facebook

User-Defined Table-Generating Functions (UDTF)

By Paul Yang (pyang@facebook.com)

facebook

Outline

- **UDTF Description and Usage**
- Execution Phase
- Compile Phase

UDF vs UDAF vs UDTF

- User Defined Functions
  - One-to-one mapping
  - concat(“foo”, “bar”)
- User Defined Aggregate Functions
  - Many-to-one mapping
  - sum(num_ads)
- User Defined Table-generating Functions
  - One-to-many mapping
  - explode([1,2,3])

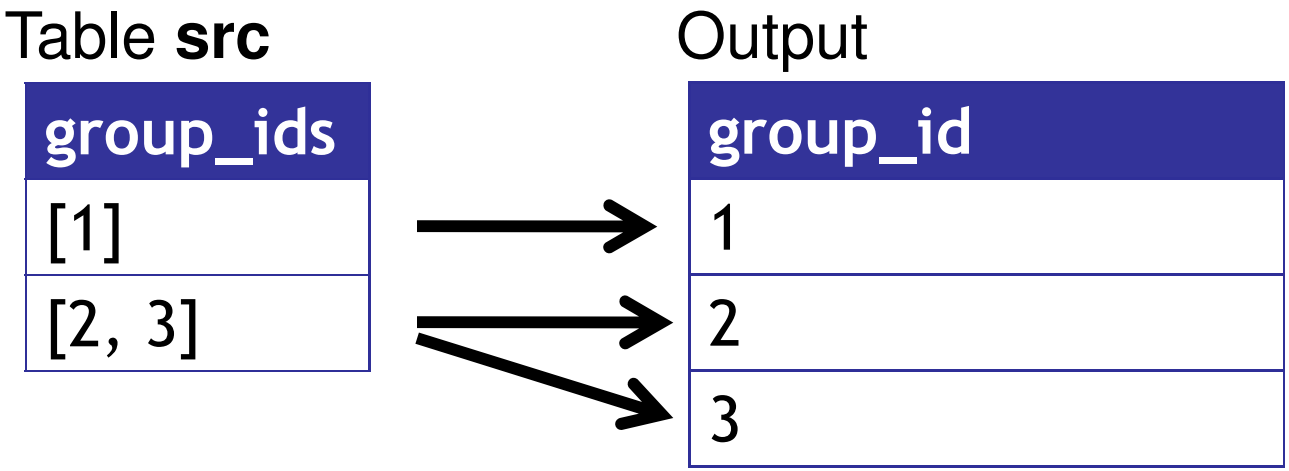
UDTF Example (Transform)

- explode(Array<?> arg)
  - Converts an array into multiple rows, with one element per row
- Transform-like syntax
  - **`SELECT udtf(col0, col1, …) AS colAlias FROM srcTable`**

facebook

UDTF Example (Transform)

- **`SELECT explode(group_ids) AS group_id FROM src`**



UDTF (Lateral View)

- Transform syntax limited to single expression
  - ~~`SELECT pageid, explode(adid_list)…`~~
- Use lateral view
  - Creates a virtual table using UDTF
- Lateral view syntax
  -
```
…FROM baseTable
LATERAL VIEW udtf(col0, col1…)
tableAlias AS colAlias0, colAlias1…
```

facebook

UDTF (Lateral View)

- Example Query
- **`SELECT src.*, myTable.* FROM src LATERAL VIEW explode(group_ids) myTable AS group_id`**

src

user_id	group_ids
100	[1]
101	[2,3]

facebook

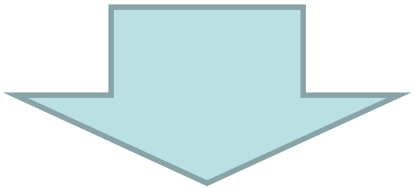
UDTF (Lateral View)

src

user_id	group_ids
100	[1]
101	[2,3]

