CONTENTS:

Distinct Vs. Group By

Distinct Vs. Group By

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

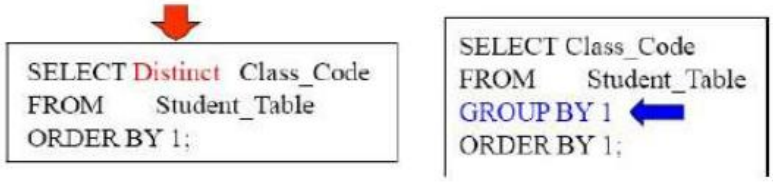
```
SELECT Distinct  Class_Code
FROM     Student_Table
ORDER BY 1;
```

```
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.

GROUP BY Vs. DISTINCT – Good Advice

It is better to use this example

```
Select First_Name, Last_Name From Employee_Table
Group By 1,2;
```

This example could cause skewing

```
Select DISTINCT First_Name, Last_Name From
Employee_Table;
```

Ad Hoc users should use the GROUP BY example above instead of the DISTINCT example. Both queries will provide the same results, but the GROUP BY could be significantly faster. Teradata has improved this issue with Teradata V13, but in some cases, the DISTINCT still causes skewing and is slower.

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;
```

Class_Code	Grade_Pt
?	?
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.

Explain

Explain

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 all-AMPs RETRIEVE by way of an all-rows scan, that means that Teradata is doing a Full Table Scan, and thus it is reading every row in the table. 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 = 'Loraine'
            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 = 'Loraine'" 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 beforehand.

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.

Format Functions

Format Functions

The FORMAT Command

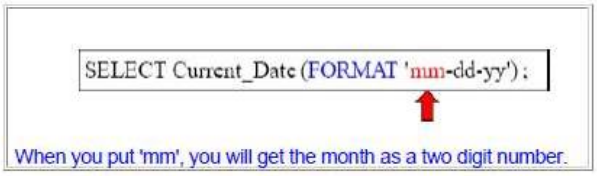
```
SELECT Current_Date (FORMAT 'mm-dd-yy') ;
```

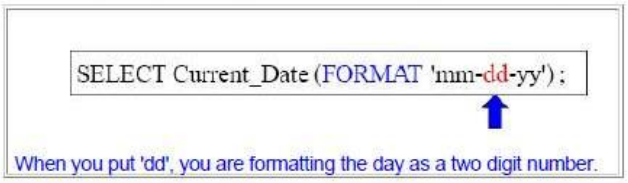
DATE
05-07-13

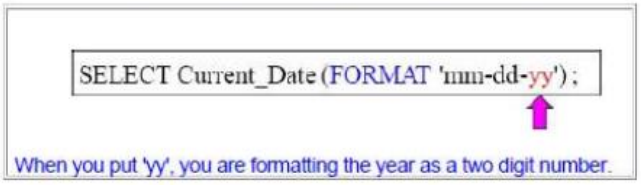
Today's date has been formatted for a 2-digit year!

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







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, 2013
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, 2013
SELECT Current_Date (FORMAT'mm-dd-yy') ;
How will the answer appear?
ANSWER: 03-20-13
```

Quiz – How will the Date Appear after Formatting

```
Current Date = March 20th, 2013
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, 2013
SELECT Current_Date (FORMAT'mm-dd-yyyy') ;
How will the answer appear?
ANSWER: 03-20-2013
```

Formatting with MMM for the Abbreviated Month

```
Current Date = March 20th, 2013
```

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, 2013
```

```
SELECT Current_Date (FORMAT 'mmm-dd-yyyy');
```

```
How will the answer appear?
You get the first three letters of the month!

ANSWER: Mar-20-2013
```

Formatting with MMMM for the Full Month Name

```
Current Date = March 20th, 2013
```

```
SELECT Current_Date (FORMAT 'mmmm-dd-yyyy');
```

```
How will the answer appear?
```

Formatting with MMMM for the Full Month

```
Current Date = March 20th, 2013
```

```
SELECT Current_Date (FORMAT 'mmmm-dd-yyyy');
```

How will the answer appear?

ANSWER: March-20-2013

Formatting with DDD for the Julian Day

Current Date = March 20th, 2013

```
SELECT Current_Date (FORMAT 'mm-ddd-yyyy');
```

How will the answer appear?

ANSWER: 03-079-2013

Julian Date

Formatting with DDD for the Julian Day

Current Date = March 20th, 2013

```
SELECT Current_Date (FORMAT 'mm-ddd-yyyy');
```

How will the answer appear?

?

Formatting with EEE or EEEE for the Day of the Week

Current Date = March 20th, 2013

```
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, 2013

How will the answers appear?

```
SELECT Current_Date (FORMAT 'eee-mm-ddd-yyyy');
```

ANSWER: Sun-03-20-13

There are 3 e's in front!

```
SELECT Current_Date (FORMAT 'eeee-mm-ddd-yyyy');
```

ANSWER: Sunday-03-20-13

There are 4 e's in front!

Placing Spaces inside your Formatting Commands with a B

```
Current Date = March 20th, 2013
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');
```

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 5133000346 (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 5133000346 (FORMAT '999-999-9999');
```

How will the answer appear?

