```

```
SELECT *  
FROM SQL_Class.Employee_Table ;
```

```
.QUIT  
.LOGOFF
```

We created this script in Notepad.

We saved the SCRIPT and called it:

C:\Temp\Script1.txt

We then go to our Command Prompt and type:

C: BTEQ < C:\Temp\Script1.txt

The < sign tells BTEQ to get all commands from the Script and NOT the keyboard.

Once our script is created and saved we can go to the command prompt. There we invoke BTEQ, place a less than sign and give it the script name using the full path.

Running a BTEQ Batch Script through the Run Command

```
.logon localtd/dbc, apple123
```

```
SELECT *  
FROM SQL_Class.Employee_Table ;
```

```
.QUIT  
.LOGOFF
```

We created this script in Notepad.

We saved the SCRIPT and called it:

C:\Temp\Script1.txt

Enter your logon or **BTEQ** Command:

.logon **localtd/dbc**, apple123

.RUN FILE=C:\Temp\Script1.txt

Once our script is created and saved we can go logon to BTEQ. There we put in the .RUN FILE command and give it the script name using the full path.

Using BTEQ Scripts to IMPORT Data

```
.logon localtd/dbc, apple123
```

```
.IMPORT DATA FILE = C:\Temp\CustData.txt
```

```
.REPEAT *
```

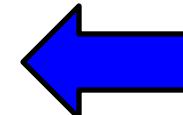
```
USING (IN_CustNo      VARCHAR(11)
       ,IN_Cust_Name   VARCHAR(20)
       ,IN_Cust_Phone  VARCHAR(8))
```

```
INSERT INTO SQL_Class.Customer_Table
VALUES      (:IN_CustNo
             ,:IN_Cust_Name
             ,:IN_Cust_Phone);
```

```
.QUIT
```

```
.LOGOFF
```

1



We created the script
in Notepad.

We saved the SCRIPT
and called it:

C:\Temp\ImpScript.txt

2

In the command prompt type:

C: **BTEQ < C:\Temp\ImpScript.txt**



The < sign tells BTEQ to get
all commands from the Script
and NOT the keyboard.

The example shows a BTEQ IMPORT Script. We are taking data from our flat file called C:\Temp\CustData.txt and importing the records into the Customer_Table.

Creating a BTEQ IMPORT for a Comma Separated Value File

```
.logon localtd/dbc, apple123  
.IMPORT VARTEXT ',' FILE= C:\Temp\var1.txt  
.REPEAT *  
USING (IN_CustNo      VARCHAR(11)  
       ,IN_Cust_Name   VARCHAR(20)  
       ,IN_Cust_Phone  VARCHAR(8))  
  
INSERT INTO SQL_Class.Customer_Table  
VALUES      (:IN_CustNo  
            ,:IN_Cust_Name  
            ,:IN_Cust_Phone) ;  
.QUIT  
.LOGOFF
```

CSV files are a Comma Separated Value flat file. Each column is separated by a comma. We have taken our comma separated value flat file called C:\Temp\var1.txt and we are importing that to our SQL_Class.Customer_Table. Notice how each column is defined using VARCHAR. This is necessary when using CSV files.

Four Great Examples/Ways to Run a Teradata BTEQ Script

At your Command Prompt

1 C: BTEQ < C:\Temp\Script1.txt

The < sign tells BTEQ to get all commands from the Script and NOT the keyboard. The output and error messages come to your PC.

2 C: BTEQ < C:\Temp\Script1.txt > C:\Temp\Output.txt

Invoke
BTEQ

Take Commands
from the script

> means send all output
messages to the file.

Logon to a BTEQ Session:

3 Enter your logon or BTEQ Command:

.logon localtd/**dbc**, apple123

.RUN FILE=C:\Temp\Script1.txt

4 The fastest and easiest way is via the Nexus Query Chameleon. It will Create the Script and Executed it immediately or you can Schedule it to run later on the Nexus.

BTEQ Export – Four types of Export Variations

1	EXPORT DATA	Exports data in record mode, but completely unreadable to viewers.
2	EXPORT Report	Exports in report mode, which includes the headers and is readable.
3	EXPORT INDICDATA	Exports in INDICDATA mode warning mainframes of NULLs.
4	EXPORT DIF	Exports and converts data into Data Interchange Format for PC's.

BTEQ allows for multiple techniques to export data. We usually think of an export as moving data off of Teradata to a normal flat file. That is example number one and that is called RECORD Mode.

BTEQ can actually take your SQL output report and include the Headers and export all together. It looks like an electronic report. That is EXPORT REPORT mode. This is also called Field Mode.

When there are NULL's in your data and you export them to a mainframe the actual mainframe application could run into problems so INDICDATA warns of NULL values.

Use DIF to be able to be used by PC applications in Data Interchange Format.

Creating a BTEQ Export Script in Record Mode

```
.logon localtd/dbc, apple123  
.EXPORT DATA FILE= C:\Temp\Employee1.txt
```



Tells BTEQ to Export the next query run in Record Mode to the output file listed.

```
SELECT *  
FROM SQL_Class.Employee_Table ;
```

```
.EXPORT RESET
```



Tells BTEQ to now close the Export

```
.QUIT
```

```
.LOGOFF
```

This is a script you would create in Notepad. The first statement is the LOGON statement. The last statement is the LOGOFF statement. You basically tell BTEQ you are going to export a file. Then you run a SELECT Query and BTEQ Exports it!

Creating a BTEQ Export Script in Report Mode

```
.logon localtd/dbc, apple123
```

```
.EXPORT REPORT FILE= C:\Temp\Emp_Report.txt
```



Tells BTEQ to Export the next query run in Report Mode to the output file listed.

```
SELECT *  
FROM SQL_Class.Employee_Table ;
```

```
.EXPORT RESET
```



Tells BTEQ to now close the Export

```
.QUIT
```

```
.LOGOFF
```

This is a script you would create in Notepad. The first statement is the LOGON statement. The last statement is the LOGOFF statement. You basically tell BTEQ you are going to export a file in REPORT Mode so users can read it including the headers.

The Appearance of Record Mode Vs Report Mode Data

Record Mode

```
|| %$@#Coffing      * }{MN $#/  smith @$!!_FQ]{\ { | Jones  
@ @      $%Davis:+E+      R<DSIQ @ @Reilly #$_@*  
R@Johnson    *$%_F(E#>Young#)      $*adomr0$#@
```



Record Mode looks like garbage when viewed, but it is correct and will load perfectly

Report Mode

Report Mode is an electronic report designed to be viewed.

Employee_No	Dept_No	Last_Name	First_Name	Salary
1333454	200	Smith	John	48000.00
1325457	200	Coffing	Billy	41888.88
1256349	400	Harrison	Herbert	54500.00
1121334	400	Strickling	Cletus	54500.00
2341218	400	Reilly	William	36000.00

You will often be alarmed if you try and view the output of Record Mode. It looks like garbage, but it is exactly what you want with a flat file that will be imported later. The Report Mode is pretty and designed to be viewed electronically, including headers.

Chapter 41

Your Nexus Query

Chameleon

“Anything you can do needs to be done, so pick up the tool
of your choice and get started.”

- Ben Linder

Table of Contents Chapter 41 - Your Nexus Query Chameleon

- [The Old Nexus Logo](#)
- [The New Nexus Logo](#)
- [Watch the Video on the new Nexus Super Join Builder](#)
- [How to Customize your System Tree View](#)
- [Introducing the new Nexus Super Join Builder](#)
- [Define your Joins and tell Nexus to “Add and Remember Me”](#)
- [Nexus knows what Tables Join together](#)
- [Nexus Presents Tables and their columns in Color](#)
- [Nexus Builds your SQL Automagically](#)
- [Nexus can Cube a Table and Join to Everything Possible](#)
- [The Cube SQL created Automagically](#)
- [Manipulate the Columns with the Columns Tab](#)
- [Single Click and ORDER BY using the Sort Tab](#)
- [Using the Joins Tab of Nexus](#)
- [The SQL Tab reflects the changes we make in all other Tabs](#)
- [WHERE Tab shows Tables Indexes](#)
- [The Answer Set Tab shows the Results](#)
- [The Metadata Tab shows Metadata](#)
- [Nexus Makes a View look like a Table](#)
- [Nexus Joins Views to other Views in seconds](#)
- [Nexus can Cube a View and Join to all other related Views](#)
- [Nexus Cubes Views in Seconds](#)
- [The Cube SQL created on Views is done Automagically](#)
- [Views with the Underlying Indexes of the Base Tables](#)
- [WHERE Tab shows Views Underlying Base Table Indexes](#)
- [After an Answer Set Returns you can do many things](#)
- [After an Answer Set Returns Perform OLAP Calculations](#)

Continued on next page

Table of Contents Chapter 41 - Your Nexus Query Chameleon Continued

- [After an Answer Set Returns you can Graph and Chart](#)
- [Bisualize](#)
- [Bisualize Builds Dynamic Charts](#)
- [Saving an Answer Set in another Format](#)
- [Sandbox – How to Create a Sandbox \(1 of 5\)](#)
- [How to Compress the Tables in an Entire Database](#)
- [How to Compress A Single Table](#)
- [Compression Reports](#)
- [Let Nexus Build your Load Scripts with SmartScript](#)
- [SmartScript Building a FastLoad](#)
- [The Teradata DBA Launchpad](#)
- [Convert Teradata DDL to Another Database Vendor](#)
- [Replicate Data from One Teradata System to Another](#)
- [Compare and Synchronize with SmartSync](#)
- [Compare Data on Two tables with SmartSync](#)
- [Compare Data and See the Results with SmartSync](#)
- [Synchronize the Data with SmartSync](#)
- [Compare DDL with SmartSync](#)
- [Play Games and Learn Teradata SQL](#)
- [Play Games and Learn Teradata Architecture](#)
- [Watch the Video on India](#)

The Old Nexus Logo



This was the original logo of the Nexus, but after partnering with Microsoft to build the next generation Query Tool they created the new logo for us (seen on the next page).

The New Nexus Logo



Download the Nexus for a free trial at www.CoffingDW.com

Microsoft created the new logo for us to represent the Nexus as an Enterprise tool.

Watch the Video on the new Nexus Super Join Builder



Tera-Tom Trivia

Tom Coffing has been one of the first teachers of Teradata data warehousing in remote places like Johannesburg Africa. In Africa, after Apartheid Tom trained both black and white students together which was a first time experience for many of the students.

Click on the link below or place it in your browser and watch the video on the Nexus Super Join builder.

<http://www.coffingdw.com/TbasicsV12/Nexus.wmv>



System: **Teradata** ▼ Database: **SQL_Class** ▼ Execute

Systems ▼ X Query1 * X ▼

- + DB2
- + Oracle
- + Netezza
- + Greenplum
- + ParAccel

Kognitio

PDW

Sandbox

- Teradata

+ Sybase

+ Vertica

+ SQL Server

Systems Tree