explode(group_ids) **myTable** AS group_id

group_id
1
2
3



Join input rows to output rows

Result

user_id	group_ids	group_id
100	[1]	1
101	[2,3]	2
101	[2,3]	3

facebook

Outline

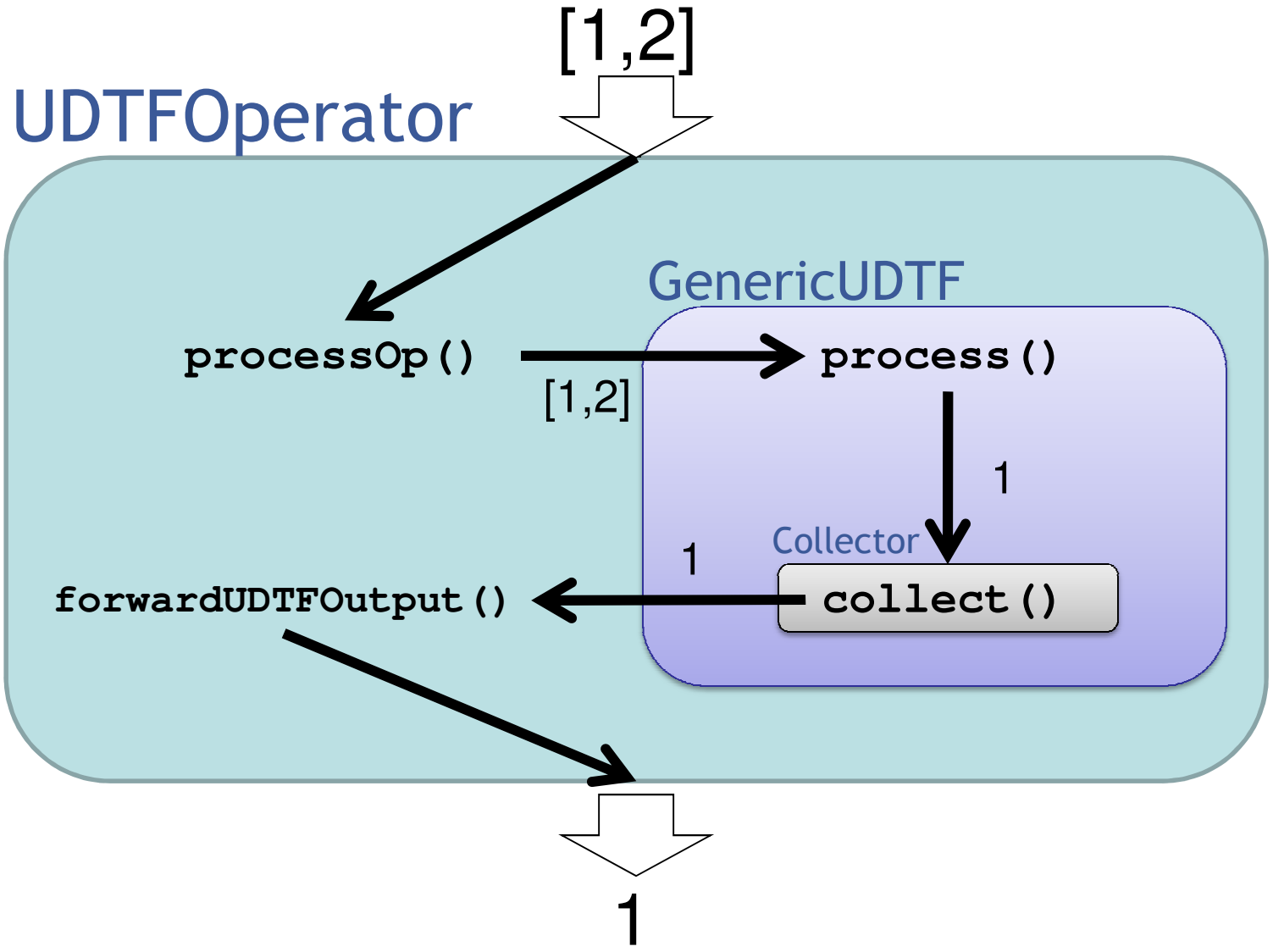
- UDTF Description and Usage
- **Execution Phase**
- Compile Phase

Execution Phase

- 2 New Operators
  - UDTFOperator
  - LateralViewJoinOperator
- Different Operator DAG's
  - For `SELECT udtf(…)`
  - For `… FROM udtf(…) … LATERAL VIEW`