513-300-0346

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 '5133000346' (FORMAT '99-999-9999');
```

How will the answer appear?

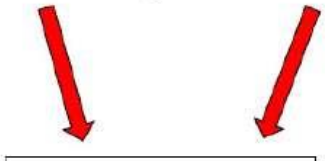
?

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 '5133000346' (FORMAT '99-999-9999');
```



****

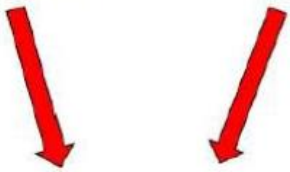
This is not an error, but something is wrong!

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?

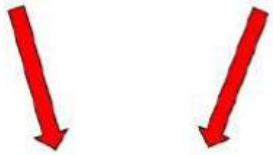
?

You can also FORMAT characters. Look at this example. It doesn't matter if the X's are capitalized or not.

Formatting with 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?

?

A '9' in a format statement means 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');
```

How will the answer appear?

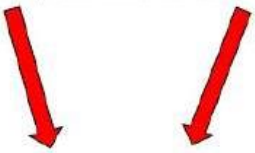
1021.53

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');
```



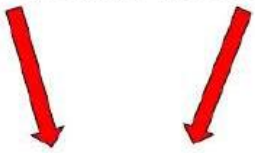
How will the answer appear?

?

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');
```



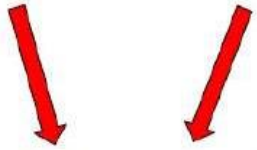
How will the answer appear?

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');
```



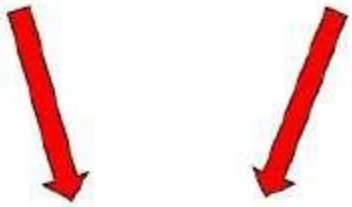
How will the answer appear?

?

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');



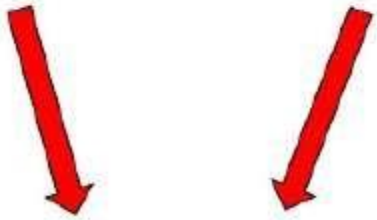
How will the answer appear?

ANSWER: $1021.53

Formatting with $ and Commas

You can also use commas in your Formatting statements.

SELECT 1021.53 (FORMAT '$,$$$,$$9.99');

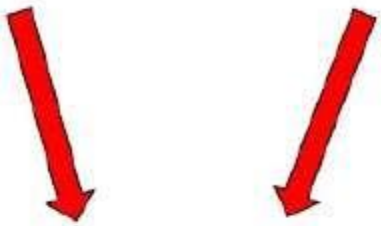


How will the answer appear?

Formatting with $ and Commas Visual

You can also use commas in your Formatting statements.

SELECT 1021.53 (FORMAT '$,$$$,$$9.99');



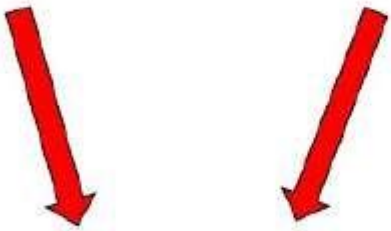
How will the answer appear?

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 0.53 (FORMAT '$,$$$,$$9.99');

How will the answer appear?

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
03-20-13  03-20-2013 Mar-20-2013    March-20-13  03-079-2013  Sun 03-20-13 Sunday 03-20-13
```

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 present 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 to 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'(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 ANSI Mode, a capital 'A' is seen as different in comparison to 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
```

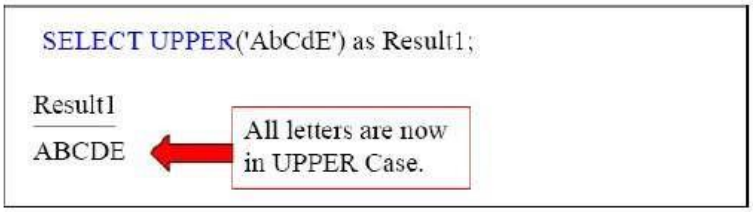
All Capitol letters are now in LOWER Case

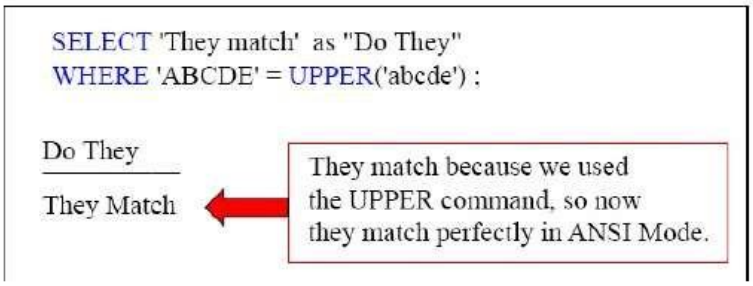
```
SELECT 'They match' as "Do They"
WHERE LOWER('ABCDE') = 'abcde';

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 to a little 'a'. The LOWER command can be used to make sure the case of a column is lowered.

Using the UPPER Command





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.

HELP and SHOW

HELP and SHOW

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     T       ?
 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

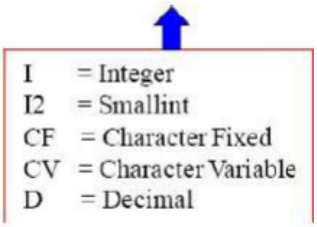
U	UDT
V	View
X	Authorization

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	C F	?	Y	X(20)	?	20
First_Name	CV	?	Y	X(12)	?	12
Salary	D	?	Y	-------.99	?	4



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	ALTER TABLE	BEGIN LOGGING

BEGIN TRANSACTION	CHECKPOINT	COLLECT STATISTICS
COMMIT	COMMENT	CREATE DATABASE
CREATE INDEX	CREATE MACRO	CREATE TABLE
CREATE USER	CREATE VIEW	DATABASE
DELETE	DELETE DATABASE	DELETE USER
DBS SQL Functions		
ABS	ADD_MONTHS	AVERAGE
CHARACTERS	CAST	CHAR2HEXINT
COUNT	CORR	COVAR_POP
CSUM	EXP	EXTRACT
FORMAT	INDEX	HASHAMP
HASHBKAMP	HASHBUCKET	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 FALLBACK,
  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))
   FROMMJL01.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.

Join Functions

Join Functions

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 ;
```

Fully Qualified with CUST

CUST is now Table ALIAS

ORD in now Table ALIAS

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 else we receive an error!