```
SELECT Employee_No
      ,Dept_No
      ,Last_Name
      ,First_Name
      ,Salary
   FROM SQL_CLASS.Employee_Table ;
```

Query Window

Messages Results1

Drag a column here to group by that column

	Employee_No	Dept_No	Last_Name	First_Name	Salary
1	1324657	200	Coffing	Billy	41888.88
2	2000000	?	Jones	Squiggy	32800.50
3	1333454	200	Smith	John	48000.00
4	1232578	100	Chambers	Mandee	48850.00
5	1121334	400	Strickling	Cletus	54500.00
6	2312225	300	Larkins	Lorraine	40200.00
7	2312225	400	Reilly	William	36000.00
8	2341218	10	Smythe	Richard	64300.00

Results Window



System: **Teradata** ▼ Database: **SQL_Class** ▼  Execute           

Systems X Query1 * X

+ I DB2

+ O Oracle

+ N Netezza

+ G Greenplum

+ A ParAccel

+ Kognitio

+ M PDW

+ S Sandbox

- T Teradata

+ S Sybase

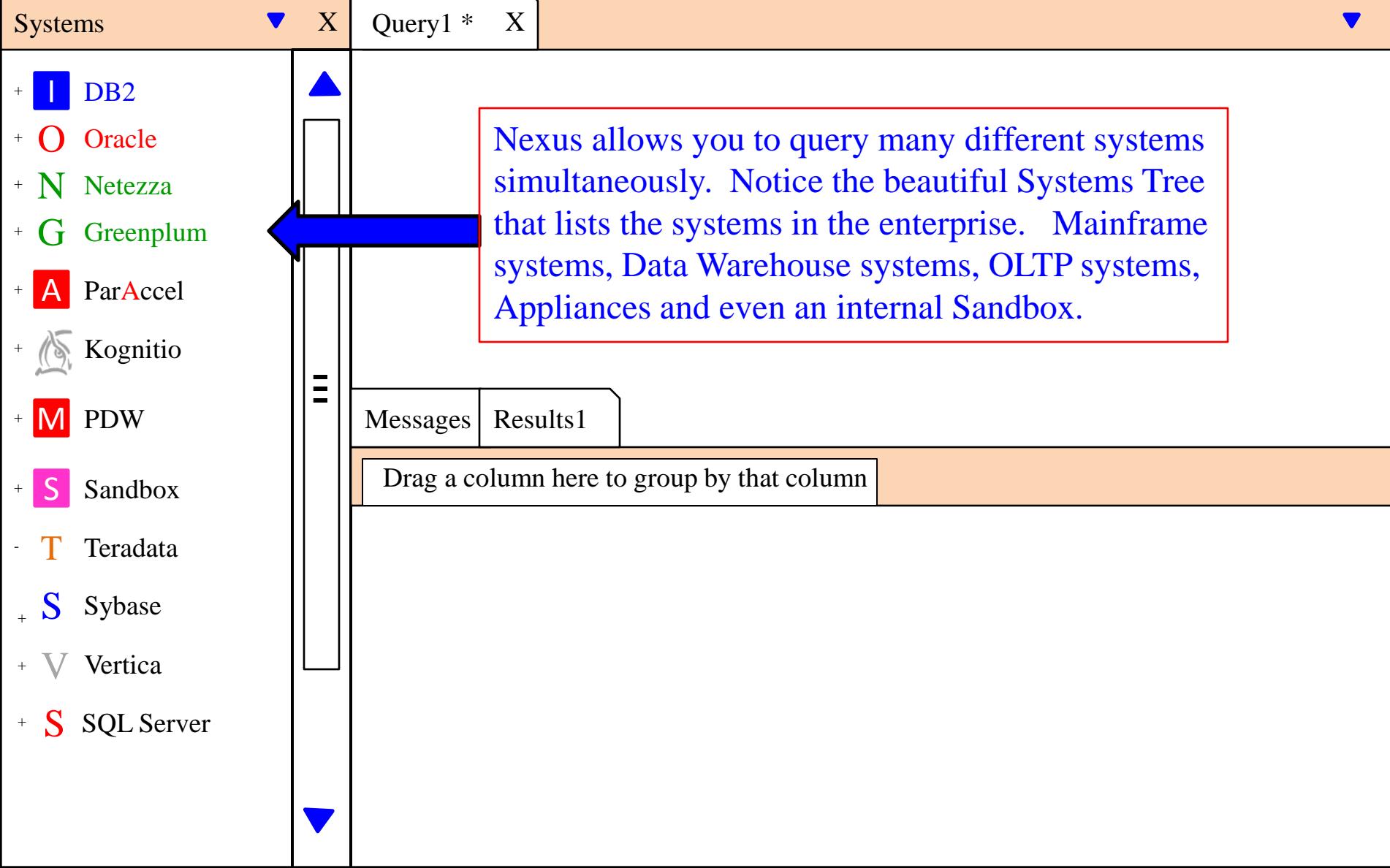
+ V Vertica

+ S SQL Server

Nexus allows you to query many different systems simultaneously. Notice the beautiful Systems Tree that lists the systems in the enterprise. Mainframe systems, Data Warehouse systems, OLTP systems, Appliances and even an internal Sandbox.

Messages Results1

Drag a column here to group by that column





System: **Teradata** ▼ Database: **SQL_Class** ▼  Execute           

Systems	Query1 *
<ul style="list-style-type: none">-  Teradata-  SQL_Class<ul style="list-style-type: none">-  Tables<ul style="list-style-type: none">+  Addresses+  Claims+  Course_Table+  Customer_Table+  Department_Table+  Employee_Table+  Order_Table+  Providers+  Views+  Macros+  Procedures+  Triggers+  Join Indexes+  Functions	X

Each Systems Tree is designed to look exactly like the database vendor expects

This is an example of a Teradata System Tree, which has Tables, Views, Macros, Procedures, Triggers, Join Indexes and Functions

System: **Teradata** ▼ Database: **SQL_Class** ▼  Execute          

Systems	Query1 *
<ul style="list-style-type: none">-  Teradata-  SQL_Class<ul style="list-style-type: none">-  Tables<ul style="list-style-type: none">+  Addresses+  Claims+  Course_Table+  Customer_Table+  Department_Table+  Employee_Table-  Columns<ul style="list-style-type: none"> Employee_No (Integer) Dept_No (Smallint) Last_Name (Char(20)) First_Name (Varchar(20)) Salary (Decimal(8,2))+  Order_Table+  Providers	<p>The columns for each table are listed with their data types. Blue indicates the Primary Index (Employee_No). Drag and Drop columns inside Nexus if you don't want to type.</p>

How to Customize your System Tree View



Systems X Query1 * X

Right Click on the system Icon

View Tree

Hierarchy

No Hierarchy

My Databases

All Databases and Users

All Databases and Users with Perm Space

All Databases

All Users

Edit My Databases

Add

Remove

Please enter the database or user

SQL_Class

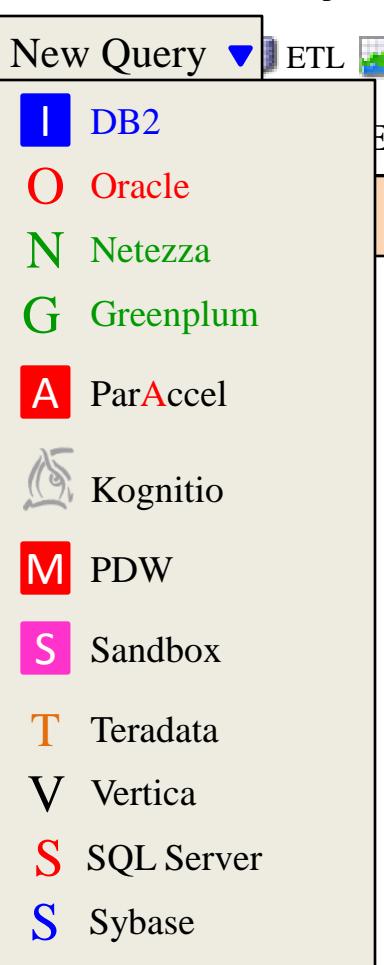
The interface displays a system tree on the left under 'Teradata' and 'SQL_Class'. A context menu is open over the 'Teradata' icon, containing options like 'Connect tree to server', 'Disconnect tree from server', 'View Tree', 'Modify data source connection', 'Remove data source connection', 'Set as default data source connection', and 'Refresh data source connection'. Another context menu is open over the 'SQL_Class' icon, listing 'Hierarchy', 'No Hierarchy', 'My Databases' (which is checked), 'All Databases and Users', 'All Databases and Users with Perm Space', 'All Databases', and 'All Users'. A third context menu is open over the 'Edit My Databases' button, with 'Add' and 'Remove' options. A text input field is shown with the placeholder 'Please enter the database or user', and the value 'SQL_Class' is entered.



System: Teradata ▾ Database: SC

Systems ▾ X

- + I DB2
- + O Oracle
- + N Netezza
- + G Greenplum
- + A ParAccel
- +  Kognitio
- + M PDW
- + S Sandbox
- T Teradata
- + S Sybase
- + V Vertica
- + S SQL Server



Dashboard Bisualize History Sandbox Joins DBA

Execute



The New Query button will allow you to query different systems simultaneously.

The next slide will show what happens when we query them all simultaneously!

Messages Results1

Drag a column here to group by that column



System: **Teradata** ▼ Database: **SQL_Class** ▼ Execute

Systems	X	Query 1 * X	Query 2 * X	Query 3 * X	Query 4 * X	Query 5 * X	Query 6 * X	▼																																													
+ I DB2	▲		↑																																																		
+ O Oracle			↑																																																		
+ N Netezza				↑																																																	
+ G Greenplum					↑																																																
+ A ParAccel						↑																																															
+ Kognitio	≡						↑																																														
+ M PDW	≡																																																				
+ S Sandbox																																																					
- T Teradata																																																					
+ S Sybase																																																					
+ V Vertica																																																					
+ S SQL Server	▼																																																				
<pre> SELECT Employee_No ,Dept_No ,Last_Name ,First_Name ,Salary FROM SQL_CLASS.Employee_Table ; </pre>																																																					
<p>Messages Results1</p> <p>Drag a column here to group by that column</p> <table border="1"> <thead> <tr> <th>Employee No</th> <th>Dept No</th> <th>Last Name</th> <th>First Name</th> <th>Salary</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>1324657</td> <td>200</td> <td>Coffing</td> <td>Billy 41888.88</td> </tr> <tr> <td>2</td> <td>2000000</td> <td>?</td> <td>Jones</td> <td>Squiggy 32800.50</td> </tr> <tr> <td>3</td> <td>1333454</td> <td>200</td> <td>Smith</td> <td>John 48000.00</td> </tr> <tr> <td>4</td> <td>1232578</td> <td>100</td> <td>Chambers</td> <td>Mandee 48850.00</td> </tr> <tr> <td>5</td> <td>1121334</td> <td>400</td> <td>Strickling</td> <td>Cletus 54500.00</td> </tr> <tr> <td>6</td> <td>2312225</td> <td>300</td> <td>Larkins</td> <td>Lorraine 40200.00</td> </tr> <tr> <td>7</td> <td>2312225</td> <td>400</td> <td>Reilly</td> <td>William 36000.00</td> </tr> <tr> <td>8</td> <td>2341218</td> <td>10</td> <td>Smythe</td> <td>Richard 64300.00</td> </tr> </tbody> </table>									Employee No	Dept No	Last Name	First Name	Salary	1	1324657	200	Coffing	Billy 41888.88	2	2000000	?	Jones	Squiggy 32800.50	3	1333454	200	Smith	John 48000.00	4	1232578	100	Chambers	Mandee 48850.00	5	1121334	400	Strickling	Cletus 54500.00	6	2312225	300	Larkins	Lorraine 40200.00	7	2312225	400	Reilly	William 36000.00	8	2341218	10	Smythe	Richard 64300.00
Employee No	Dept No	Last Name	First Name	Salary																																																	
1	1324657	200	Coffing	Billy 41888.88																																																	
2	2000000	?	Jones	Squiggy 32800.50																																																	
3	1333454	200	Smith	John 48000.00																																																	
4	1232578	100	Chambers	Mandee 48850.00																																																	
5	1121334	400	Strickling	Cletus 54500.00																																																	
6	2312225	300	Larkins	Lorraine 40200.00																																																	
7	2312225	400	Reilly	William 36000.00																																																	
8	2341218	10	Smythe	Richard 64300.00																																																	

Each Query
(and answer set)
is placed in a colored
tab that identifies the
system

Introducing the new Nexus Super Join Builder

The screenshot shows the Nexus Super Join Builder interface. On the left, a sidebar lists supported systems: DB2, Oracle, Netezza, Greenplum, ParAccel, Kognitio, PDW, Sandbox, Teradata, and SQL_Class (with sub-options: Tables, Addresses, Claims, Course_Table, and Department). The main area is titled "Query1 * X" and contains a navigation bar with tabs: Objects (red), Columns (orange), Sort (yellow), Joins (green), Where (purple), SQL (blue, selected), Answer Set (pink), Metadata (magenta), and Analytics (cyan). Below the navigation bar, several callout boxes provide instructions for each tab:

- Objects: Shows the Tables/Views In your query
- Columns: Use this to manipulate the columns
- Sort: Use this to ORDER BY
- Joins: Change the Joins here
- Where: Add/Change the WHERE clause
- SQL: Turns Red when your Answer Set arrives
- Answer Set: See the SQL Nexus has generated
- Metadata: Shows the Metadata for a Table/View
- Analytics: Build OLAP commands here

A large red-bordered box in the bottom right corner contains the text:

All 9 Tabs work together to build your query or join.
You just Drag and Drop (Tables or Views) and then Point and Click (through the tabs) and your query is built!

The Super Join Builder has 9 Tabs that are all there for your single query. Manipulate through the tabs and make changes to your query. The Super Join Builder does amazing things!

Define your Joins and tell Nexus to “Add and Remember Me”

Systems ▼ X Query1 * X ▼

Objects Columns Sort Joins Where SQL Answer Set Metadata Analytics

Custom Joins Join Type INNER

Addresses

Add Join..

Select *
Street varchar(30)
City varchar(20)
State char(2)
Zip integer
AreaCode smallint
Phone integer
Subscriber_No integer

Claims

Add Join..

Select *
Claim_Id integer
Claim_Date decimal(9,2)
Subscriber_No integer
Member_No smallint
Claim_Amt decimal(12,2)
Provider_No smallint
Claim_Service smallint

Add Join

Add and Remember Join

```
graph LR; Addresses[Addresses] --> Claims[Claims];
```

Drag two tables inside Nexus and define the Join Columns. Click on “Add and Remember Join” and its in the Join Menu permanently. Check out the cool Join Menu on the next page.

Nexus knows what Tables Join together

This tab displays the objects in your Query

Objects Columns Sort Joins Where SQL Answer Set Metadata Analytics

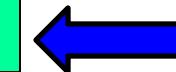
Claims

Select *
Claim_Id integer
Claim_Date decimal(10,2)
Subscriber_No integer
Member_No smallint
Claim_Amt decimal(12,2)
Provider_No smallint
Claim_Service smallint

Add Join ...

Subscribers ► Addresses
Providers ►
Services ►

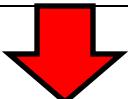
Left click on the Add Join and Nexus shows you what other tables can join in a beautiful menu.



Watch what happens on the next slide when we click on the menu to Join our Claims table to both the Subscribers table and the Addresses Table.

How would you like to NEVER write another Join again? The Nexus makes that possible with the Nexus Super Join Builder. Just Right Click on a table and the menu tells you what other table(s) are eligible in the join. Then when you click on the next table Nexus writes the Join SQL automatically.

Nexus Presents Tables and their columns in Color



This tab displays the objects in your Query

Objects

Columns

Sort

Joins

Where

SQL

Answer Set

Metadata

Analytics

Claims

Claims	
<input checked="" type="checkbox"/> Select *	Add Join...
<input checked="" type="checkbox"/> Claim_Id integer	
<input checked="" type="checkbox"/> Claim_Date decimal(9,2)	
<input checked="" type="checkbox"/> Subscriber_No integer	
<input checked="" type="checkbox"/> Member_No smallint	
<input checked="" type="checkbox"/> Claim_Amt decimal(12,2)	
<input checked="" type="checkbox"/> Provider_No smallint	
<input checked="" type="checkbox"/> Claim_Service smallint	

Subscribers

Subscribers	
<input type="checkbox"/> Select *	Add Join...
<input checked="" type="checkbox"/> Last_Name varchar(20)	
<input checked="" type="checkbox"/> First_Name varchar(20)	
<input type="checkbox"/> Gender char(1)	
<input type="checkbox"/> SSN integer	
<input type="checkbox"/> Member_No smallint	
<input type="checkbox"/> Subscriber_No integer	

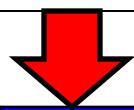
Addresses

Addresses	
<input type="checkbox"/> Select *	Add Join...
<input checked="" type="checkbox"/> Street varchar(30)	
<input checked="" type="checkbox"/> City varchar(20)	
<input checked="" type="checkbox"/> State char(2)	
<input checked="" type="checkbox"/> Zip integer	
<input type="checkbox"/> AreaCode smallint	
<input type="checkbox"/> Phone integer	
<input type="checkbox"/> Subscriber_No integer	

The tables in our join are color coded. You will see why soon!

All Three tables we wish to join appear now with their columns and data types. Each table is a different color so when users utilize the other Tabs they see the columns in color to indicate which table there from. The Join SQL is already built, but we can use the other tabs to customize the SQL. The next picture will show the SQL already built!

Nexus Builds your SQL Automagically



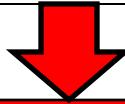
This tab displays the SQL created by Nexus

Objects	Columns	Sort	Joins	Where	SQL	Answer Set	Metadata	Analytics
---------	---------	------	-------	-------	-----	------------	----------	-----------

