DISTRIBUTE

ClustrixDB uses a hash to determine where a given row of data or a table's index (representation) should reside in a cluster. The columns selected for hashing are referred to as the "distribution key" for that representation. Every index requires guidance relative to which columns should comprise the distribution key.

By default, the distribution key uses the first column of an index, regardless of how many columns comprise the index. This is true for all indices including the primary key.

The DISTRIBUTE = clause may be used to override the single column default and define which columns of an index are to be hashed instead.

Using Default Distribution

In this example, the primary key of post_id was hashed and data for the table was distributed using that value and the default distribution setting of DISTRIBUTE = 1.

```
sql> CREATE TABLE user_posts (
     post_id int AUTO_INCREMENT,
     user_id int,
     posted_on timestamp,
     data blob,
     PRIMARY KEY (`post_id`) /*$ DISTRIBUTE=1 */,
     KEY `user_id_posted_on_idx` (`user_id`,`posted_on`) /*$ DISTRIBUTE=1 */
     );
```

In some cases, data and index distribution based on a single column can result in poor or "lumpy" distribution. To resolve this, we recommend putting the most unique (selective) column into the first column of composite index, or expanding the distribution key from a single column to multiple columns. There are two ways to do this.

1. Expand the distribution key to include more columns of the index

The example below shows the multi-column alternate key user_id_posted_on_idx distributed using both columns in the index instead of just the first.

```
sql> CREATE TABLE user_posts (
     post_id int AUTO_INCREMENT,
     user_id int,
     posted_on timestamp,
     data blob,
     PRIMARY KEY (`post_id`) /*$ DISTRIBUTE=1 */,
     KEY `user_id_posted_on_idx` (`user_id`,`posted_on`) /*$ DISTRIBUTE=2 */
     );
```

2. Expand the distribution key to also include column(s) of the primary key

The example below shows the alternate key user_id_posted_on_idx with DISTRIBUTE = 3. This means that the index will be distributed on both of its columns (user_id, posted_on) as well as the primary key (post_id). If the primary key is a compound key, you could further expand the distribution to include additional columns of the primary key.

```
sql> CREATE TABLE user_posts (
     post_id int AUTO_INCREMENT,
     user_id int,
     posted_on timestamp,
     data blob,
     PRIMARY KEY (`post_id`) /*$ DISTRIBUTE=1 */,
     KEY `user_id_posted_on_idx` (`user_id`,`posted_on`) /*$ DISTRIBUTE=3 */
     );
```

Modifying Distribution

Modifying Distribution - Primary Key

To modify the distribution of a composite primary key after a table has been created, follow this ALTER TABLE syntax:

```
ALTER TABLE tbl_name
PRIMARY KEY   [DISTRIBUTE = n
]
```

The distribution count for a primary key cannot exceed the number of columns in the primary key.

Modifying Distribution - Alternate Keys

To modify the distribution of other indexes (non- primary key) after a table has been created, specify the index_name in an ALTER TABLE per this syntax:

```
ALTER TABLE tbl_n
ame  [ ,INDEX ind
ex_name   [DISTRI
BUTE = n]]  [ ,
INDEX  index_name
   [DISTRIBUTE = n
 ]]
```

The maximum distribution value for non-primary keys is the combined number of columns in both the alternate and primary key.