Generated Columns

ClustrixDB supports Generated Columns, which are computed based on values from other columns. They are defined as part of the column's definition and can be:

- STORED, where values stored and updated as data is updated
- VIRTUAL, where values are computed as needed and do not take up any storage

When to use Generated Columns

You can use generated columns to simplify queries and improve performance. If you use the same complex expression in many queries, you could encapsulate that logic as a generated column. If that expression is commonly computed, you could make the generated column STORED and take advantage of not having to compute the value each time a row is accessed, though there will be additional storage overhead.

Generated columns can be used to simulate the behavior of a materialized view or a functional index (see caveats below).

Generated columns cannot reference:

- Subqueries
- Parameters
- Variables
- User-defined functions
- AUTO_INCREMENT columns, either as a base column or in a generated column definition

DDL for Generated Columns

To create a table with a generated column:

```
sql> CREATE TABLE EMPLOYEES (
 id INT AUTO_INCREMENT PRIMARY KEY,
 first_name VARCHAR(50) NOT NULL,
 middle_name VARCHAR(50) NOT NULL,
 last_name VARCHAR(50) NOT NULL,
 fullname varchar(200) GENERATED ALWAYS AS (CONCAT(first_name,' ',middle_name, ' ', last_name)),
 email VARCHAR(100) NOT NULL
);
```

If no keyword is specified, a generated column is VIRTUAL and the computed value is not stored. To create a generated column that is STORED, use the STORED keyword:

```
sql> ALTER TABLE EMPLOYEES ADD COLUMN firstlast_name VARCHAR(100) GENERATED ALWAYS AS (CONCAT(first_name,' ',
last_name)) STORED;
```

Only STORED generated columns can be part of a primary key.

Indexing Generated Columns

Indexes on generated columns are always stored but can be created on VIRTUAL or STORED columns.

```
sql> CREATE INDEX fullname_i on EMPLOYEES(fullname);
```

Querying Generated Columns

The examples below will use the following data:

```
sql> insert into EMPLOYEES (first_name, middle_name, last_name) values ('Jane','', 'Smith');
sql> insert into EMPLOYEES (first_name, middle_name, last_name) values ('Bob', 'Ashish', 'Gupta');
sql> insert into EMPLOYEES (first_name, middle_name, last_name) values ('John', 'Lee', 'Chang');
sql> insert into employees(first_name, middle_name, last_name) values ('Peter', 'Dinesh', 'Paul');
sql> insert into employees(first_name, middle_name, last_name) values ('Jan', 'Emilia', 'Clark');
sql> insert into employees(first_name, middle_name, last_name) values ('Harrison', '', 'Porter');
```

Clustrix is able to leverage indexes for generated columns that explicitly reference generated columns by name:

```
sql> explain select * from employees where fullname like 'smith%';
+-------------------------------------------------------------------------------------------------------------------
+-----------+-----------+
| Operation                                                                                                        |
Est. Cost | Est. Rows |
+-------------------------------------------------------------------------------------------------------------------
+-----------+-----------+
| stream_combine
|    212.50 |      0.51 |
|      filter (1.fullname >= param(3)) and (1.fullname < param(2)) and like(1.fullname, param(1), param(0))
|    211.45 |      0.51 |
|        index_scan 1 := EMPLOYEES.__idx_EMPLOYEES__PRIMARY
|    211.33 |      2.00 |
+-------------------------------------------------------------------------------------------------------------------
---+-----------+-----------+
3 rows in set (0.00 sec)
```

But not where the same expression used by the generated column is used in the query:

```
sql> explain select * from employees where (CONCAT(first_name,' ',middle_name, ' ', last_name)) like '%smith';
+-------------------------------------------------------------------------------------------------------------------
+-----------+-----------+
| Operation                                                                                                        |
Est. Cost | Est. Rows |
+-------------------------------------------------------------------------------------------------------------------
+-----------+-----------+
| stream_combine
| 213.32    |      2.70 |
|      filter like(concat(1.first_name, param(3), 1.middle_name, param(2), 1.last_name), param(1), param(0))
| 212.05    |      2.70 |
|        index_scan 1 := EMPLOYEES.__idx_EMPLOYEES__PRIMARY
| 211.99    |      3.00 |
+-------------------------------------------------------------------------------------------------------------------
+-----------+-----------+
3 rows in set (0.00 sec)
```

Caveats

ClustrixDB matches all MySQL functionality for generated columns except:

- ClustrixDB allows generated columns to be created on in-memory tables. MySQL does not.
- ClustrixDB does not allow FROM_UNIXSTAMP to be used in generated columns
- If a generated column uses a smaller integer type than the column it is based on, ClustrixDB will clip the values to fit. On MYSQL, the values will overflow.
- ClustrixDB permits non-deterministic functions in generated columns. MySQL does not.
- ClustrixDB does not permit inserting into generated columns with value DEFAULT.
- Error messages generated by ClustrixDB differ from those of MySQL.

In addition, ClustrxDB has the following caveats:

- Creating generated columns with excessively large expressions can lead to a group change.