```
SELECT cla.Claim_Id, cla.Claim_Date, cla.Subscriber_No,
       cla.Claim_Amt, cla.Provider_No, cla.Claim_Service,
       sub.Last_Name, sub.First_Name,
       "add".Street, "add".City, "add"."State", "add".Zip
FROM   SQL_Class.Claims cla
       INNER JOIN
              SQL_Class.Subscribers sub
ON     cla.Subscriber_No = sub.Subscriber_No
       AND cla.Member_No    = sub.Member_No
       INNER JOIN
              SQL_Class.Addresses "add"
ON     sub.Subscriber_No = "add".Subscriber_No ;
```

By clicking on the **SQL Tab** you can see the SQL generated by the Nexus. All we did was click on the Objects Tab to select the tables and then we clicked on the columns we wanted on our report. In seconds the Nexus built this beautiful SQL.

Nexus can Cube a Table and Join to Everything Possible



This tab displays the objects in your Query

Objects

Columns

Sort

Joins

Where

SQL

Answer Set

Metadata

Analytics

Left click on the Left Side
of a Table or View and press
Create Cube with Columns

Claims

Add Join ▾

Create Cube

Create Cube with Columns

- Select *
- Claim_Id integer
- Claim_Date decimal(9,2)
- Subscriber_No integer
- Member_No smallint
- Claim_Amt decimal(12,2)
- Provider_No smallint
- Claim_Service smallint

Check out the
next slide to see
what happens next.

When you click on the left side of a table in the Objects Tab and select CUBE the Nexus will create the SQL to join every table in the entire relationship together. Watch what happens on the next slide.

Nexus can Cube a Table and Join to Everything Possible

Objects

Columns

Sort

Joins

Where

SQL

Answer Set

Metadata

Analytics



Claims

+ Add Join ▾

- Select *
- Claim_Id integer
- Claim_Date decimal(9,2)
- Subscriber_No integer
- Member_No smallint
- Claim_Amt decimal(12,2)
- Provider_No smallint
- Claim_Service smallint

Providers

+ Add Join ▾

- Select *
- Provider_Code smallint
- Provider_Name varchar(30)
- P_Address varchar(30)
- P_City varchar(20)
- P_State char(2)
- P_Zip integer
- P_Error_Rate decimal(4,4)

Subscribers

+ Add Join ▾

- Select *
- Last_Name varchar(20)
- First_Name varchar(20)
- Gender char(1)
- SSN integer
- Member_No smallint
- Subscriber_No integer

Addresses

+ Add Join ▾

- Select *
- Street varchar(30)
- City varchar(20)
- State char(2)
- Zip integer
- AreaCode smallint
- Phone integer
- Subscriber_No integer

Services

+ Add Join ▾

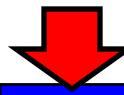
- Select *
- Service_Code smallint
- Service_Desc varchar(30)
- Service_Pay decimal(7,2)

All five tables
have now been
joined by Nexus

Check out the
SQL tab on the
next slide

After hitting Cube with all Columns all tables in the relationship are joined.

The Cube SQL created Automagically

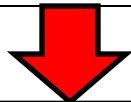


Objects	Columns	Sort	Joins	Where	SQL	Answer Set	Metadata	Analytics
---------	---------	------	-------	-------	-----	------------	----------	-----------

```
SELECT cla.Claim_Id, cla.Claim_Date, cla.Subscriber_No, cla.Member_No,  
cla.Claim_Amt, cla.Provider_No, cla.Claim_Service,  
sub.Last_Name, sub.First_Name, sub.Gender, sub.SSN,  
sub.Member_No, sub.Subscriber_No,  
"add".Street, "add".City, "add"."State", "add".Zip, "add".AreaCode, "add".Phone,  
"add".Subscriber_No,  
pro.Provider_Code, pro.Provider_Name, pro.P_Address, pro.P_City,  
pro.P_State, pro.P_Zip, pro.P_Error_Rate,  
ser.Service_Code, ser.Service_Desc, ser.Service_Pay  
FROM SQL_CLASS.CLAIMS cla  
INNER JOIN SQL_CLASS.SUBSCRIBERS sub  
    ON cla.Subscriber_No = sub.Subscriber_No  
        AND cla.Member_No = sub.Member_No  
INNER JOIN SQL_CLASS.ADDRESSES "add"  
    ON sub.Subscriber_No = "add".Subscriber_No  
INNER JOIN SQL_CLASS.PROVIDERS pro  
    ON cla.Provider_No = pro.Provider_Code  
INNER JOIN SQL_CLASS.SERVICES ser  
    ON cla.Claim_Service = ser.Service_Code;
```

This SQL created by hitting
CREATE Cube with Columns

Manipulate the Columns with the Columns Tab



This tab displays the Columns in your Query and those columns NOT selected

Objects

Columns

Sort

Joins

Where

SQL

Answer Set

Metadata

Analytics



Drag columns here to remove them from the report

Columns on Report

Claim_Id

Claim_Date

Subscriber_No

Member_No

Claim_Amt

Provider_No

Claim_Service

Last_Name

First_Name

Street

City

State

Zip

Columns NOT on Report

Gender

SSN

Member_No

Subscriber_No

AreaCode

Phone

Subscriber_No

Why are there different colors?
So you can tell which table each
column comes from! Ingenious!

Rearrange the columns in the SELECT list by clicking and dragging. Control Click highlights multiple columns and Shift Click highlights an entire range of columns.

Single Click and ORDER BY using the Sort Tab



Column Name Sort Type

Claim_Id

ASC

Street

DESC

Double Click on a column
and it will ORDER BY that
column(s) in your SQL.

Columns on Report

Claim_Id

Claim_Date

Subscriber_No

Member_No

Claim_Amt

Provider_No

Claim_Service

Last_Name

First_Name

Street

City

State

Zip

Columns NOT on Report

Gender

SSN

Member_No

Subscriber_No

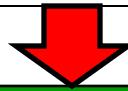
AreaCode

Phone

Subscriber_No

Double-Click or drag any column to sort by that column(s) in ASC or DESC mode.

Using the Joins Tab of Nexus



This tab displays how tables are being Joined

Objects Columns Sort Joins Where SQL Answer Set Metadata Analytics

Join Type Join Objects

INNER ▼ SQL_Class.CClaims cla
JOINS to SQL_Class.Subscribers sub
ON cla.Subscriber_No = sub.Subscriber_No
AND cla.Member_No = sub.Member_No

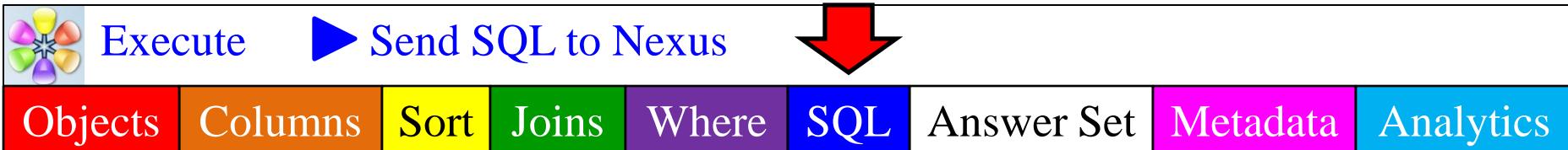
INNER ▼ SQL_Class.Subscribers sub
JOINS to SQL_Class.Addresses add
ON cla.Subscriber_No = sub.Subscriber_No
AND sub.Subscriber_No = add.Subscriber_No

INNER
LEFT
RIGHT
FULL

We just changed this
portion of the join to a
RIGHT OUTER JOIN.

The Joins Tab shows all Join Conditions and allows you to change them to Outer Joins.

The SQL Tab reflects the changes we make in all other Tabs