A two-table join using ANSI Syntax

Customer_Table

Customer_Number	Customer_Name
11111111	Billy's Best Choice
31313131	Acme Products
31323134	ACE Consulting
57896883	XYZ Plumbing
87323456	Databases N-U

Order_Table

Order_Number	Customer_Number	Order_Total
123456	11111111	12347.53
123512	11111111	8005.91
123552	31323134	5111.47
123585	87323456	15231.62
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 ;
```

INNER JOIN Keyword replaces the comma

ON Keyword is used instead of WHERE

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

Customer_Number	Customer_Name
11111111	Billy's Best Choice
31313131	Acme Products
31323134	ACE Consulting
57896883	XYZ Plumbing
87323456	Databases N-U

Order_Table

Order_Number	Customer_Number	Order_Total
123456	11111111	12347.53
123512	11111111	8005.91
123552	31323134	5111.47
123585	87323456	15231.62
123777	57896883	23454.84

```
/* Traditional Syntax */

SELECT Cust.Customer_Number,
       Customer_Name,
       Order_Number,
       Order_Total
FROM   Customer_Table as Cust,
```

```
/* ANSI 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 ;
```

```
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	Loraine	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	Loraine	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

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	Loraine	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

```
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

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	Loraine	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

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

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	Loraine	40200.00
2000000	?	Jones	Squiggy	32800.50
1000234	10	Smythe	Richard	32800.00
1121334	400	Strickling	Cletus	54500.00

Department_Table

Dept_No	Department_Name
100	Marketing
200	Research and Dev
300	Sales
400	Customer Support
500	Human Resources

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 ;
```

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	Loraine	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	Loraine	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.

LEFT OUTER JOIN Brings Back All Rows in 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	Loraine	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
Loraine	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

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	Loraine	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

```
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.

RIGHT OUTER JOIN Brings Back All Rows in 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	Loraine	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
Loraine	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

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	Loraine	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

```
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!

This 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.

FULL OUTER JOIN Brings Back All Rows in All Tables

Employee_Table

Employee_No	Dept_No	Last_Name	First_Name	Salary

Department_Table

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	Loraine	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		

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
Loraine	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!

Which Tables are the Left and which are the Right?

```
SELECT Cla.Claim_Id,
       Cla.Claim_Date,
       SUB.Last_Name,
       SUB.First_Name,
       "ADD".Phone,
       SER.Service_Pay,
       PRO.Provider_Code,
       PRO.Provider_Name,
FROM CLAIMS Cla
LEFT OUTER JOIN PROVIDERS PRO
       ON Cla.Provider_No = PRO.Provider_Code
LEFT OUTER JOIN SERVICES SER
       ON Cla.Claim_Service = SER.Service_Code
LEFT OUTER JOIN SUBSCRIBERS SUB
       ON Cla.Subscriber_No = SUB.Subscriber_No
       AND Cla.Member_No = SUB.Member_No
LEFT OUTER JOIN ADDRESSES "ADD"
       ON SUB.Subscriber_No = "ADD".Subscriber_No;
```

Your mission is to show which tables are Left Tables and which ones are Right Tables.

Answer - Which Tables are the Left and which are the Right?

```
SELECT Cla.Claim_Id,
       Cla.Claim_Date,
       SUB.Last_Name,
       SUB.First_Name,
       "ADD".Phone,
       SER.Service_Pay,
       PRO.Provider_Code,
       PRO.Provider_Name,
FROM CLAIMS Cla
LEFT OUTER JOIN PROVIDERS PRO
       ON Cla.Provider_No = PRO.Provider_Code
LEFT OUTER JOIN SERVICES SER
       ON Cla.Claim_Service = SER.Service_Code
LEFT OUTER JOIN SUBSCRIBERS SUB
       ON Cla.Subscriber_No = SUB.Subscriber_No
       AND Cla.Member_No = SUB.Member_No
LEFT OUTER JOIN ADDRESSES "ADD"
       ON SUB.Subscriber_No = "ADD".Subscriber_No;
```

There is always only one Left table (the first table)

All tables after the first table are each Right Tables.

The Spool from each join result remains the Left Table

The first table is always the left table and the rest are the right tables. It is the spool or intermediate results from each join that remain the left table. In this case all rows will return from the Claims table.