facebook

Execution Phase - UDTFOperator



facebook

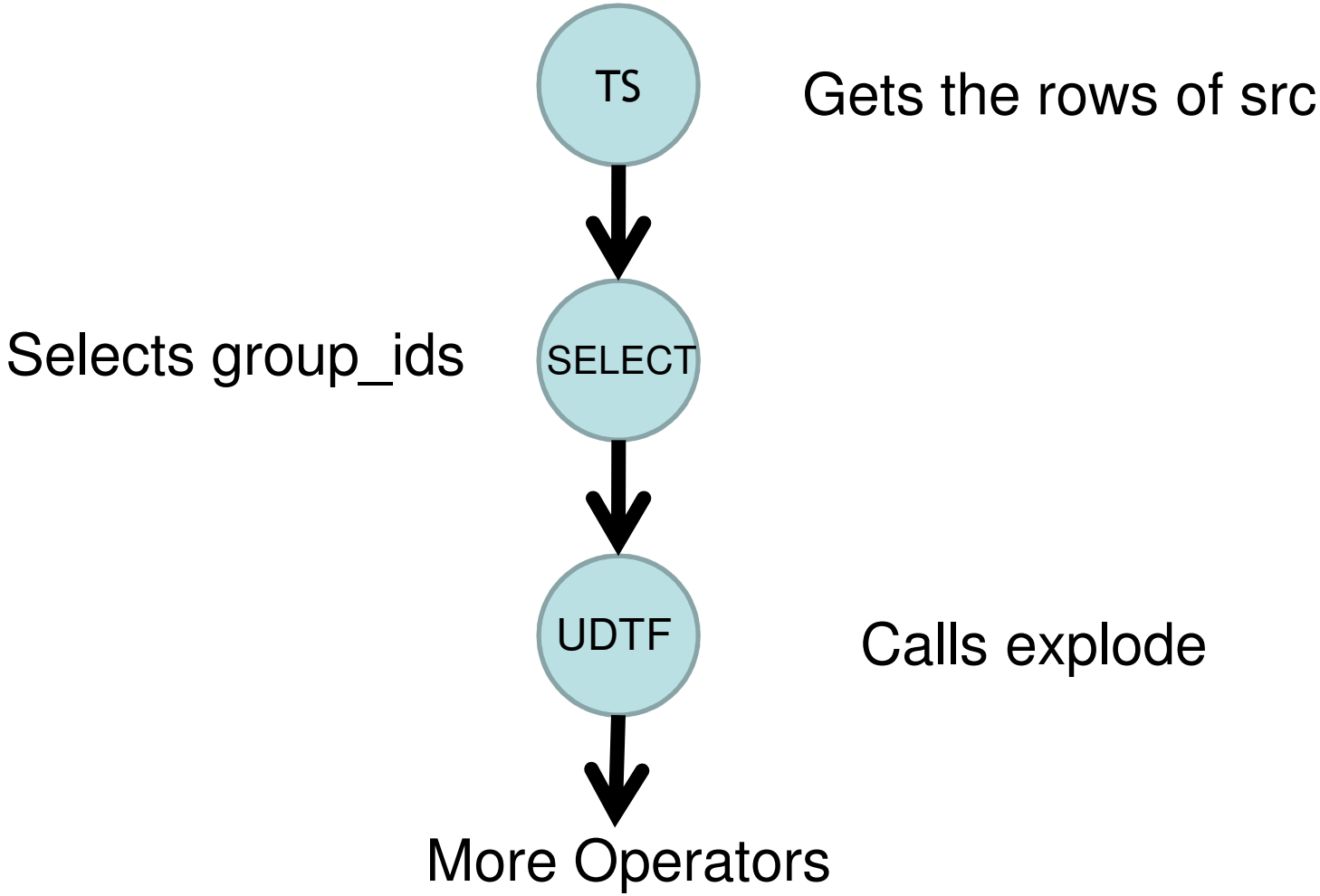
GenericUDTF Interface

```
public abstract class GenericUDTF {
  Collector collector = null;


  public abstract StructObjectInspector initialize(ObjectInspector [] argOIs)
  throws UDFArgumentException;


  public abstract void process(Object [] args) throws HiveException;


  public abstract void close() throws HiveException;


  public final void setCollector(Collector collector) {
    this.collector = collector;
  }

  protected final void forward(Object o) throws HiveException {
    collector.collect(o);
  }

}
```