```
SELECT cla.Claim_Id, cla.Claim_Date, cla.Subscriber_No,  
    cla.Claim_Amt, cla.Provider_No, cla.Claim_Service,  
    sub.Last_Name, sub.First_Name,  
    "add".Street, "add".City, "add"."State", "add".Zip  
FROM SQL_Class.Claims cla  
    INNER JOIN  
        SQL_Class.Subscribers sub  
ON cla.Subscriber_No = sub.Subscriber_No  
AND cla.Member_No = sub.Member_No  
    RIGHT OUTER JOIN  
        SQL_Class.Addresses "add"  
ON sub.Subscriber_No = "add".Subscriber_No  
ORDER BY cla.Claim_Id ASC, "add".Street DESC;
```

This is the SQL generated after we changed our Join to a Right Outer Join in the Joins Tab and in the Sort Tab dragged columns Claim_Id and Street for our ORDER BY.



System: Teradata ▼ Database: SQL_Class ▼  Execute           

Systems X Query1 * X

+ I DB2

+ O Oracle

+ N Netezza

+ G Greenplum

+ A ParAccel

+ Kognitio

+ M PDW

+ S Sandbox

- T Teradata

- SQL_Class

 Tables

 + Addresses

 + Claims

 + Course_Table

 + Department

Objects Columns Sort Joins Where SQL Answer Set Metadata Analytics



Watch what happens when we go to the Where Tab.

Employee_Table

Select *

Employee_No integer

First_Name Varchar(12)

Last_Name Char(20)

Dept_No Smallint

Salary Decimal(8,2)

WHERE Tab shows Tables Indexes



Objects

Columns

Sort

Joins

Where

SQL

Answer Set

Metadata

Analytics

Columns Used in Report

Employee_No

First_Name

Last_Name

Dept_No

Salary

Indexes

Unique Primary Index

Employee_No

Unique Secondary Index

Last_Name

First_Name

Non-Unique Secondary Index

Last_Name

Dept_No

Salary

AND

OR

IN

BETWEEN

LIKE

NOT IN

= < >

>

Where

WHERE Employee_No = < value >

Click on any column or index and it goes in the WHERE clause. Notice here we clicked on Employee_No.



System: **Teradata** ▼ Database: **SQL_Class** ▼ Execute

Objects **Columns** **Sort** **Joins** **Where** **SQL** **Answer Set** **Metadata** **Analytics**

OLAP

RANK

GROUPING SETS



OLAP Column	Sorting	Partitioning	Moving Window	OLAP Function	With Partitioning
Daily_Sales	Product_ID <input type="checkbox"/>	ASC <input type="checkbox"/>	Product_ID <input type="checkbox"/>	3	<input checked="" type="checkbox"/> SELECT *
	Sale_Date <input type="checkbox"/>	ASC <input type="checkbox"/>		<input checked="" type="checkbox"/> CSUM	<input checked="" type="checkbox"/> CSUM

Columns Used in Report

Product_ID Sale_Date Daily_Sales

Drag the Columns to their above designations and then Check (right) on which OLAP functions you desire.

<input checked="" type="checkbox"/> SELECT *	<input checked="" type="checkbox"/> SELECT *
<input checked="" type="checkbox"/> CSUM	<input checked="" type="checkbox"/> CSUM
<input checked="" type="checkbox"/> MSUM	<input checked="" type="checkbox"/> MSUM
<input checked="" type="checkbox"/> MAVG	<input checked="" type="checkbox"/> MAVG
<input checked="" type="checkbox"/> MDIFF	<input checked="" type="checkbox"/> MDIFF
<input checked="" type="checkbox"/> COUNT	<input checked="" type="checkbox"/> COUNT



System: **Teradata** ▼ Database: **SQL_Class** ▼ Execute

Objects **Columns** **Sort** **Joins** **Where** **SQL** **Answer Set** **Metadata** **Analytics**

OLAP

RANK

GROUPING SETS



OLAP
Column

Sorting

Partitioning

Daily_Sales

Product_ID

DESC

Product_ID

OLAP
Function

With
Partitioning



RANK



RANK

Columns Used in Report

Product_ID

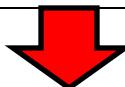
Sale_Date

Daily_Sales

Turn the Page and check
out the Answer Set

Drag the Columns to their above designations and then Check
(right) on which OLAP functions you desire.

The Answer Set Tab shows the Results

Execute  Send SQL to Nexus 

Objects Columns Sort Joins Where SQL Answer Set Metadata Analytics

Results1

Drag a column here to group by that column

			RANK
	Product_ID	Sale_Date	Daily_Sales
1	1000	10/03/2000	64300.00
2	3000	09/28/2000	61301.77
3	2000	10/01/2000	54850.29
4	1000	10/04/2000	54553.10
5	1000	09/29/2000	54500.22
6	2000	09/30/2000	49850.03
7	1000	09/28/2000	48850.40
8	2000	09/29/2000	48000.00
9	3000	09/30/2000	43868.86
10	2000	10/03/2000	43200.18

This is the Answer Set from our previous slide that performed a RANK

To get here we put the Sales_Table inside the Super Join Builder. Then we selected the columns we wanted and then went to the Analytics Tab. There we built our RANK report and pressed Execute. Then (above) we received the Answer Set.

System: Teradata ▼ Database: SQL_Class ▼  Execute          

Objects Columns Sort Joins Where SQL Answer Set Metadata Analytics

OLAP

RANK

GROUPING SETS



Product

Date Column

Sum

 Product_ID Sale_Date Daily_Sales

Date Extract

 Year Month

Columns

 Product_ID Sale_Date Daily_Sales

Drag the Columns to their above designations and then Check (right) on which Grouping Sets desire.

Group By Functions



SELECT *



Grouping Sets



Rollup



Cubes



Execute

- SQL
- Grouping Sets
- Rollup
- Cube

Execute All

Just hit the EXECUTE button and your query(s) run

The next page will show you the Answer Sets

Objects Columns Sort Joins Where SQL Answer Set Metadata Analytics

OLAP RANK

GROUPING SETS



Product

Product_ID

Date Column

Sale_Date

Sum

Daily_Sales

Date Extract

Year

Month

Columns

Product_ID

Sale_Date

Daily_Sales

Drag the Columns to their above designations and then Check (right) on which Grouping Sets desire.

Group By Functions



SELECT *



Grouping Sets



Rollup



Cubes

The Answer Set Tab shows the Results

The screenshot shows a user interface for a data analysis tool. At the top, there is a navigation bar with several tabs: 'Execute' (with a gear icon), 'Send SQL to Nexus' (with a blue arrow icon), 'Objects' (red), 'Columns' (orange), 'Sort' (yellow), 'Joins' (green), 'Where' (purple), 'SQL' (blue), 'Answer Set' (pink, currently selected), 'Metadata' (light blue), and 'Analytics' (cyan). Below the tabs, there are three buttons labeled 'Results1', 'Results 2', and 'Results 3'. A large orange banner at the bottom of the interface says 'Drag a column here to group by that column'. To the right of the table, a red-bordered box contains the text: 'This is Answer Set # 1 from our previous slide that performed Grouping Sets'. The main content area displays a table with the following data:

Product_ID	Mth	Year	Sum_Daily_Sales
1	3000	?	224587.82
2	2000	?	306611.81
3	1000	?	331204.72
4	?	10	443634.99
5	?	9	418769.36
6	?	? 2000	862404.35

To get here we put the Sales_Table inside the Super Join Builder. Then we selected the columns we wanted and then went to the Analytics Tab. There we built our RANK report and pressed Execute. Then (above) we received the Answer Set.

The Answer Set Tab shows the Results

Execute ► Send SQL to Nexus

Objects Columns Sort Joins Where SQL Answer Set Metadata Analytics

Results1 Results 2 Results 3

Drag a column here to group by that column

	Product_ID	Mth	Year	Sum_Daily_Sales