INNER JOIN with Additional AND Clause

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	Loraine	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

```
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

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	Loraine	40200.00
2000000	?	Jones	Squiggy	32800.50

Department_Table

Dept_No	Department_Name
100	Marketing
200	Research and Dev
300	Sales
400	Customer Support
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

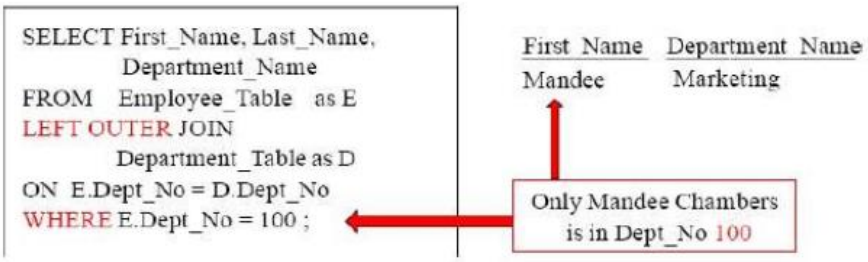
Employee_Table					Department_Table	
Employee_No	Dept_No	Last_Name	First_Name	Salary	Dept_No	Department_Name
1232578	100	Chambers	Mandee	48850.00	100	Marketing
1256349	400	Harrison	Herbert	54500.00	200	Research and Dev
2341218	400	Reilly	William	36000.00	300	Sales
2312225	300	Larkins	Loraine	40200.00	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.

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	Loraine	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		



The additional WHERE is always 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

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	Loraine	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

```
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.

Results from OUTER JOIN with Additional AND Clause

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	Loraine	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

OUTER Join with additional AND

```
SELECT First_Name, Department_Name
FROM   Employee Table   as E
```

First_Name	Department_Name
Mandee	Marketing
Herbert	?
William	?
Loraine	?

```
LEFT OUTER JOIN
      Department_Table as D
ON E.Dept_No = D.Dept_No
AND E.Dept_No = 100 ;
```

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

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	Loraine	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

```
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

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	Loraine	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

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.

Result Set of 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;
```

First_Name	Last_Name	Department_Name
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! Three Department rows had an 'm' in their name. 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	Loraine	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. Nine 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	Loraine	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 ON Clause in the join!

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

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	Loraine	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	Loraine	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*/
```

```
/* Query 2*/
```

```
SELECT First_Name,
  Department_Name
FROM Employee_Table
   INNER JOIN
   Department_Table ;
```

```
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

Customer_Number	Customer_Name
11111111	Billy's Best Choice
31313131	Acme Products
31323134	ACE Consulting
57896883	XYZ Plumbing
87323456	Databases N-U

Order_Table

Order_Number	Customer_Number	Order_Total
123456	11111111	12347.53
123512	11111111	8005.91
123552	31323134	5111.47
123585	87323456	15231.62
123777	57896883	23454.84

A Cross Join is the ANSI equivalent to a Product Join

Only a WHERE will work. ON Will NOT!

```
SELECT Customer_Name,
       Order_Number
FROM   Customer_Table
CROSS JOIN
       Order_Table
WHERE  Order_Number = 123456
ORDER BY 1;
```

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

Customer_Number	Customer_Name
11111111	Billy's Best Choice
31313131	Acme Products
31323134	ACE Consulting
57896883	XYZ Plumbing
87323456	Databases N-U

Order_Table

Order_Number	Customer_Number	Order_Total
123456	11111111	12347.53
123512	11111111	8005.91
123552	31323134	5111.47
123585	87323456	15231.62
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 quite often just isn't worth anything!

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	Loraine	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	Loraine	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

Customer_Number	Customer_Name
11111111	Billy's Best Choice
31313131	Acme Products

Order_Table

Order_Number	Customer_Number	Order_Total
123456	11111111	12347.53
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
INNERJOIN 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
INNERJOIN 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

Customer_Number	Customer_Name
11111111	Billy's Best Choice
31313131	Acme Products
31323134	ACE Consulting
57896883	XYZ Plumbing
87323456	Databases N-U

Order_Table

Order_Number	Customer_Number	Order_Total
123456	11111111	12347.53
123512	11111111	8005.91
123552	31323134	5111.47
123585	87323456	15231.62
123777	57896883	23454.84

```
SELECT Customer_Name,
       Order_Number,
       Order_Total
FROM Customer_Table as Cust
INNERJOIN 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
INNERJOIN 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! They are both inner joins, so they bring back the same results. Had they been outer joins, the WHERE and the AND would act differently.

Quiz – Will both queries bring back the same Answer Set?

Customer_Table

Customer_Number	Customer_Name
11111111	Billy's Best Choice
31313131	Acme Products
31323134	ACE Consulting
57896883	XYZ Plumbing
87323456	Databases N-U

Order_Table

Order_Number	Customer_Number	Order_Total
123456	11111111	12347.53
123512	11111111	8005.91
123552	31323134	5111.47
123585	87323456	15231.62
123777	57896883	23454.84

```
SELECT Customer_Name,
       Order_Number,
       Order_Total
FROM Customer_Table as Cust
LEFTOUTER 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

Customer_Number	Customer_Name
11111111	Billy's Best Choice
31313131	Acme Products
31323134	ACE Consulting
57896883	XYZ Plumbing
87323456	Databases N-U

