# Indexes in Teradata

Teradata as we all know is a Relational Database Management System (RDBMS) for the world's largest commercial databases. It is the market leader in Data Warehousing. Its architecture is so designed that it takes advantage of the concept of parallelism.

Almost every RDBMS use KEY concept i.e. Primary Key, Foreign Key etc. while implementing the DDL of a table. But in **Teradata we don't have any concept of Keys instead what we have is INDEX.** Indexes are conceptually different from keys. PRIMARY KEY is relational modeling term that defines in the logical model whereas the PRIMARY INDEX is a physical database implementation term that defines the actual columns used to distribute and access rows in a table.

**What is an Index?**

In the Teradata RDBMS, an index is used to define row uniqueness and retrieve data rows; it also can be used to enforce the primary key and unique constraint for a table. The Teradata RDBMS support five types of indexes:

1. Unique Primary Index (UPI)
2. Unique Secondary Index (USI)
3. Non-Unique Primary Index (NUPI)
4. Non-Unique Secondary Index (NUSI)
5. Join Index

A typical index contains two fields:

1. A Value
2. A Pointer to instances of the value in a data table

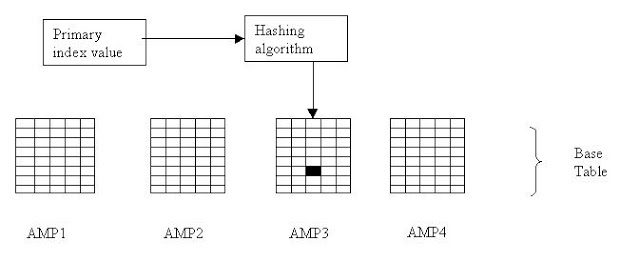
Teradata uses the index and hashing to distribute rows across the AMPs, the value is condensed into an entity called a row hash.

The following rules apply to the indexes:

* As mentioned above index help in distribution and retrieval of rows of a table. It can be made up of one or more columns.
* A table can have a number of indexes, including 1 Primary Index and up to 32 Secondary Index.
* An index defined for a table can be primary or secondary, and can be unique or non-unique. Each type of index affects the performance of the system and also the data integrity.
* An index is generally defined on columns of a table which are generally used in WHERE or join condition.
* Index helps in enforcing PRIMARY KEY and UNIQUE constraint.

**Primary Index**

Primary Index determines the distribution of the rows on the disks controlled AMPs. **A primary index in Teradata RDBMS is required for row distribution and storage.** When a row is inserted, its hash code is calculated using a hashing algorithm and depending on the whether the index is UNIQUE or Non-UNIQUE it is checked for duplication of those index. Rows having the same hash value are stored on the same AMP.

****

**Primary Index Rules**

Rule 1: One Primary Index per table.

Rule 2: A Primary Index value can be unique or non-unique.

Rule 3: The Primary Index value can be NULL.

Rule 4: The Primary Index value can be modified.

Rule 5: The Primary Index of a populated table cannot be modified.

Rule 6: A Primary Index has a limit of 64 columns.

**Two Types of Primary Indexes (UPI or NUPI)**

**Unique Primary Index (UPI)**

A Unique Primary Index (UPI) is unique and cannot have any duplicates.

If you try and insert a row with a Primary Index value that is already in the table, the row will be rejected. An UPI enforces UNIQUENESS for a column.

A Unique Primary Index (UPI) will always spread the rows of the table evenly amongst the AMPs. UPI access is always a one-AMP operation.

**Non-Unique Primary Index (NUPI)**

A Non-Unique Primary Index (NUPI) means that the values for the selected column can be non-unique. Duplicate values can exist.

A Non-Unique Primary Index will almost never spread the table rows evenly amongst the AMPs.

All-AMP operation will take longer if the data is unevenly distributed. You might pick a NUPI over an UPI because the NUPI column may be more effective for query access and joins.



**Multi-Column Primary Indexes:**

Teradata allows more than one column to be designated as the Primary Index. It is still only one Primary Index, but it is merely made up by combining multiple columns together. Teradata allows up to 64 combined columns to make up the one Primary Index required for a table.

This is often done for two reasons:

1. To get better data distribution among the AMPs
2. Users often use multiple keys in query

**Hashing Process to find out a row from a table**

1. The primary index value goes into the hashing algorithm.
2. The output of the hashing algorithm is the row hash value.
3. The hash map points to the specific AMP where the row resides.
4. The PE sends the request directly to the identified AMP.
5. The AMP locates the row(s) on its vdisk.
6. The data is sent over the BYNET to the PE, and the PE sends the answer set on to the client application.

**Duplicate Row Hash Values**

It is possible for the hashing algorithm to end up with the same row hash value for two different rows.

There are two ways this could happen:

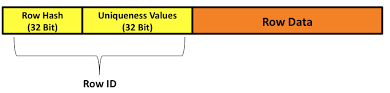
**Duplicate NUPI values:** If a Non-Unique Primary Index is used, duplicate NUPI values will produce the same row hash value.