1	3000	10	2000	84908.06
2	3000	10	?	84908.06
3	3000	9	2000	139679.76
4	3000	9	?	139679.76
5	3000	?	?	224587.82
6	2000	10	2000	166872.90
7	2000	10	?	166872.90
8	2000	9	2000	139738.91
9	2000	9	?	139738.91
10	2000	?	?	306611.81
11	1000	10	2000	191854.03
12	1000	10	?	191854.03
13	1000	9	2000	139350.69
14	1000	9	?	139350.69
15	1000	?	?	331204.72
16	?	?	?	862404.35

This is Answer Set # 2 from our previous slide that performed Group by Rollup

The Answer Set Tab shows the Results

Execute  Send SQL to Nexus

Objects Columns Sort Joins Where SQL Answer Set Metadata Analytics

Results 1 Results 2 Results 3

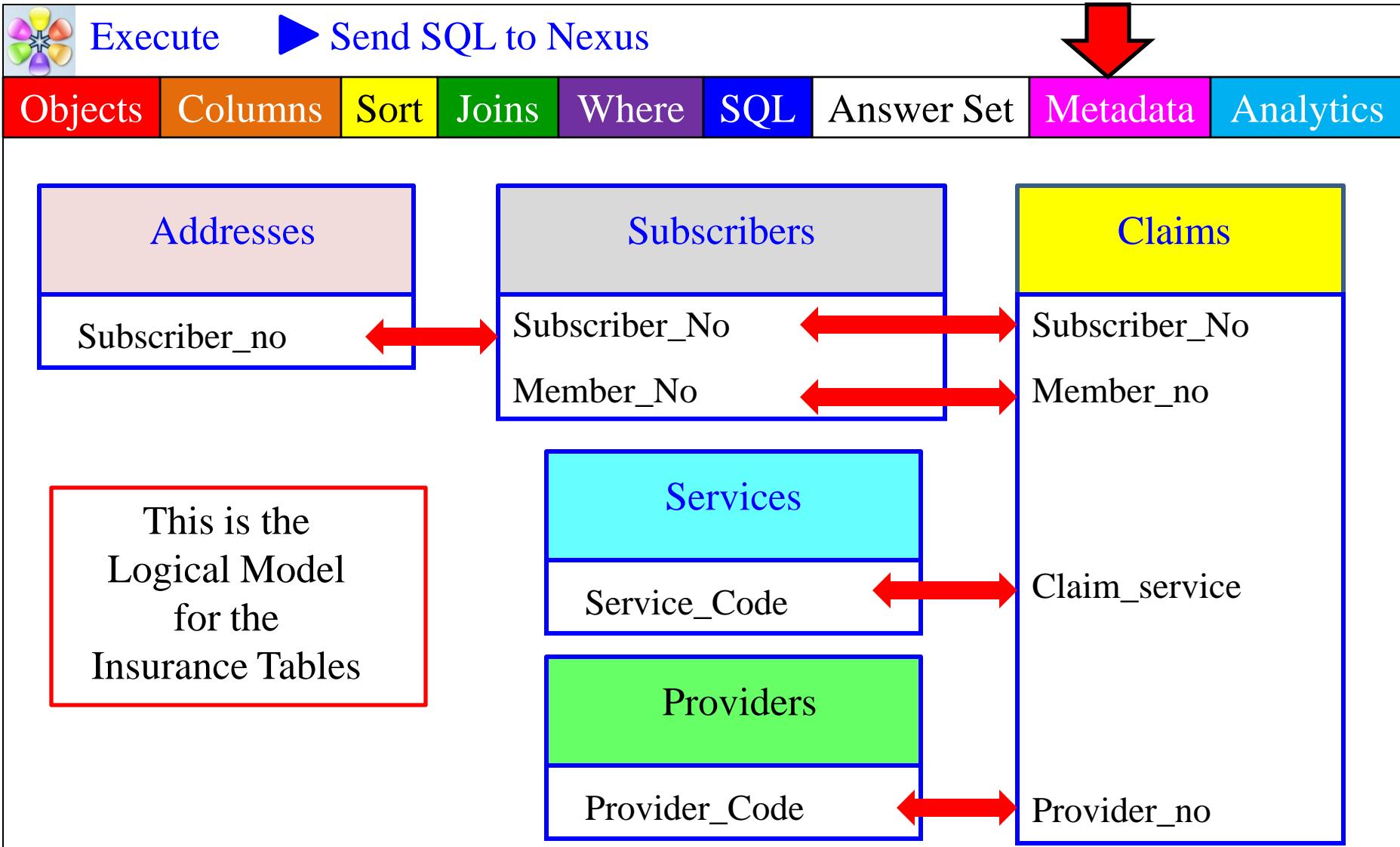
Drag a column here to group by that column

Product_ID	Mth	Year	Sum_Daily_Sales	
1	3000	10	2000	84908.06
2	3000	10	?	84908.06
3	3000	9	2000	139679.76
4	3000	9	?	139679.76
5	3000	?	2000	224587.82
6	3000	?	?	224587.82
7	2000	10	2000	166872.90
8	2000	10	?	166872.90
9	2000	9	2000	139738.91
10	2000	9	?	139738.91
11	2000	?	2000	306611.81
12	2000	?	?	306611.81
13	1000	10	2000	191854.03
14	1000	10	?	191854.03
15	1000	9	2000	139350.69
16	1000	9	?	139350.69

This is Answer Set # 3 from our previous slide that performed Group by Cube

Entire result set not displayed

The Metadata Tab shows Metadata



The Metadata tab allows a company to create Metadata for each table or object. This can be a word document, Power Point slide or whatever the company decides.

Nexus Makes a View look like a Table

Objects

Columns

Sort

Joins

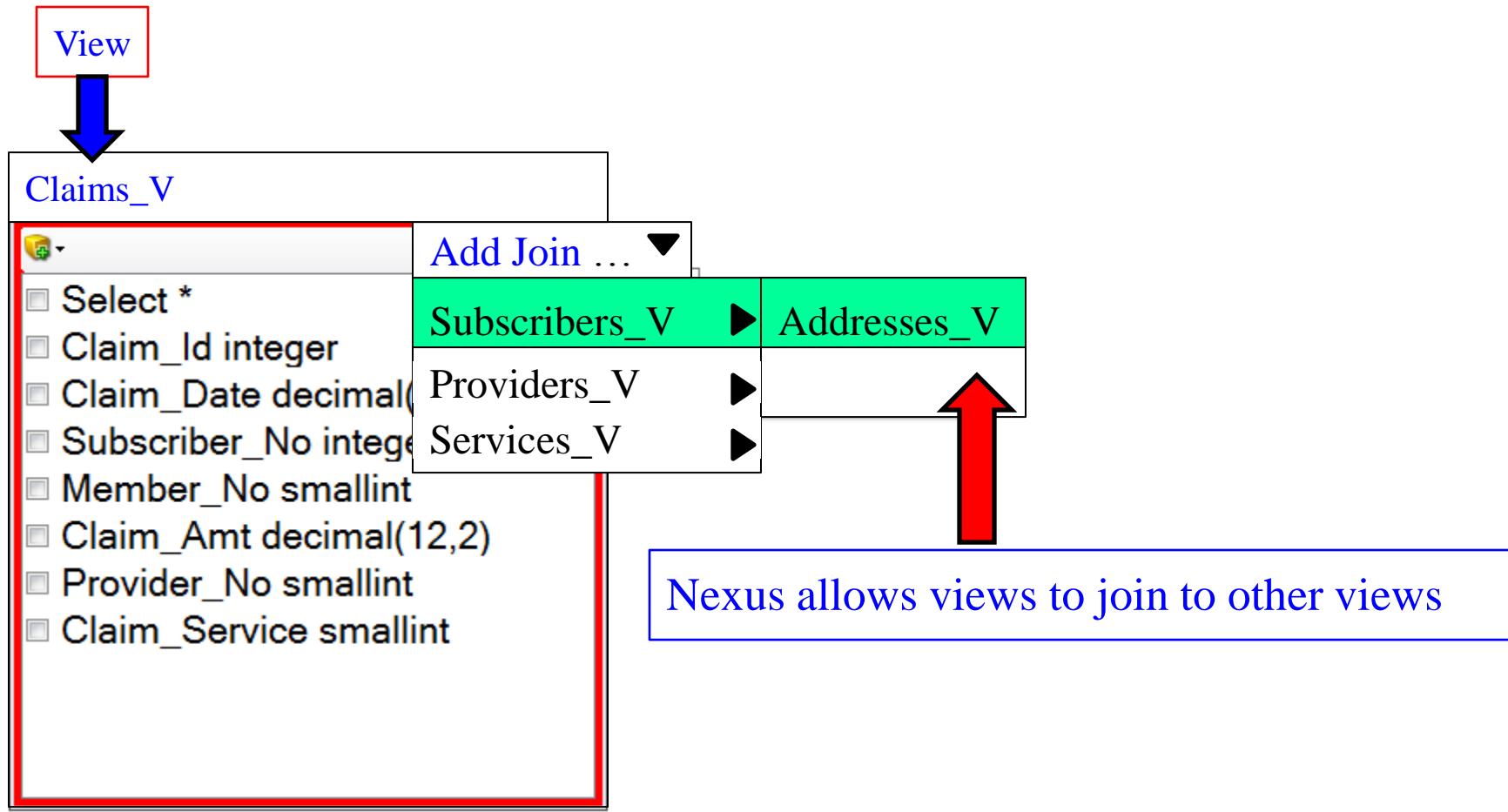
Where

SQL

Answer Set

Metadata

Analytics



Views in the Super Join Builder look like tables and can join to other views in a menu-driven fashion. Views looking like tables is brilliant and quite beneficial.

Nexus Joins Views to other Views in seconds

Objects Columns Sort Joins Where SQL Answer Set Metadata Analytics

<p>Claims_V</p> <p><input checked="" type="checkbox"/> Select *</p> <p><input checked="" type="checkbox"/> Claim_Id integer</p> <p><input checked="" type="checkbox"/> Claim_Date decimal(9,2)</p> <p><input checked="" type="checkbox"/> Subscriber_No integer</p> <p><input checked="" type="checkbox"/> Member_No smallint</p> <p><input checked="" type="checkbox"/> Claim_Amt decimal(12,2)</p> <p><input checked="" type="checkbox"/> Provider_No smallint</p> <p><input checked="" type="checkbox"/> Claim_Service smallint</p>	<p>Subscribers_V</p> <p><input type="checkbox"/> Select *</p> <p><input checked="" type="checkbox"/> Last_Name varchar(20)</p> <p><input checked="" type="checkbox"/> First_Name varchar(20)</p> <p><input type="checkbox"/> Gender char(1)</p> <p><input type="checkbox"/> SSN integer</p> <p><input type="checkbox"/> Member_No smallint</p> <p><input type="checkbox"/> Subscriber_No integer</p>	<p>Addresses_V</p> <p><input type="checkbox"/> Select *</p> <p><input checked="" type="checkbox"/> Street varchar(30)</p> <p><input checked="" type="checkbox"/> City varchar(20)</p> <p><input checked="" type="checkbox"/> State char(2)</p> <p><input checked="" type="checkbox"/> Zip integer</p> <p><input type="checkbox"/> AreaCode smallint</p> <p><input type="checkbox"/> Phone integer</p> <p><input type="checkbox"/> Subscriber_No integer</p>
--	--	---

With a few clicks of the mouse the SQL to join these three views is complete

Views looking like tables is brilliant and quite beneficial. We can even join views to other views in seconds and pick the columns we want on our report.

Nexus can Cube a View and Join to all other related Views

This tab displays the objects in your Query

Objects

Columns

Sort

Joins

Where

SQL

Answer Set

Metadata

Analytics

Left click on the Left Side
of a View and select
Create Cube with Columns

Claims_V

Add Join ▾

Create Cube

Create Cube with Columns

- Select *
- Claim_Id integer
- Claim_Date decimal(9,2)
- Subscriber_No integer
- Member_No smallint
- Claim_Amt decimal(12,2)
- Provider_No smallint
- Claim_Service smallint

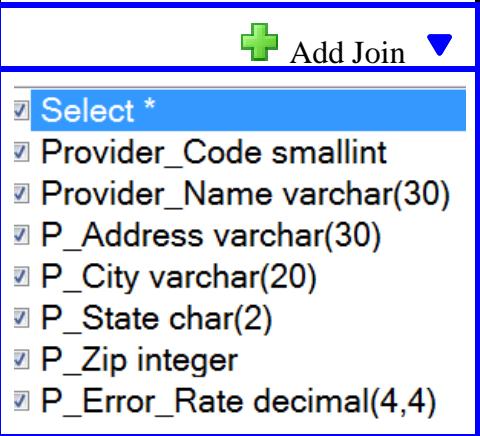
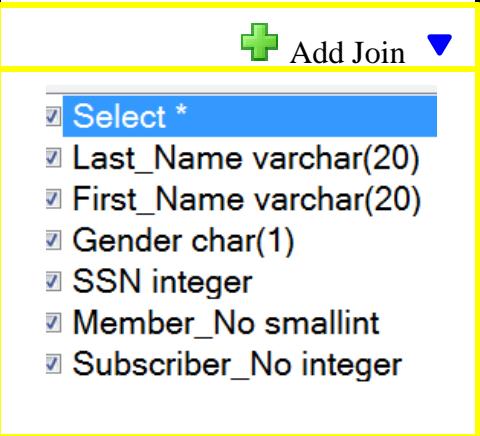
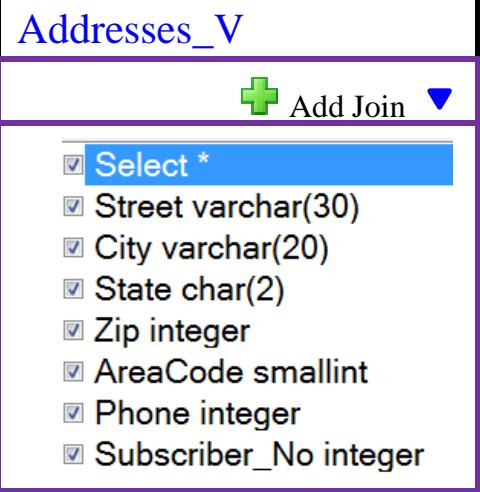
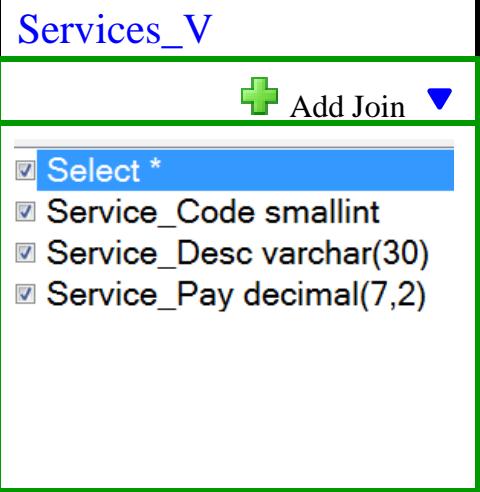
Check out the
next slide to see
what happens next.

When you click on the left side of a View in the Objects Tab and select CUBE the Nexus will create the SQL to join every View in the entire relationship together. Watch what happens on the next slide.

Nexus Cubes Views in Seconds

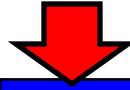
Objects Columns Sort Joins Where SQL Answer Set Metadata Analytics



Claims_V  + Add Join ▾ <input checked="" type="checkbox"/> Select * <input checked="" type="checkbox"/> Claim_Id integer <input checked="" type="checkbox"/> Claim_Date decimal(9,2) <input checked="" type="checkbox"/> Subscriber_No integer <input checked="" type="checkbox"/> Member_No smallint <input checked="" type="checkbox"/> Claim_Amt decimal(12,2) <input checked="" type="checkbox"/> Provider_No smallint <input checked="" type="checkbox"/> Claim_Service smallint	Providers_V  + Add Join ▾ <input checked="" type="checkbox"/> Select * <input checked="" type="checkbox"/> Provider_Code smallint <input checked="" type="checkbox"/> Provider_Name varchar(30) <input checked="" type="checkbox"/> P_Address varchar(30) <input checked="" type="checkbox"/> P_City varchar(20) <input checked="" type="checkbox"/> P_State char(2) <input checked="" type="checkbox"/> P_Zip integer <input checked="" type="checkbox"/> P_Error_Rate decimal(4,4)	Subscribers_V  + Add Join ▾ <input checked="" type="checkbox"/> Select * <input checked="" type="checkbox"/> Last_Name varchar(20) <input checked="" type="checkbox"/> First_Name varchar(20) <input checked="" type="checkbox"/> Gender char(1) <input checked="" type="checkbox"/> SSN integer <input checked="" type="checkbox"/> Member_No smallint <input checked="" type="checkbox"/> Subscriber_No integer
Addresses_V  + Add Join ▾ <input checked="" type="checkbox"/> Select * <input checked="" type="checkbox"/> Street varchar(30) <input checked="" type="checkbox"/> City varchar(20) <input checked="" type="checkbox"/> State char(2) <input checked="" type="checkbox"/> Zip integer <input checked="" type="checkbox"/> AreaCode smallint <input checked="" type="checkbox"/> Phone integer <input checked="" type="checkbox"/> Subscriber_No integer	Services_V  + Add Join ▾ <input checked="" type="checkbox"/> Select * <input checked="" type="checkbox"/> Service_Code smallint <input checked="" type="checkbox"/> Service_Desc varchar(30) <input checked="" type="checkbox"/> Service_Pay decimal(7,2)	All five Views have now been joined by Nexus

After hitting Cube with all Columns all Views in the relationship are joined.

The Cube SQL created on Views is done Automagically



Objects	Columns	Sort	Joins	Where	SQL	Answer Set	Metadata	Analytics
---------	---------	------	-------	-------	-----	------------	----------	-----------