Order_Table

Order_Number	Customer_Number	Order_Total
123456	11111111	12347.53
123512	11111111	8005.91
123552	31323134	5111.47
123585	87323456	15231.62
123777	57896883	23454.84

```
SELECT Customer_Name,
       Order_Number,
       Order_Total
FROM Customer_Table as Cust
LEFTOUTER 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 OUTERJOIN
     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

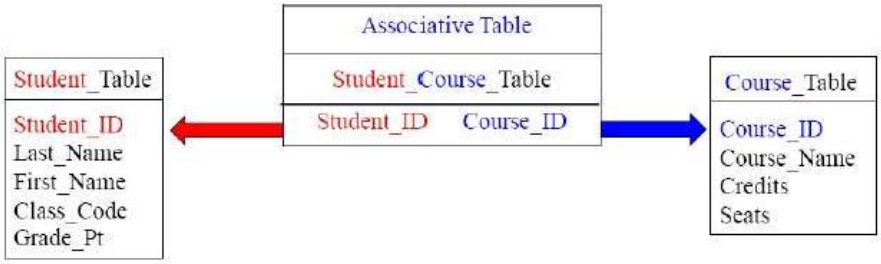
Course_ID	Course_Name	Credits	Seats
100	Database Concepts	3	50
200	Introduction to SQL	3	20
210	Advanced SQL	3	22
220	V2R3 SQL Features	2	25
300	Physical Database Design	4	20
400	Database Administration	4	16

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, and its sole purpose is to join these two tables together.

Quiz – Can you Write the 3-Table Join to ANSI Syntax?


```
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!

Please re-write the above query using ANSI Syntax.

Answer – Can you Write the 3-Table Join to ANSI Syntax?

Traditional Syntax

```
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 ;
```

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;
```

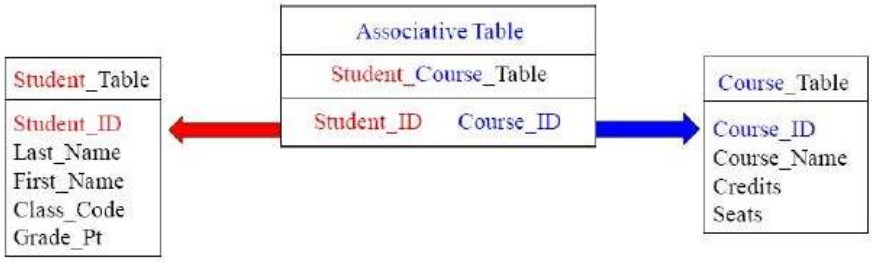
The above query has been written using ANSI Syntax.

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?

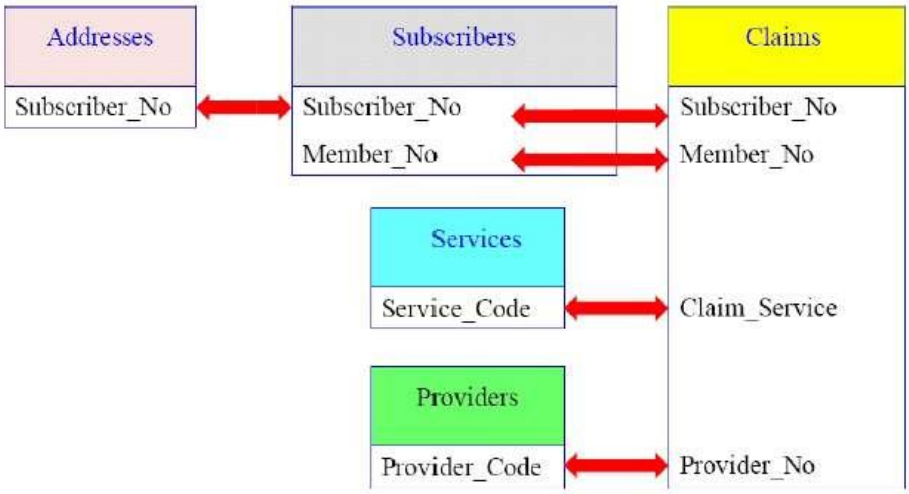



