LanguageManual LateralView

Lateral View Syntax

```
lateralView: LATERAL VIEW udtf(expression) tableAlias AS columnAlias (',' columnAlias)*
fromClause: FROM baseTable (lateralView)*
```

Description

Lateral view is used in conjunction with user-defined table generating functions such as `explode()`. As mentioned in Built-in Table-Generating Functions, a UDTF generates zero or more output rows for each input row. A lateral view first applies the UDTF to each row of base table and then joins resulting output rows to the input rows to form a virtual table having the supplied table alias.

> **Version**
>
> Prior to Hive 0.6.0, lateral view did not support the predicate push-down optimization. In Hive 0.5.0 and earlier, if you used a WHERE clause your query may not have compiled. A workaround was to add `set hive.optimize.ppd=false;` before your query. The fix was made in Hive 0.6.0; see https://issues.apache.org/jira/browse/HIVE-1056: Predicate push down does not work with UDTF's.

> **Version**
>
> From Hive 0.12.0, column aliases can be omitted. In this case, aliases are inherited from field names of StructObjectInspector which is returned from UTDF.

Example

Consider the following base table named `pageAds`. It has two columns: `pageid` (name of the page) and `adid_list` (an array of ads appearing on the page):

Column name	Column type
pageid	STRING
adid_list	Array<int>

An example table with two rows:

pageid	adid_list
front_page	[1, 2, 3]
contact_page	[3, 4, 5]

and the user would like to count the total number of times an ad appears across all pages.

A lateral view with explode() can be used to convert `adid_list` into separate rows using the query:

```
SELECT pageid, adid
FROM pageAds LATERAL VIEW explode(adid_list) adTable AS adid;
```

The resulting output will be

pageid (string)	adid (int)
"front_page"	1
"front_page"	2
"front_page"	3
"contact_page"	3
"contact_page"	4
"contact_page"	5

Then in order to count the number of times a particular ad appears, count/group by can be used:

```
SELECT adid, count(1)
FROM pageAds LATERAL VIEW explode(adid_list) adTable AS adid
GROUP BY adid;
```

int adid	count(1)
1	1
2	1
3	2
4	1
5	1

Multiple Lateral Views

A FROM clause can have multiple LATERAL VIEW clauses. Subsequent LATERAL VIEWS can reference columns from any of the tables appearing to the left of the LATERAL VIEW.

For example, the following could be a valid query:

```
SELECT * FROM exampleTable
LATERAL VIEW explode(col1) myTable1 AS myCol1
LATERAL VIEW explode(myCol1) myTable2 AS myCol2;
```

LATERAL VIEW clauses are applied in the order that they appear. For example with the following base table:

Array<int> col1	Array<string> col2
[1, 2]	[a", "b", "c"]
[3, 4]	[d", "e", "f"]

The query:

```
SELECT myCol1, col2 FROM baseTable
LATERAL VIEW explode(col1) myTable1 AS myCol1;
```

Will produce:

int mycol1	Array<string> col2
1	[a", "b", "c"]
2	[a", "b", "c"]
3	[d", "e", "f"]
4	[d", "e", "f"]

A query that adds an additional LATERAL VIEW:

```
SELECT myCol1, myCol2 FROM baseTable
LATERAL VIEW explode(col1) myTable1 AS myCol1
LATERAL VIEW explode(col2) myTable2 AS myCol2;
```

Will produce:

int myCol1	string myCol2
1	"a"
1	"b"
1	"c"
2	"a"
2	"b"
2	"c"
3	"d"
3	"e"
3	"f"
4	"d"
4	"e"
4	"f"

Outer Lateral Views

> **Version**
>
> Introduced in Hive version 0.12.0

The user can specify the optional `OUTER` keyword to generate rows even when a `LATERAL VIEW` usually would not generate a row. This happens when the UDTF used does not generate any rows which happens easily with `explode` when the column to explode is empty. In this case the source row would never appear in the results. `OUTER` can be used to prevent that and rows will be generated with `NULL` values in the columns coming from the UDTF.

For example, the following query returns an empty result:

```
SELEC * FROM src LATERAL VIEW explode(array()) C AS a limit 10;
```

But with the `OUTER` keyword

```
SELECT * FROM src LATERAL VIEW OUTER explode(array()) C AS a limit 10;
```

it will produce:

238 val_238 NULL
86 val_86 NULL
311 val_311 NULL
27 val_27 NULL
165 val_165 NULL
409 val_409 NULL
255 val_255 NULL
278 val_278 NULL
98 val_98 NULL
...