```
SELECT cla.Claim_Id, cla.Claim_Date, cla.Subscriber_No, cla.Member_No,  
cla.Claim_Amt, cla.Provider_No, cla.Claim_Service,  
sub.Last_Name, sub.First_Name, sub.Gender, sub.SSN,  
sub.Member_No, sub.Subscriber_No,  
"add".Street, "add".City, "add"."State", "add".Zip, "add".AreaCode, "add".Phone,  
"add".Subscriber_No,  
pro.Provider_Code, pro.Provider_Name, pro.P_Address, pro.P_City,  
pro.P_State, pro.P_Zip, pro.P_Error_Rate,  
ser.Service_Code, ser.Service_Desc, ser.Service_Pay  
FROM SQL_CLASS.CLAIMS_V cla  
INNER JOIN SQL_CLASS.SUBSCRIBERS_V sub  
    ON cla.Subscriber_No = sub.Subscriber_No  
        AND cla.Member_No = sub.Member_No  
INNER JOIN SQL_CLASS.ADDRESSES_V "add"  
    ON sub.Subscriber_No = "add".Subscriber_No  
INNER JOIN SQL_CLASS.PROVIDERS_V pro  
    ON cla.Provider_No = pro.Provider_Code  
INNER JOIN SQL_CLASS.SERVICES_V ser  
    ON cla.Claim_Service = ser.Service_Code;
```

This SQL created by hitting
CREATE Cube with Columns
is seen here joining five views.

Views with the Underlying Indexes of the Base Tables

This tab displays the objects in your Query

Objects

Columns

Sort

Joins

Where

SQL

Answer Set

Metadata

Analytics

Employee_V



Add Join

- Select *
- Emp_No integer
- Lname char(20)
- Fname Varchar(12)
- Sal Decimal(8,2)
- Dept Smallint

If we go to the WHERE Tab we can see the underlying indexes.
Turn the page and see!

We have just placed a view called Employee_V inside the Super Join Builder. We have selected all the columns. Now watch what happens when we go to the WHERE Tab.

WHERE Tab shows Views Underlying Base Table Indexes



Objects

Columns

Sort

Joins

Where

SQL

Answer Set

Metadata

Analytics

Columns Used in Report

Emp_No

Fname

Lname

Sal

Dept

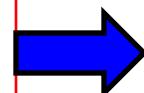
Unique Primary Index

Emp_No

Unique Secondary Index

Lname Fname

Indexes of the
underlying Base
Tables



Non-Unique Secondary Index

Lname

Dept

Sal

AND
OR
IN
BETWEEN
LIKE
NOT IN
= <>
>

Where

WHERE Emp_No = < value >

Click on any column or index and it
goes in the WHERE clause. Notice
here we clicked on Emp_No.

After an Answer Set Returns you can do many things

Execute  Send SQL to Nexus



Objects Columns Sort Joins Where SQL Answer Set Metadata Analytics

Results1

Drag a column here to group by that column

	Employee_No	Dept_No	Last_Name	First_Name	Salary
1	1324657	200	Coffing	Billy	41888.88
2	2000000	?	Jones	Squiggy	32800.50
3	1333454	200	Smith	John	48000.00
4	1232578	100	Chambers	Mandee	48850.00
5	1121334	400	Strickling	Cletus	54500.00
6	2312225	300	Larkins	Lorraine	40200.00
7	2312225	400	Reilly	William	36000.00
8	2341218	10	Smythe	Richard	64300.00

Options after the Answer Set Returns

OLAP 
Rank 
Pivot Result Set 
Bisualize 
Field Chooser 
Export/Save as 
Print Results 
Set Grid Font 

After the Answer Set returns the Nexus allows you to manipulate the results further. You can perform OLAP calculations, Rank and many more.

After an Answer Set Returns Perform OLAP Calculations

Objects	Columns	Sort	Joins	Where	SQL	Answer Set	Metadata	Analytics
OLAP Column		Sorting		Partitioning	Moving Window		OLAP Function	With Partitioning
Salary		Dept_No	ASC	Dept_No	3		SELECT *	SELECT *
							CSUM	CSUM
							MSUM	MSUM
Use Ctrl and Drag to move result grid column headers to OLAP Control								
Results1	Report1							
Drag a column here to group by that column								
<u>Employee No</u> <u>Dept No</u> <u>Last Name</u> <u>First Name</u> <u>Salary</u>								
1	1324657	200	Coffing	Billy	41888.88			
2	2000000	?	Jones	Squiggy	32800.50			
3	1333454	200	Smith	John	48000.00			
4	1232578	100	Chambers	Mandee	48850.00			
5	1121334	400	Strickling	Cletus	54500.00			
6	2312225	300	Larkins	Lorraine	40200.00			
7	2312225	400	Reilly	William	36000.00			
8	2341218	10	Smythe	Richard	64300.00			

We just performed OLAP on the answer set. Now the report comes (above in yellow), but what is amazing is that all calculations were performed internally by Nexus.

After an Answer Set Returns you can Graph and Chart

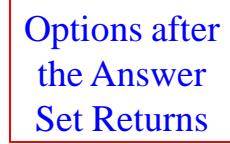
Execute  Send SQL to Nexus 

Objects Columns Sort Joins Where SQL Answer Set Metadata Analytics

Results1

Drag a column here to group by that column

	Employee_No	Dept_No	Last_Name	First_Name	Salary
1	1324657	200	Coffing	Billy	41888.88
2	2000000	?	Jones	Squiggy	32800.50
3	1333454	200	Smith	John	48000.00
4	1232578	100	Chambers	Mandee	48850.00
5	1121334	400	Strickling	Cletus	54500.00
6	2312225	300	Larkins	Lorraine	40200.00
7	2312225	400	Reilly	William	36000.00
8	2341218	10	Smythe	Richard	64300.00

Options after the Answer Set Returns 

OLAP 
Rank 
Pivot Result Set 
Bisualize 
Field Chooser 
Export/Save as 
Print Results 
Set Grid Font 



After the Answer Set returns the Nexus allows you to manipulate the results further. You can use Bisualize to Graph and Chart your answer set dynamically.



Find

Attributes

- Sale_Date
- Measures

Measures

- Product_ID
- Daily_Sales
- Values

Pages

Data

Filters

Level of Detail

Encodings

Markin

Label

Color

Size

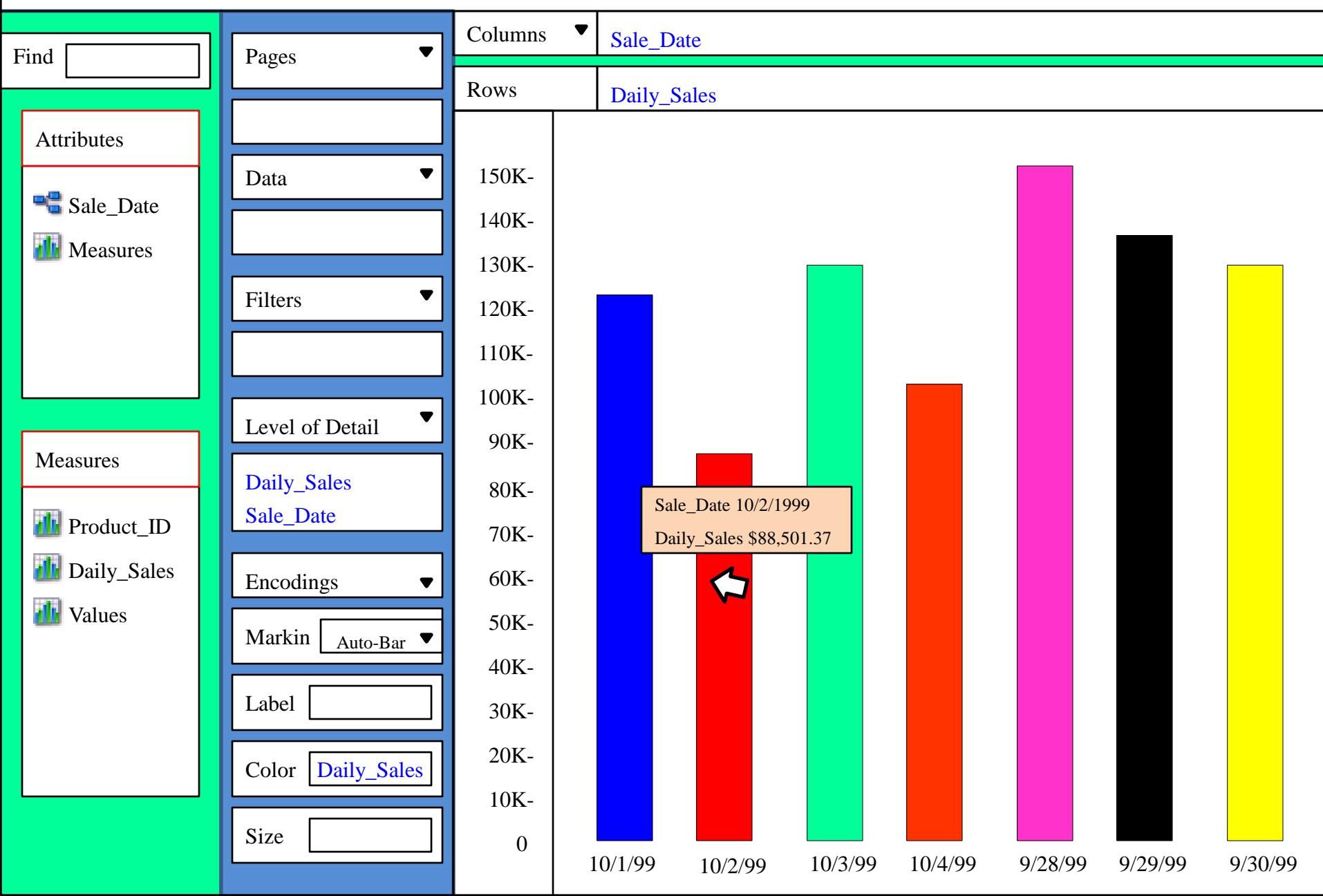
Columns

Rows

Bisualize Builds Dynamic Charts

Pay attention to the next slide because we are going to Drag and Drop the Attributes and Measures to their proper places and our chart will display beautifully!

Bisualize Builds Dynamic Charts



Saving an Answer Set in another Format

Objects Columns Sort Joins Where SQL **Answer Set** Metadata Analytics

Results1

Drag a column here to group by that column

	Employee No	Dept No	Last Name	First Name	Salary
1	1324657	200	Coffing	Billy	41888.88
2	2000000	?	Jones	Squiggy	32800.50
3	1333454	200	Smith	John	48000.00
4	1232578	100	Chambers	Mandee	48850.00
5	1121334	400	Strickling	Cletus	54500.00
6	2312225	300	Larkins	Lorraine	40200.00
7	2312225	400	Reilly	William	36000.00
8	2341218	10	Smythe	Richard	64300.00

Options after the Answer Set Returns

OLAP Rank Pivot Result Set Bisualize Field Chooser

Export/Save as Print Results Set Grid Font

Microsoft Excel (*.xlsx)
Microsoft Excel (97-2003) (*.xls)
Comma separated values (*.csv)
User delimited text (*.txt)
XML (*.xml)
Microsoft Access (*.mdb)

Export/Save As

You can save an Answer Set directly to Excel or to many different formats.

Sandbox – How to Create a Sandbox (1 of 5)

The screenshot shows a database management interface with a toolbar at the top containing various icons for New Query, ETL, Dashboard, Bisualize, History, Sandbox, Joins, and DBA. Below the toolbar, the system is set to Teradata and the database is SQL_Class. A red arrow points to the 'Sandbox' icon in the toolbar.

The main window displays a tree view of systems on the left, including DB2, Oracle, Netezza, Greenplum, ParAccel, Kognitio, PDW, Teradata, and Sandbox. A red box with an upward arrow highlights the 'Sandbox' entry, with the text "Your new Sandbox will then appear".

A central window titled "Sandbox: Create Tables" is open. It contains three steps:

- Step 1: Setup or Select Sandbox Database**
 - SQL Server Compact Edition (fast, holds up to 4 GB in storage and does not require a SQL Server installation) (selected)
 - Existing data source connection*

*Note: You can also drag and drop the query result tab onto the "Tables" folder in the systems tree
- Step 2: Create new database**
 - Create new database (A new data source connection will be created in your systems tree which will allow for querying and database administration.)
 - Save As... (button)
- Step 3: Save the database**
 - Check each table
 - Name the database and choose "Replica"
 - Select the result set in

A red arrow points to the "Create new database" button. Another red arrow points to the "Save As..." button. A third red arrow points to the "File name:" field where "Sandbox" is typed. A fourth red arrow points to the "Save" button at the bottom right.

On the right side of the interface, there is a large red box containing the text "Click on Sandbox", another red box containing "Click on Create new database", and a third red box containing "Give it a name and hit Save".

Sandbox - Join Answer Sets from different Systems (2 of 5)

System: Teradata ▼ Database: SQL_Class ▼ Execute

Systems	Query1 * X																																																					
+ I DB2 + O Oracle + N Netezza + G Greenplum + A ParAccel + Kognitio + M PDW + S Sandbox + Database + Tables + Oracle_Dept_Tbl	<p>This is your sandbox. Drag answer sets here and re-query or join them along with other answer sets.</p> <p>SELECT Employee_No ,Dept_No ,Last_Name ,First_Name ,Salary FROM SQL_CLASS.Employee_Table ;</p> <p>Messages Results1</p> <p>Drag a column here to group by that column</p> <table border="1"><thead><tr><th>Employee_No</th><th>Dept_No</th><th>Last_Name</th><th>First_Name</th><th>Salary</th></tr></thead><tbody><tr><td>1</td><td>1324657</td><td>200</td><td>Coffing</td><td>Billy</td><td>41888.88</td></tr><tr><td>2</td><td>2000000</td><td>?</td><td>Jones</td><td>Squiggy</td><td>32800.50</td></tr><tr><td>3</td><td>1333454</td><td>200</td><td>Smith</td><td>John</td><td>48000.00</td></tr><tr><td>4</td><td>1232578</td><td>100</td><td>Chambers</td><td>Mandee</td><td>48850.00</td></tr><tr><td>5</td><td>1121334</td><td>400</td><td>Strickling</td><td>Cletus</td><td>54500.00</td></tr><tr><td>6</td><td>2312225</td><td>300</td><td>Larkins</td><td>Lorraine</td><td>40200.00</td></tr><tr><td>7</td><td>2312225</td><td>400</td><td>Reilly</td><td>William</td><td>36000.00</td></tr><tr><td>8</td><td>2341218</td><td>10</td><td>Smythe</td><td>Richard</td><td>64300.00</td></tr></tbody></table>	Employee_No	Dept_No	Last_Name	First_Name	Salary	1	1324657	200	Coffing	Billy	41888.88	2	2000000	?	Jones	Squiggy	32800.50	3	1333454	200	Smith	John	48000.00	4	1232578	100	Chambers	Mandee	48850.00	5	1121334	400	Strickling	Cletus	54500.00	6	2312225	300	Larkins	Lorraine	40200.00	7	2312225	400	Reilly	William	36000.00	8	2341218	10	Smythe	Richard	64300.00
Employee_No	Dept_No	Last_Name	First_Name	Salary																																																		
1	1324657	200	Coffing	Billy	41888.88																																																	
2	2000000	?	Jones	Squiggy	32800.50																																																	
3	1333454	200	Smith	John	48000.00																																																	
4	1232578	100	Chambers	Mandee	48850.00																																																	
5	1121334	400	Strickling	Cletus	54500.00																																																	
6	2312225	300	Larkins	Lorraine	40200.00																																																	
7	2312225	400	Reilly	William	36000.00																																																	
8	2341218	10	Smythe	Richard	64300.00																																																	



All queries on your sandbox Tables run internally on your PC!