```
Select S.*, C.*
From Student_Table as S
   INNER JOIN
      Student_Course_Table as SC
   INNER JOIN
      Course_Table as C
ON          C.Course_ID = SC.Course_ID
ON          SC.Student_ID = S.Student_ID;
```

The trick is to put the first ON clause for the last join and go backwards

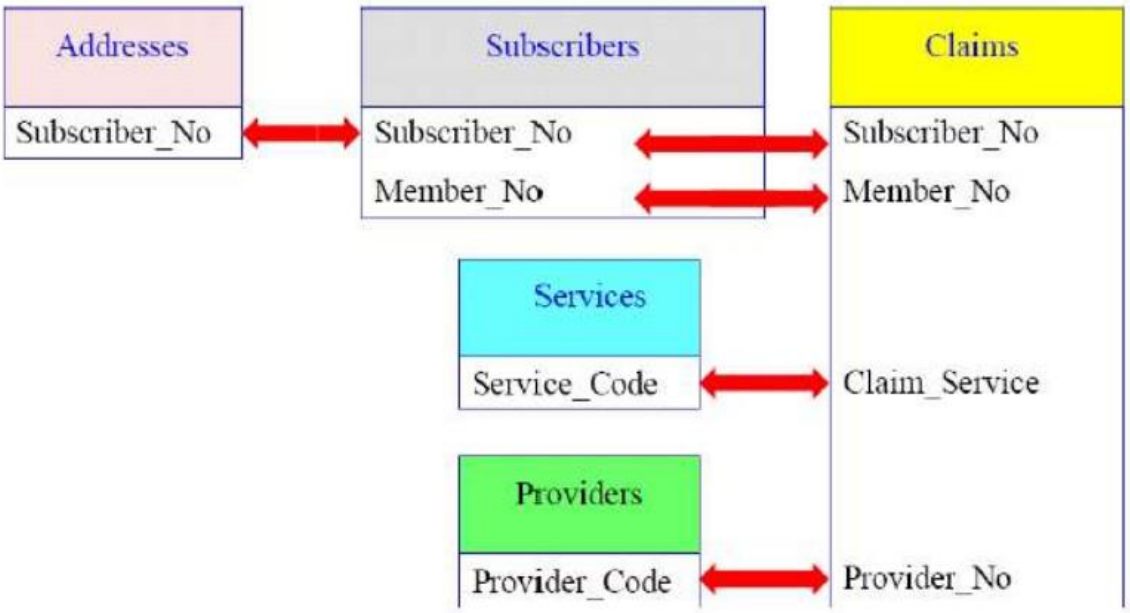
This is tricky. The only way it works is to place the ON clauses backwards. The first ON Clause represents the last INNER JOIN, and then moves backwards.

The 5-Table Join – Logical Insurance Model



Above is the logical model for the insurance tables showing the Primary Key and Foreign Key relationships (PK/FK).

Quiz - Write a Five Table Join Using ANSI Syntax



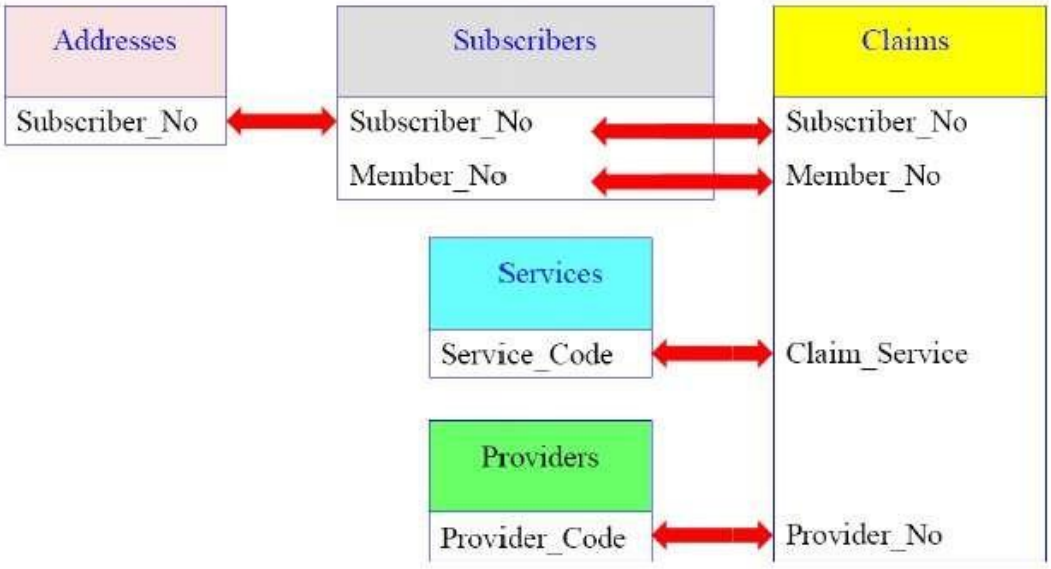
Your mission is to write a five table join selecting all columns using ANSI syntax.

Answer - Write a Five Table Join Using ANSI Syntax

```
SELECT
   cla1.*, sub1.*, add1.* ,pro1.*, ser1.*
FROM       CLAIMS AS cla1
INNER JOIN
           SUBSCRIBERS AS sub1
ON          cla1.Subscriber_No = sub1.Subscriber_No
AND          cla1.Member_No = sub1.Member_No
INNER JOIN
           ADDRESSES AS add1
ON          sub1.Subscriber_No = add1.Subscriber_No
INNER JOIN
           PROVIDERS AS pro1
ON          cla1.Provider_No = pro1.Provider_Code
INNER JOIN
           SERVICES AS ser1

ON        cla1.Claim_Service = ser1.Service_Code ;
```

Above is the example writing this five table join using ANSI syntax.

Quiz - Write a Five Table Join Using ANSI Syntax



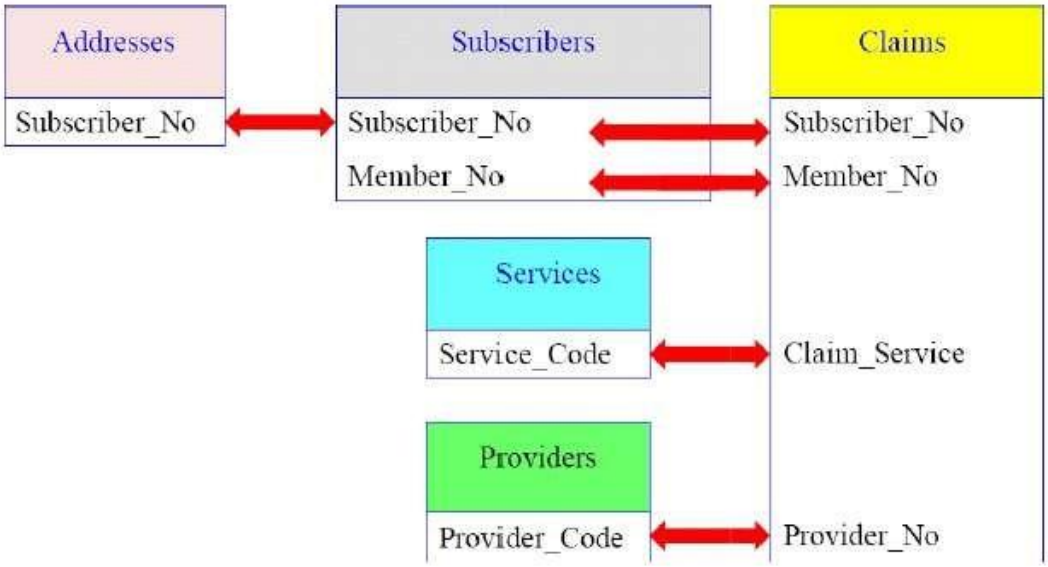
Your mission is to write a five table join selecting all columns using ANSI syntax.

Answer - Write a Five Table Join Using ANSI Syntax