**Hash synonym:** Also called a hash collision, this occurs when the hashing algorithm calculates an identical row hash value for two different Primary Index values.

To differentiate each row in a table, every row is assigned a **unique Row ID**. The Row ID is the combination of the row **hash value**and **uniqueness value.**

Row ID = Row Hash Value + Uniqueness Value

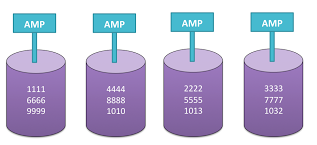
The uniqueness value is used to differentiate between rows whose Primary Index values generate identical row hash values. In most cases, only the row hash value portion of the Row ID is needed to locate the row.

****

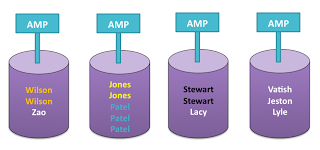
When each row is inserted, the AMP adds the row ID, stored as a prefix of the row.  
The first row inserted with a particular row hash value is assigned a uniqueness value of 1. The uniqueness value is incremented by 1 for any additional rows inserted with the same row hash value.

Below Diagrams will show how data resides in AMP.

**Even Distribution with an UPI**

****

**Uneven Distribution with a NUPI**

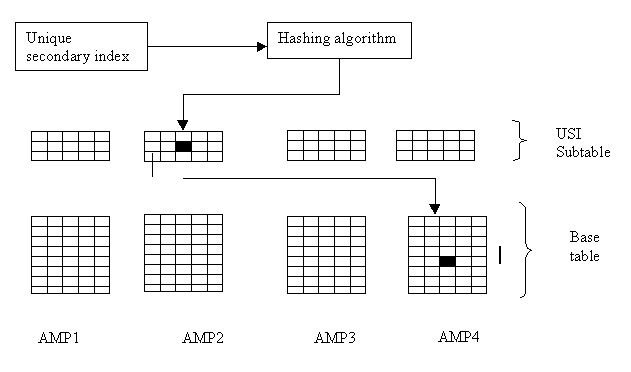
****

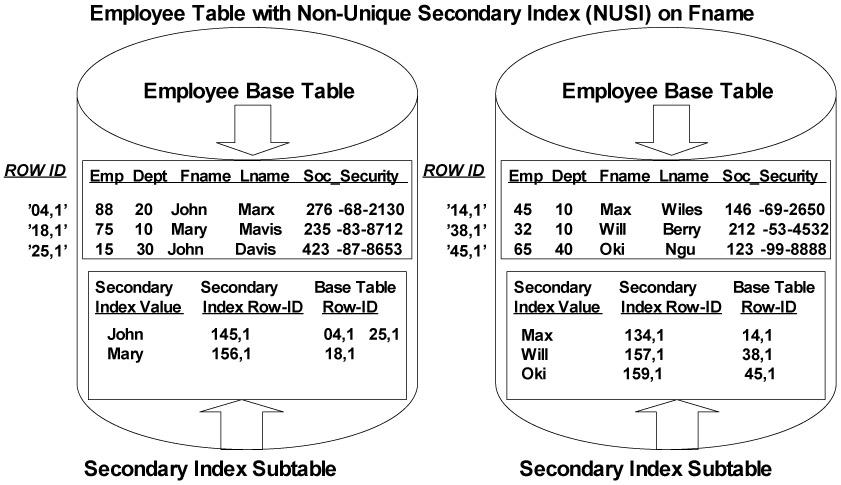
**Secondary Index**

In addition to a primary index, up to 32 unique and non-unique secondary indexes can be defined to a table. Comparing to primary indexes, Secondary indexes allow an alternate path to access the rows in the table which is less frequently used. Basically a Secondary Index is a subtable that is stored in all AMPs separately from the primary table. The subtable made and maintained by the system contains the following information:

* RowIDs of the subtable rows
* Base table index column values
* RowIDs of the base table rows

Below diagram shows how the secondary index subtable on each AMP is associated with the base table by the RowIDs:





Secondary indexes are mostly used to improve access on the queries that use a non PI column in search conditions.

Following are some common usages of SI:

* If a non PI column is being used in where clause often, define SI on it
* You can use USI to enforce uniqueness in a PPI table where partition columns are not part of PI

**Join Index**

A join index is nothing but an indexing structure which contains columns from multiple tables. Rather than having to join individual tables each time the join operation is needed, the query can be resolved via a join index and in most cases dramatically improves performance.

SYNTAX:

CREATE JOIN INDEX [<database-name>.]<index\_name>

[[NO] FALLBACK]

AS SELECT

[<columns>]

[SUM numeric-expression]

[COUNT coulmn-expression]

[EXTRACT year|month from date-expressino]

FROM [<database-name(s).]<table-names>]

[WHERE <search-condition>]

[GROUP BY <coulmn-name>]

[ORDER BY <coulmn-name>]

PRIMARY INDEX (<column-name>)

[Index<column-name> ORDER by HASH|VALUES]

;