Right Click on Results1 Tab and Rename the Tab.

Sandbox - Join Answer Sets from different Systems (3 of 5)

System: Teradata ▼ Database: SQL_Class ▼ Execute

Systems	Query1 * X																																																					
+ I DB2 + O Oracle + N Netezza + G Greenplum + A ParAccel + Kognitio + M PDW + S Sandbox + Database + Tables + Oracle_Dept_Tbl	<p>SELECT Employee_No ,Dept_No ,Last_Name ,First_Name ,Salary FROM SQL_CLASS.Employee_Table ;</p> <p>Messa Teradata_Emp_Tbl</p> <p>After Renaming just Drag this Tab to your Sandbox to the Tables Icon</p> <p>Drag a column here to group by that column</p> <table border="1"><thead><tr><th>Employee_No</th><th>Dept_No</th><th>Last_Name</th><th>First_Name</th><th>Salary</th></tr></thead><tbody><tr><td>1</td><td>1324657</td><td>200</td><td>Coffing</td><td>Billy</td><td>41888.88</td></tr><tr><td>2</td><td>2000000</td><td>?</td><td>Jones</td><td>Squiggy</td><td>32800.50</td></tr><tr><td>3</td><td>1333454</td><td>200</td><td>Smith</td><td>John</td><td>48000.00</td></tr><tr><td>4</td><td>1232578</td><td>100</td><td>Chambers</td><td>Mandee</td><td>48850.00</td></tr><tr><td>5</td><td>1121334</td><td>400</td><td>Strickling</td><td>Cletus</td><td>54500.00</td></tr><tr><td>6</td><td>2312225</td><td>300</td><td>Larkins</td><td>Lorraine</td><td>40200.00</td></tr><tr><td>7</td><td>2312225</td><td>400</td><td>Reilly</td><td>William</td><td>36000.00</td></tr><tr><td>8</td><td>2341218</td><td>10</td><td>Smythe</td><td>Richard</td><td>64300.00</td></tr></tbody></table>	Employee_No	Dept_No	Last_Name	First_Name	Salary	1	1324657	200	Coffing	Billy	41888.88	2	2000000	?	Jones	Squiggy	32800.50	3	1333454	200	Smith	John	48000.00	4	1232578	100	Chambers	Mandee	48850.00	5	1121334	400	Strickling	Cletus	54500.00	6	2312225	300	Larkins	Lorraine	40200.00	7	2312225	400	Reilly	William	36000.00	8	2341218	10	Smythe	Richard	64300.00
Employee_No	Dept_No	Last_Name	First_Name	Salary																																																		
1	1324657	200	Coffing	Billy	41888.88																																																	
2	2000000	?	Jones	Squiggy	32800.50																																																	
3	1333454	200	Smith	John	48000.00																																																	
4	1232578	100	Chambers	Mandee	48850.00																																																	
5	1121334	400	Strickling	Cletus	54500.00																																																	
6	2312225	300	Larkins	Lorraine	40200.00																																																	
7	2312225	400	Reilly	William	36000.00																																																	
8	2341218	10	Smythe	Richard	64300.00																																																	

Sandbox - Join Answer Sets from different Systems (4 of 5)

System: Teradata ▼ Database: SQL_Class ▼ Execute

Systems	Query1 * X																																																					
+ I DB2 + O Oracle + N Netezza + G Greenplum + A ParAccel + Kognitio + M PDW + S Sandbox + Database + Tables + Oracle_Dept_Tbl + Teradata_Emp_Tbl	<p>SELECT Employee_No ,Dept_No ,Last_Name ,First_Name ,Salary FROM SQL_CLASS.Employee_Table ;</p> <p>Messages Teradata_Emp_Tbl</p> <p>Drag a column here to group by that column</p> <table border="1"><thead><tr><th>Employee_No</th><th>Dept_No</th><th>Last_Name</th><th>First_Name</th><th>Salary</th></tr></thead><tbody><tr><td>1</td><td>1324657</td><td>200</td><td>Coffing</td><td>Billy</td><td>41888.88</td></tr><tr><td>2</td><td>2000000</td><td>?</td><td>Jones</td><td>Squiggy</td><td>32800.50</td></tr><tr><td>3</td><td>1333454</td><td>200</td><td>Smith</td><td>John</td><td>48000.00</td></tr><tr><td>4</td><td>1232578</td><td>100</td><td>Chambers</td><td>Mandee</td><td>48850.00</td></tr><tr><td>5</td><td>1121334</td><td>400</td><td>Strickling</td><td>Cletus</td><td>54500.00</td></tr><tr><td>6</td><td>2312225</td><td>300</td><td>Larkins</td><td>Lorraine</td><td>40200.00</td></tr><tr><td>7</td><td>2312225</td><td>400</td><td>Reilly</td><td>William</td><td>36000.00</td></tr><tr><td>8</td><td>2341218</td><td>10</td><td>Smythe</td><td>Richard</td><td>64300.00</td></tr></tbody></table>	Employee_No	Dept_No	Last_Name	First_Name	Salary	1	1324657	200	Coffing	Billy	41888.88	2	2000000	?	Jones	Squiggy	32800.50	3	1333454	200	Smith	John	48000.00	4	1232578	100	Chambers	Mandee	48850.00	5	1121334	400	Strickling	Cletus	54500.00	6	2312225	300	Larkins	Lorraine	40200.00	7	2312225	400	Reilly	William	36000.00	8	2341218	10	Smythe	Richard	64300.00
Employee_No	Dept_No	Last_Name	First_Name	Salary																																																		
1	1324657	200	Coffing	Billy	41888.88																																																	
2	2000000	?	Jones	Squiggy	32800.50																																																	
3	1333454	200	Smith	John	48000.00																																																	
4	1232578	100	Chambers	Mandee	48850.00																																																	
5	1121334	400	Strickling	Cletus	54500.00																																																	
6	2312225	300	Larkins	Lorraine	40200.00																																																	
7	2312225	400	Reilly	William	36000.00																																																	
8	2341218	10	Smythe	Richard	64300.00																																																	

We now have an Oracle Table and a Teradata Table in our sandbox.

Sandbox - Join Answer Sets from different Systems (5 of 5)

System: Teradata ▼ Database: SQ

Systems

- + I DB2
- + O Oracle
- + N Netezza
- + Greenplum
- + A ParAccel
- + Kognitio
- + M PDW
- + S Sandbox
- + T Teradat
- + S Sandbox
- + Database
 - + Tables
 - + Oracle_Dept_Tbl
 - + Teradata_Emp_Tbl

New Query ▼ ETL Dashboard Bisualize History Sandbox Joins DBA

Execute

Query1 * X

SELECT T.* , Mgr_No
FROM Oracle_Dept_Tbl as O
INNER JOIN
Teradata_Emp_Tbl as T
ON O.Dept_No = T.Dept_No ;

This query ran from our Sandbox internally on our PC!

Results 1

a column here to group by that column

Emp_No	Dept_No	Last_Name	First_Name	Salary	Mgr_No
1324657	200	Coffing	Billy	41888.88	1000234
2000000	?	Jones	Squiggy	32800.50	1000234
1333454	200	Smith	John	48000.00	1256349
1232578	100	Chambers	Mandee	48850.00	1256349
1121334	400	Strickling	Cletus	54500.00	1256349
2312225	300	Larkins	Lorraine	40200.00	1256349
2312225	400	Reilly	William	36000.00	1256349
2341218	10	Smythe	Richard	64300.00	1333454

We just did a Join on a Teradata and Oracle answer set

How to Compress the Tables in an Entire Database



Systems X | Query1 * X

- T Teradat
- S SQL_Class Right Click on the Database
 - Tables
 - + Addresses
 - + Claims
 - + Course_Table
 - + Customer_Table
 - + Department_Table
 - + Employee_Table
 - + Order_Table
 - + Providers
 - + Views
 - + Macros
 - + Procedures
 - + Triggers
 - + Join Indexes
 - + Functions

Show all objects
Create database
Create user
Drop database
Add to My Databases
SmartCompress database ←
SmartReplica database
SmartSync data
SmartSync objects
SmartScript
SmartDBAdmin
Refresh database
Properties
Refresh data source connection
Properties

How to Compress the Tables in an Entire Database

Execute Schedule Compress DB SmartCompress Suffix _sc

Compress Staged for Replace Reports Custom Values Log

Name	Size
- SQL_Class	110,080
+ Addresses	2,560
+ Claims	4,096
+ Course_Table	4,096
+ Customer_Table	5,120
+ Department_Table	5,120
+ Employee_Table	9,216
+ Emp_Job_Table	3,072
+ Hierarchy_Table	3,584
+ Job_Table	3,072
+ Names_Table	3,072
+ Order_Table	5,120
+ Providers	2,560

Processing Options

- Compress Only
- Compress and Replace
- Replace Only
- Compress In-Place (ALTER)

Advanced

- Rebuild Indexes
- Re-collect Statistics
- Load Data
- Replace from Original
- Drop Prior Compressed Table
- Delete Compressed Data
- Sample

Max Compress Values

Convert to CHAR

Add custom value

Checkmark the tables and select the Processing Options and hit EXECUTE 

How to Compress A Single Table

System: **Teradata** ▼ Database: **SQL_Class** ▼ Execute

Systems	X	Query1 * X	▼
- Teradata			
- SQL_Class			
- Tables			
+ Addresses	Right Click on the Table	<ul style="list-style-type: none"> Super Join Builder Quick Select View DDL Query Designer Query Templates SmartCompress Table SmartReplica Object SmartSync data SmartSync object SmartScript Data Cleanser Create Cube Open – All rows Open - Selection Drop tableRename table Refresh tableProperties	
+ Claims			
+ Course_Table			
+ Customer_Table			
+ Department_Table			
+ Employee_Table			
+ Order_Table			
+ Providers			
+ Views			
+ Macros			
+ Procedures			
+ Triggers			
+ Join Indexes			
+ Functions			

How to Compress a Single Table

Execute Schedule Compress DB SmartCompress Suffix _sc

Compress Staged for Replace Reports Custom Values Log

Name	Size
- SQL_Class	110,080
+ Addresses	2,560

Processing Options

- Compress Only
- Compress and Replace
- Replace Only
- Compress In-Place (ALTER)

Advanced

- Rebuild Indexes
- Re-collect Statistics
- Load Data
- Replace from Original
- Drop Prior Compressed Table
- Delete Compressed Data
- Sample

Max Compress Values

Convert to CHAR

Add custom value

Checkmark the table and select the Processing Options and hit EXECUTE 

Compression Reports



Execute



Schedule

Compress DB

SmartCompress

Suffix

_sc

Compress

Staged for Replace

Reports

Custom Values

Log

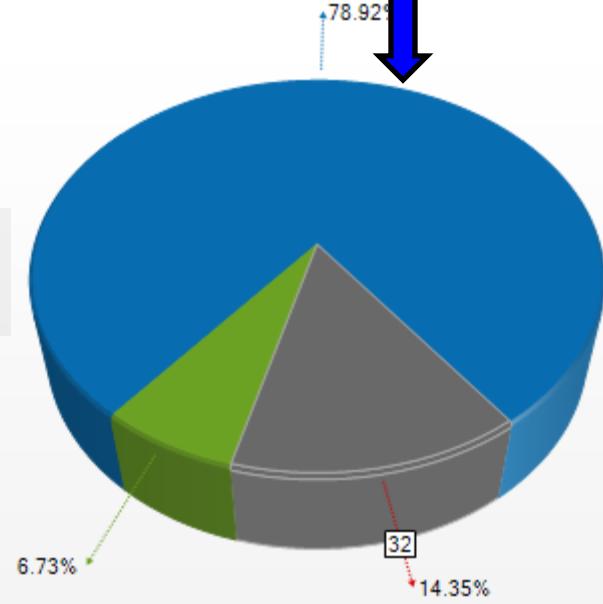
Report Name

Compress Savings

Compressible Columns Candidate Rank

Percentage of Compressed Savings Per Column

Compressed Savings and Cost Per Column



Let Nexus Build your Load Scripts with SmartScript

System: **Teradata** ▼ Database: **SQL_Class** ▼ Execute

Systems	X	Query1 * X	▼
- Teradata			
- SQL_Class			
- Tables			
+ Addresses	Right Click on the Table	Super Join Builder Quick Select View DDL Query Designer Query Templates SmartCompress Table SmartReplica Object SmartSync data SmartSync object SmartScript FastLoad Data Cleanser Create Cube Open – All rows Open - Selection Drop table Rename table Refresh table Properties	
+ Claims			
+ Course_Table			
+ Customer_Table			
+ Department_Table			
+ Employee_Table			
+ Order_Table			
+ Providers			
+ Views			
+ Macros			
+ Procedures			
+ Triggers			
+ Join Indexes			
+ Functions			

SmartScript Building a FastLoad

SmartScript FastLoad



Execute

System: Teradata [Edit...](#)

Quick Start

Script

Log

* Indicates required information

Build Script 

Database *

SQL_Class

Error Database *

SQL_Class

Table *

Addresses

Error Table 1 *

Addresses_ERR1

Error Table 2 *

Addresses_ERR2



Drop Existing Table



Drop Existing Table



Drop Existing Error Tables



Drop Existing Error Tables



Define Table

Standard

Advanced

Input

Type *

FILE

Format *

FORMATTED

Options

 Indicators

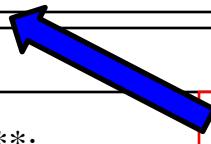
Delimiter

 Display Errors No Stop Insert Into Values Define Statement

Preview of Script