```
SELECT
   cla1.*, sub1.*, add1.* ,pro1.*, ser1.*
FROM       CLAIMS AS cla1
INNER JOIN
           SUBSCRIBERS AS sub1
ON          cla1.Subscriber_No = sub1.Subscriber_No
AND          cla1.Member_No = sub1.Member_No
INNER JOIN
           ADDRESSES AS add1
ON          sub1.Subscriber_No = add1.Subscriber_No
INNER JOIN
           PROVIDERS AS pro1
ON          cla1.Provider_No = pro1.Provider_Code
INNER JOIN
           SERVICES AS ser1
ON          cla1.Claim_Service = ser1.Service_Code ;
```

Above is the example writing this five table join using ANSI syntax.

Quiz - Write a Five Table Join Using Non-ANSI Syntax



Your mission is to write a five table join selecting all columns using Non-ANSI syntax.

Answer - Write a Five Table Join Using Non-ANSI Syntax

```
SELECT   cla1.*, sub1.*, add1.* ,pro1.*, ser1.*
FROM     CLAIMS AS cla1,
         SUBSCRIBERS AS sub1,
         ADDRESSES AS add1,
         PROVIDERS AS pro1,
         SERVICES AS ser1

WHERE    cla1.Subscriber_No = sub1.Subscriber_No
AND      cla1.Member_No = sub1.Member_No
AND      sub1.Subscriber_No = add1.Subscriber_No
AND      cla1.Provider_No = pro1.Provider_Code
AND      cla1.Claim_Service = ser1.Service_Code ;
```

Above is an example of writing this five table join using Non-ANSI syntax.

Quiz –Re-Write this putting the ON clauses at the END

```
SELECT
   cla1.*, sub1.*, add1.* ,pro1.*, ser1.*
FROM       CLAIMS AS cla1
INNER JOIN
           SUBSCRIBERS AS sub1
ON          cla1.Subscriber_No = sub1.Subscriber_No
AND          cla1.Member_No = sub1.Member_No
INNER JOIN
           ADDRESSES AS add1
ON          sub1.Subscriber_No = add1.Subscriber_No
INNER JOIN
           PROVIDERS AS pro1
ON          cla1.Provider_No = pro1.Provider_Code
INNER JOIN
           SERVICES AS ser1
ON          cla1.Claim_Service = ser1.Service_Code ;
```

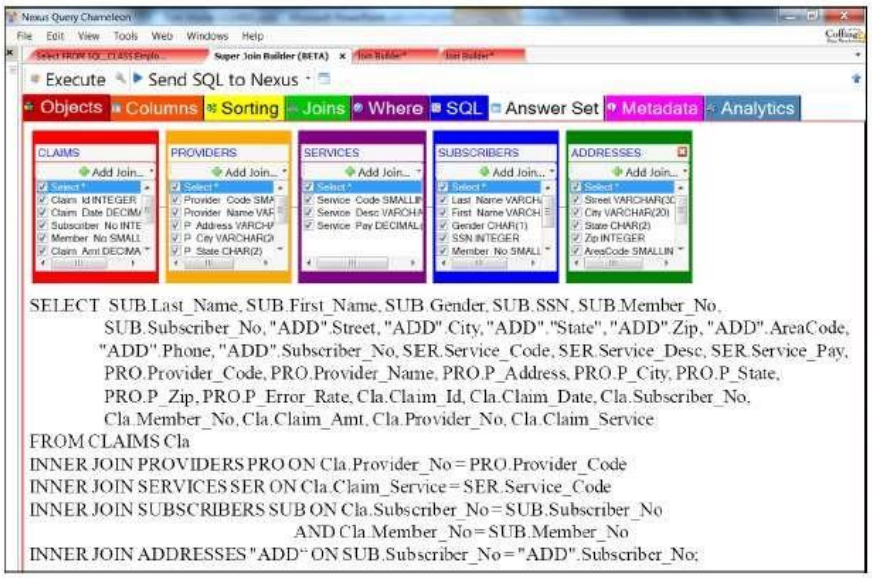
Above is an example of writing this five table join using Non-ANSI syntax.

Answer –Re-Write this putting the ON clauses at the END

```
SELECT     cla1.*, sub1.*, add1.* ,pro1.*, ser1.*
FROM       PROVIDERS AS pro1
INNER JOIN
           ADDRESSES AS add1