```
.LOGON Teradata/dbc,**PASSWORD**;  
SET RECORD FORMATTED ;
```

Put in the Input Filename and hit BUILD SCRIPT



SmartScript Building a FastLoad

Review/Edit your Script and hit Execute  or Schedule your Script to run 

SmartScript FastLoad

 ExecuteSystem: Teradata [Edit...](#)[Quick Start](#)[Script](#)[Log](#)

```
/* Please verify that you have the correct TDPIP. */
/* You can change the TDPIP by clicking Edit on the toolbar, and selecting a host file entry. */


```

```
.LOGON Teradata/dbc,**PASSWORD**;
```

```
SET RECORD FORMATTED;
```

```
DEFINE
```

```
Subscriber_No (INTEGER)
, Street (VARCHAR(30))
, City (VARCHAR(20))
, State (CHAR(2))
, Zip (INTEGER)
, AreaCode (SMALLINT)
, Phone (INTEGER)
```

```
FILE='C:\Temp\Addresses_Flatfile.txt';
```

```
BEGIN LOADING SQL_CLASS.Addresses
```

```
    ERRORFILES SQL_CLASS.Addresses_ERR1, SQL_CLASS.Addresses_ERR2
```

The Teradata DBA Launchpad



- Direct Data Export
- Direct Data Import
- SmartReplica
- DBAdmin
- AI DBA
- Create Database
- Create User
- DDL Builder
- View Builder
- Convert DDL
- Tera-Trivia
- Data Cleanser

DBA Launchpad

Please select a menu item on the left to get started



Convert Teradata DDL to Another Database Vendor

System: **Teradata** ▼ Database: **SQL_Class** ▼ Execute

The interface shows two main sections: 'Objects' on the left and 'To be converted' on the right. In the 'Objects' section, under 'Tables', several tables are listed with checkboxes: Addresses, Claims, Course_Table, Customer_Table, Department_Table, Employee_Table, Emp_Job_Table, Hierarchy_Table, Job_Table, Names_Table, Order_Table, and Providers. Most of these have checkboxes checked. In the 'To be converted' section, the same list of tables is shown, but the checkboxes are all unchecked. A large blue arrow points from the 'Objects' section to the 'To be converted' section. Below these sections are two dropdown menus: 'Convert DDL to' set to 'Netezza' and 'Convert To' set to 'Netezza'. A red arrow points upwards from the 'To be converted' section towards the top of the page.

1. Select the objects you would like to convert on the left
2. Click the blue arrow to send them into the conversion queue on the right
3. Finally, click the Convert DDL button to convert

Objects

<input checked="" type="checkbox"/>	SQL_Class
<input checked="" type="checkbox"/>	Tables
<input checked="" type="checkbox"/>	Addresses
<input checked="" type="checkbox"/>	Claims
<input checked="" type="checkbox"/>	Course_Table
<input checked="" type="checkbox"/>	Customer_Table
<input checked="" type="checkbox"/>	Department_Table
<input checked="" type="checkbox"/>	Employee_Table
<input checked="" type="checkbox"/>	Emp_Job_Table
<input checked="" type="checkbox"/>	Hierarchy_Table
<input checked="" type="checkbox"/>	Job_Table
<input checked="" type="checkbox"/>	Names_Table
<input checked="" type="checkbox"/>	Order_Table
<input checked="" type="checkbox"/>	Providers

To be converted

<input type="checkbox"/>	SQL_Class
<input type="checkbox"/>	Tables
<input type="checkbox"/>	Addresses
<input type="checkbox"/>	Claims
<input type="checkbox"/>	Course_Table
<input type="checkbox"/>	Customer_Table
<input type="checkbox"/>	Department_Table
<input type="checkbox"/>	Employee_Table
<input type="checkbox"/>	Emp_Job_Table
<input type="checkbox"/>	Hierarchy_Table
<input type="checkbox"/>	Job_Table
<input type="checkbox"/>	Names_Table
<input type="checkbox"/>	Order_Table
<input type="checkbox"/>	Providers

Convert DDL to

Netezza

CREATE TABLE ADDRESSES

(
SUBSCRIBER_NO INTEGER,
STREET VARCHAR(30),
CITY VARCHAR(20),
STATE CHAR(2),
ZIP INTEGER,
AREACODE SMALLINT,
PHONE INTEGER
,
UNIQUE (SUBSCRIBER_NO))
DISTRIBUTE ON (SUBSCRIBER_NO);

CREATE TABLE CLAIMS

(
CLAIM_ID INTEGER,
CLAIM_DATE DATE,
CLAIM_SERVICE SMALLINT,
SUBSCRIBER_NO INTEGER,
MEMBER_NO SMALLINT,
CLAIM_AMT NUMERIC(12,2),
PROVIDER_NO SMALLINT
,
UNIQUE (CLAIM_ID, CLAIM_DATE,
CLAIM_SERVICE))
DISTRIBUTE ON (CLAIM_ID,
CLAIM_DATE,
CLAIM_SERVICE);

CREATE TABLE Course_Table

(
Course_ID INTEGER,

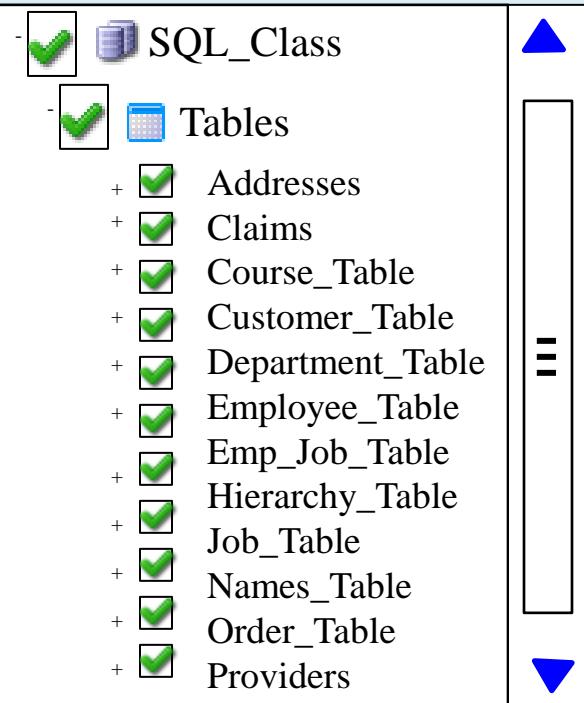
Replicate Data from One Teradata System to Another



SmartReplica

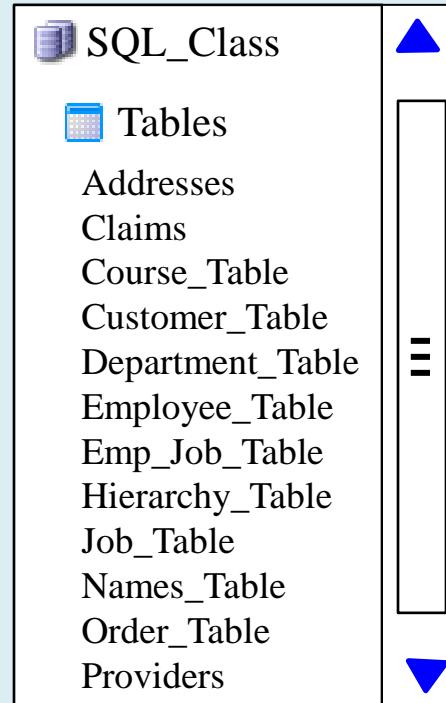
Source: Teradata Production

Source System



Target: Teradata Test

Target System- Package



Execute

Database Setting

SQL_Test

Same name as source

Object Setting

Same name as source

Data

Data Quantity Volume

All Data

None

Subset

Sample

Selection

Edit

Compare and Synchronize with SmartSync

System: **Teradata** ▼ Database: **SQL_Class** ▼ Execute

Systems	X	Query1 * X
- T Teradata - SQL_Class - Tables - Addresses + Claims + Course_Table + Customer_Table + Department_Table + Employee_Table + Order_Table + Providers + Views + Macros + Procedures + Triggers + Join Indexes + Functions	<p>Right Click on a Database</p> <p>OR</p> <p>Right Click on a Table</p> <p>Super Join Builder Quick Select View DDL Query Designer Query Templates SmartCompress Table SmartReplica Object SmartSync data SmartSync object SmartScript Data Cleanser Create Cube Open – All rows Open - Selection Drop table Rename table Refresh table Properties</p>	<p>SmartSync Data will allow you to compare the data on two different Teradata Systems, Databases or Tables and it will show you the differences.</p> <p>SmartSync Data will also be able to reconcile those differences!</p> <p>SmartSync Object will allow you to compare the DDL on two different Teradata Systems, Databases or Tables and it will show you the differences.</p>

Compare Data on Two tables with SmartSync

Execute



Source: Teradata

Target: Teradata [Edit...](#)

Choose your Target System
and then press Execute

Objects

Results

Synchronize

Source System

Database

SQL_Class ▾

Object

Employee_Table ▾

Target System

Database

SQL_Class_BKP ▾

Object

Employee_Table ▾

Object Settings (blue indicates Primary Index)

- Employee_Table
- Employee_No
- Dept_No
- Last_Name
- First_Name
- Salary (Decimal)

Data Quantity

All Data

Subset

Selection [Edit](#)

Options

Intersystem compare

Show identical rows

Difference Threshold

Compare Data and See the Results with SmartSync

Objects	Results	Synchronize
---------	---------	-------------

In Source, Not Target

- 1 1 row exists in Source Table Sql_Class.Employee_Table that does not exist in Target SQL_Class_BKP.Employee_Table

Employee_No	Dept_No	Last_Name	First_Name	Salary
2312225	300	Larkins	Lorraine	40200.00

In Target, Not Source

- 2 0 rows exist in Source Table Sql_Class.Employee_Table that does not exist in Target SQL_Class_BKP.Employee_Table

In Both with Differences

- 3 3 rows different in Source Table Sql_Class.Employee_Table and Target object SQL_Class_BKP.Employee_Table

	Employee_No	Dept_No	Last_Name	First_Name	Salary
SOURCE	1256349	400	Harrison	Herbert	54500.00
TARGET	1256349	400	Harrison	Herbby	54500.00
SOURCE	1324657	200	Coffing	Billy	41888.88
TARGET	1324657	200	Coffing	Billy	123.00
SOURCE	2000000	?	Jones	Squiggy	32800.50
TARGET	2000000	0	James	Squiggy	32800.50

Synchronize the Data with SmartSync

Execute



Source: Teradata

Target: Teradata [Edit...](#)

Objects

Results

Synchronize

Synchronize **Target** Object to **Source** Object

- Insert missing rows
- Delete extra rows
- Update differences with matching rows



Perform Synchronization



Preview in Query Window

Synchronize **Source** Object to **Target** Object

- Insert missing rows
- Delete extra rows
- Update differences with matching rows



Perform Synchronization



Preview in Query Window

You have the ability to choose how you want to synchronize!

Compare DDL with SmartSync

Execute 



Schedule

Source: Teradata

Target: Teradata [Edit...](#)

Objects

Results

Log

Compare Level

Object

Database

System

Source System

Database

SQL_Class ▾

Object

Employee_Table ▾

Target System

Database

SQL_Class_BKP ▾

Object

Employee_Table ▾

Compare Options

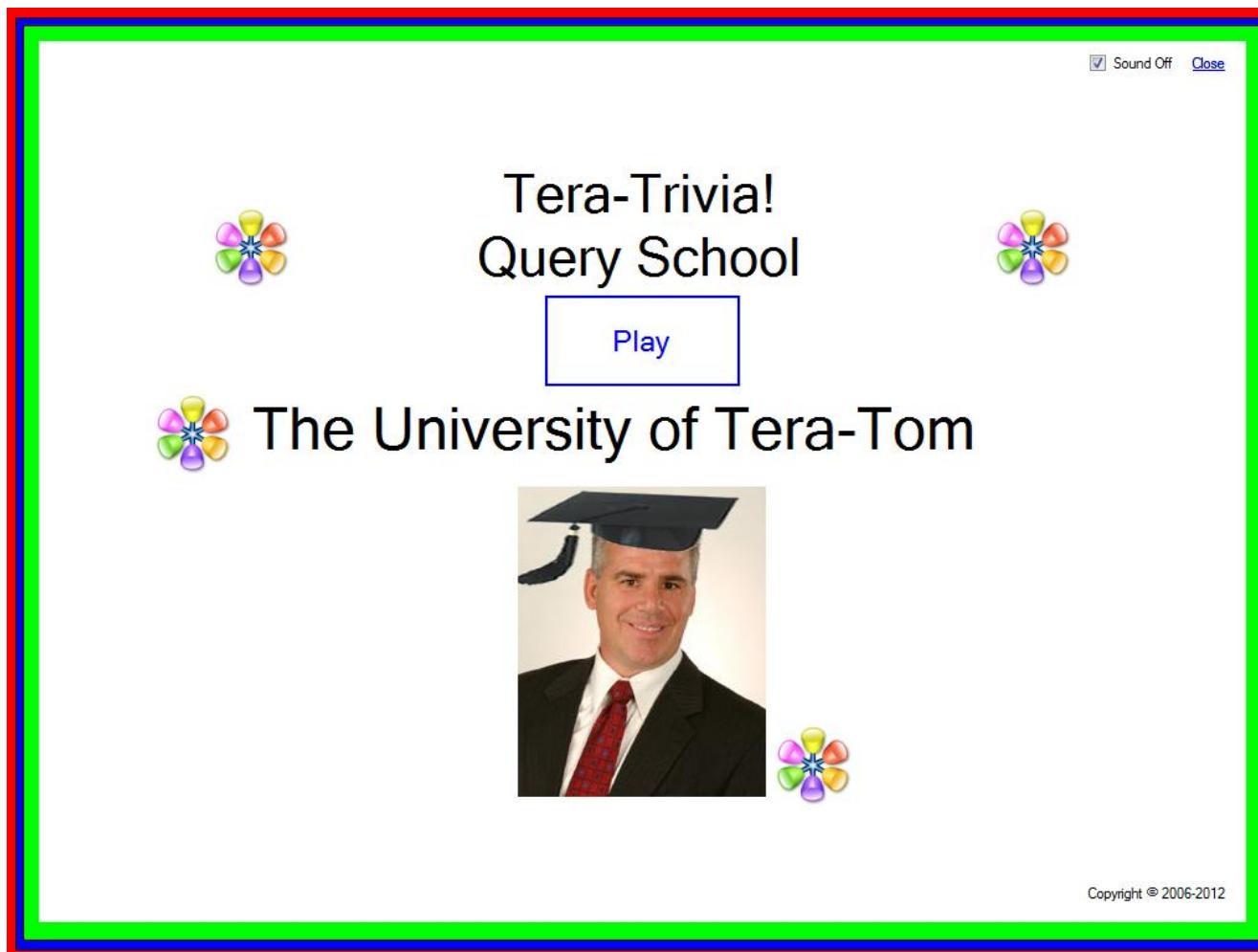
Select Function

- Select/Unselect All
 - + Compare Object Headers
 - + Compare Columns
 - + Compare Indexes
 - + Compare Statistics
 - + Compare Definitions
 - + Compare Object Rights

Advanced

Display	Filters	Definition	Log
<input type="checkbox"/> Unchanged Rows	<input checked="" type="checkbox"/> Changed Rows	<input checked="" type="checkbox"/> Unmatched Rows in Source	<input checked="" type="checkbox"/> Unmatched Rows in Target
<input type="checkbox"/> Counts and Results ▾	<input type="checkbox"/> Show Line Numbers	<input checked="" type="checkbox"/> Silent Processing	

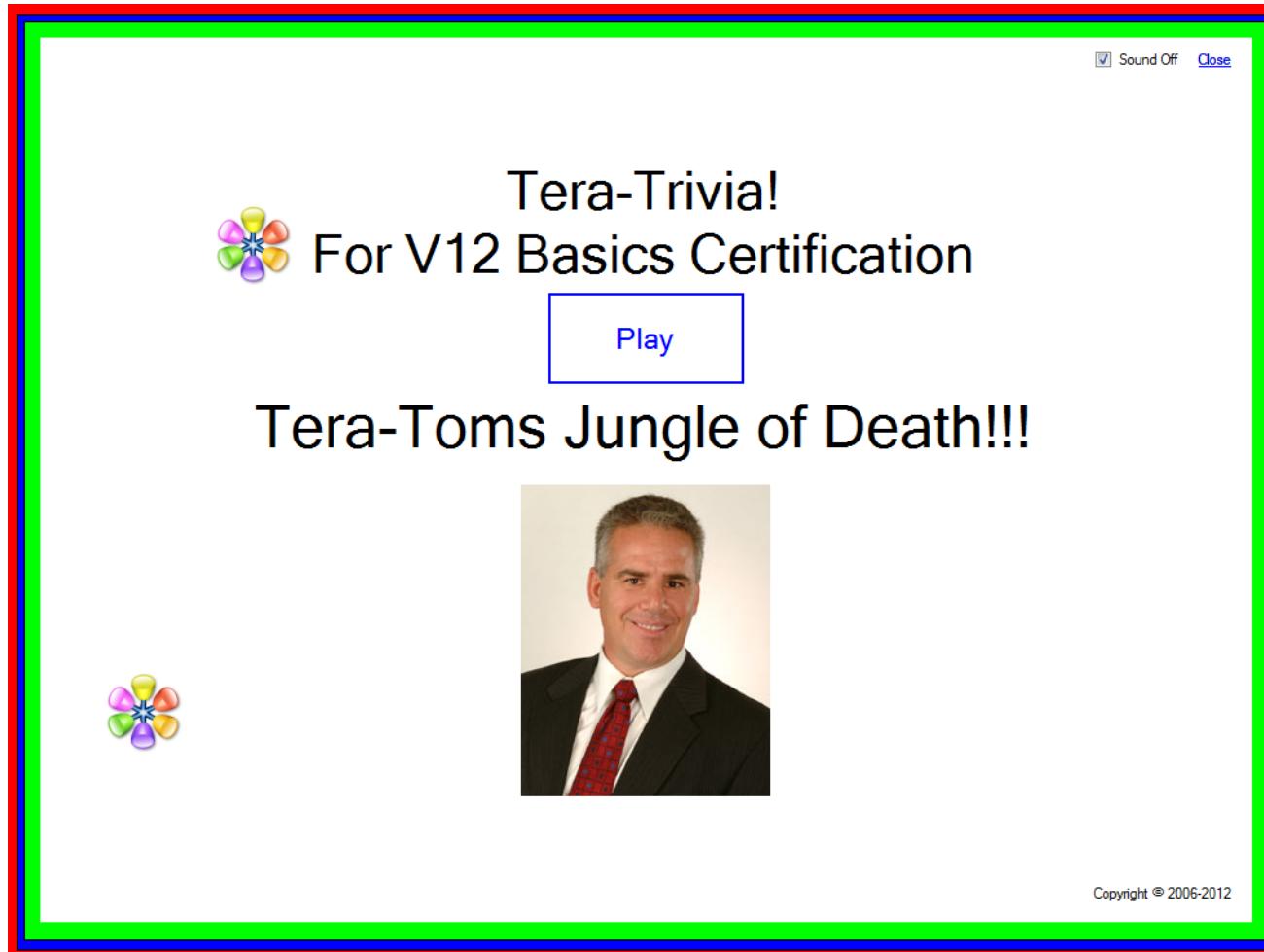
Play Games and Learn Teradata SQL



Copyright © 2006-2012

Just go to the DBA button and then select Tera-Trvia. You will have the option of playing Teradata Basics or Teradata SQL games. These are very advanced games designed to test and teach your Teradata skills.

Play Games and Learn Teradata Architecture



Just go to the DBA button and then select Tera-Trivia. You will have the option of playing Teradata Basics or Teradata SQL games. These are very advanced games designed to test and teach your Teradata skills.

Watch the Video on India



Tera-Tom Trivia

Tom Coffing has taught Teradata classes in China, India, Africa, Malaysia, Europe, Canada, Mexico and throughout the United States. Tom has taught over 1,000 live Teradata classes in his 20 years of Teradata experience.

Click on the link below or place it in your browser and watch the video on Tera-Tom's trip to India

<http://www.coffingdw.com/TbasicsV12/India.wmv>