INNER JOIN
          SUBSCRIBERS AS sub1
INNER JOIN
          SERVICES AS ser1
INNER JOIN
          CLAIMS as cla1
ON         cla1.Claim_Service = ser1.Service_Code
ON            cla1.Subscriber_No = sub1.Subscriber_No
AND           cla1.Member_No = sub1.Member_No
ON               sub1.Subscriber_No =add1.Subscriber_No
ON                  cla1.Provider_No = pro1.Provider_Code ;
```

Above is an example of writing this five table join using ANSI syntax with the ON clauses at the end. Also to make this happen, we had to move the tables around. Notice that the first ON clause represents the last two tables being joined and then it works backwards.

The Nexus Query Chameleon Writes the SQL for Users.



Let Nexus show users the table relationships and then let Nexus build the SQL. Just load the ERwin logical model inside Nexus, and then all users can point and click.

Join Indexes

Join Indexes

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

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	Loraine	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	Jon	48000.00

Department_Table

Dept_No	Department_Name
100	Marketing
200	Research and Dev
300	Sales
400	Customer Support
500	Human Resources

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	Loraine	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

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	Loraine	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	Jon	48000.00

Department_Table

Dept_No	Department_Name
100	Marketing
200	Research and Dev
300	Sales
400	Customer Support
500	Human Resources

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	Loraine	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 Dve

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

Customer_Number	Customer_Name
11111111	Billy's Best Choice
31313131	Acme Products
31323134	ACE Consulting
57896883	XYZ Plumbing
87323456	Databases N-U

Order_Table

Order_Number	Customer_Number	Order_Total
123456	11111111	12347.53
123512	11111111	8005.91
123552	31323134	5111.47
123585	87323456	15231.62
123777	57896883	23454.84

Not Repeated

Join Index named CUST_ORDER_IDX

Customer_Number	Customer_Name	Order_Number	Order_Date	Order_Total
1111111	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.

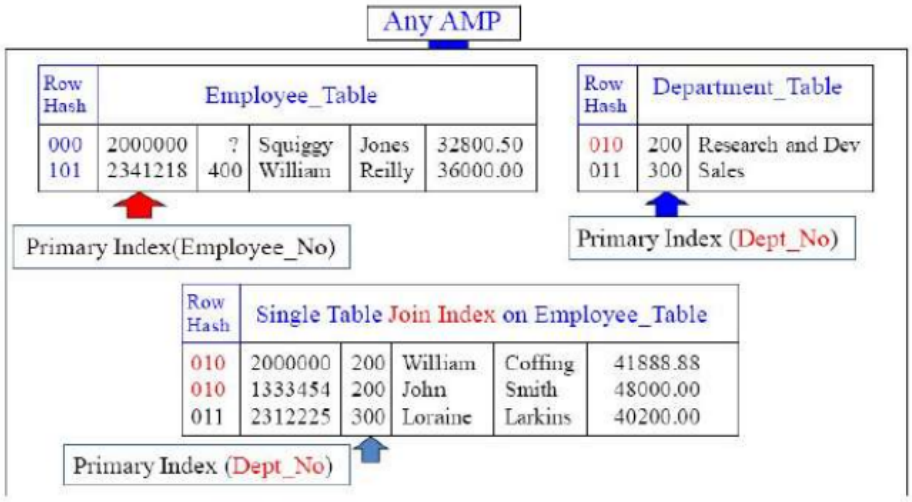
Creating a Single-Table Join Index

```
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.

Single Table Join Index Great For LIKE Clause

Build a STJI with column that contains three columns. They are the LIKE column being queried, the primary index column of the base table and the keyword ROWID!

```
CREATE JOIN INDEX Good_For_Like AS
SELECT License_Plate, PI_Col, ROWID/* Global Join Index */
FROM BMV_Table
Primary Index (PI_Col) ;
```

```
SELECT *
FROM BMV_Table
WHERE License_Plate LIKE'TeraT%' ;
```

The PE will choose to scan the narrow table (the Join Index) and qualify all rows that qualify with a car license LIKE 'TeraT%', then the PE uses the ROWID to get data from the BMV_Table where row is on the same AMP because both the base table and join index are on the same AMP because they both have the same Primary Index. This can save enormous time for queries using the LIKE command.

A LIKE command on a base table will never use a Non-Unique Secondary Index (NUSI). The above technique should be tested and only used if a lot of users are utilizing the LIKE command on a large table. If that is the case a lot of time can be saved.

Single Table Join Index with Value Ordered NUSI

A Value Ordered NUSI can only be done on columns that are 4-byte integers. Dates qualify because they are stored internally in Teradata as 4-byte integers.

```
CREATE JOIN INDEX OrdersJI AS
Select * from Order_Table
Primary Index(Customer_Number) ;

Create Index (Order_Date) order by values
on OrdersJI;
```

A value ordered index has been expanded from 16 to 64 columns.

Indexes are always sorted by their hash, but a Value Ordered index is sorted on each AMP by values and not hash. This is a great technique for you to run trials on.

Aggregate Join Indexes

Aggregate Join Indexes may be defined on:

Single Tables - A columnar subset of a base table with aggregates automatically maintained by Teradata.

Multiple Tables - A columnar subset of as many as 64 base tables with aggregate columns automatically maintained by Teradata.

Sparse Join Indexes are defined with a WHERE clause that limits the number of base table rows included and the space required to store them.

Aggregate Join Indexes can only include SUM and COUNT values. You can calculate Averages from these two columns though.

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!

New Aggregate Join Index (Teradata V14.10)

```
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
   ,MAX (Order_Total) as Max_Ord
   ,MIN(Order_Total) as Min_Ord
 FROM   Order_Table
 Group by 1, 2, 3;
```

Only Sum and Count can be used with an Aggregate Join Index (pre Teradata V14.10).

Now, Min and Max can be used with an Aggregate Join Index (Teradata V14.10).

Aggregate Join Index (AJI) has always supported (SUM and COUNT), but with Teradata V14.10, there is support for MIN/MAX aggregates also.

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.