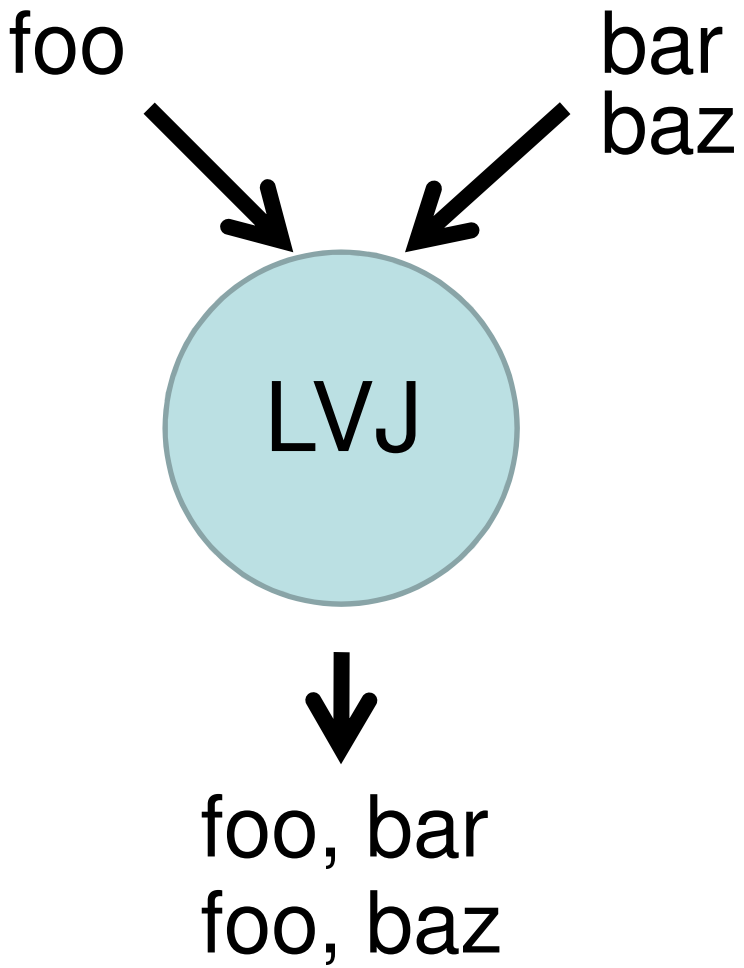
facebook

Execution Phase - Transform

```
SELECT explode(group_ids) AS group_id
FROM src
```

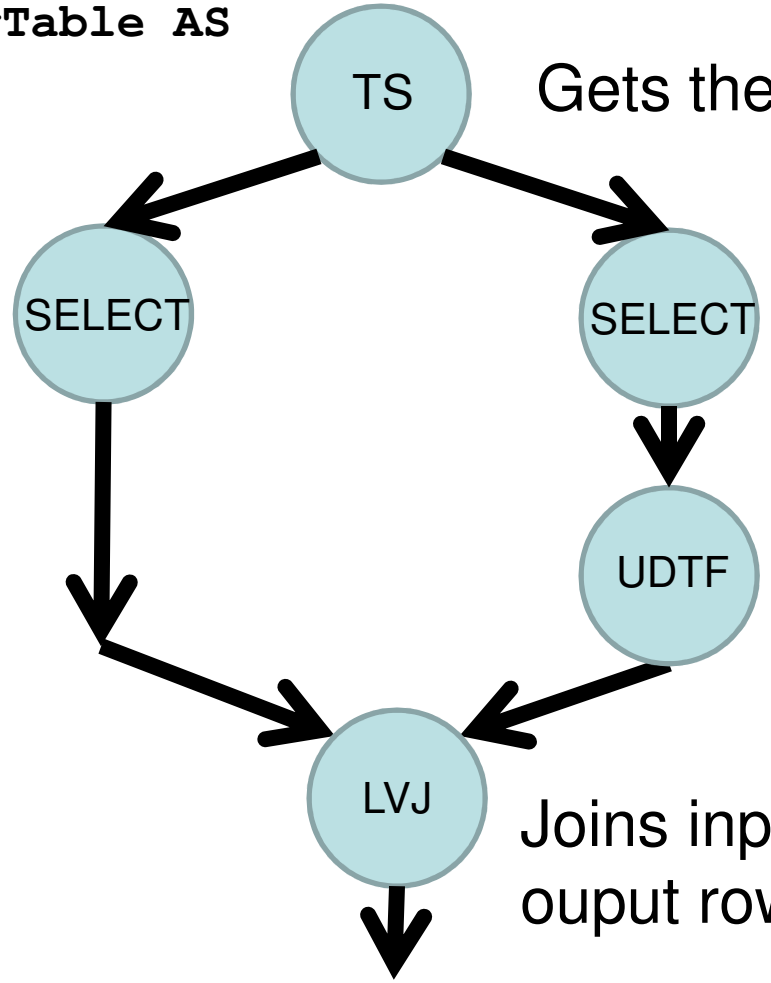


Execution Phase - LateralViewJoinOperator



Execution Phase (Lateral View)

```
FROM src LATERAL VIEW
explode(group_ids) myTable AS
group_id
```



Gets the rows of src

Selects other needed columns (user_id, group_ids)

Selects arg. cols for UDTF (group_ids)

Calls explode()

Joins input and ouput rows

More Operators

facebook

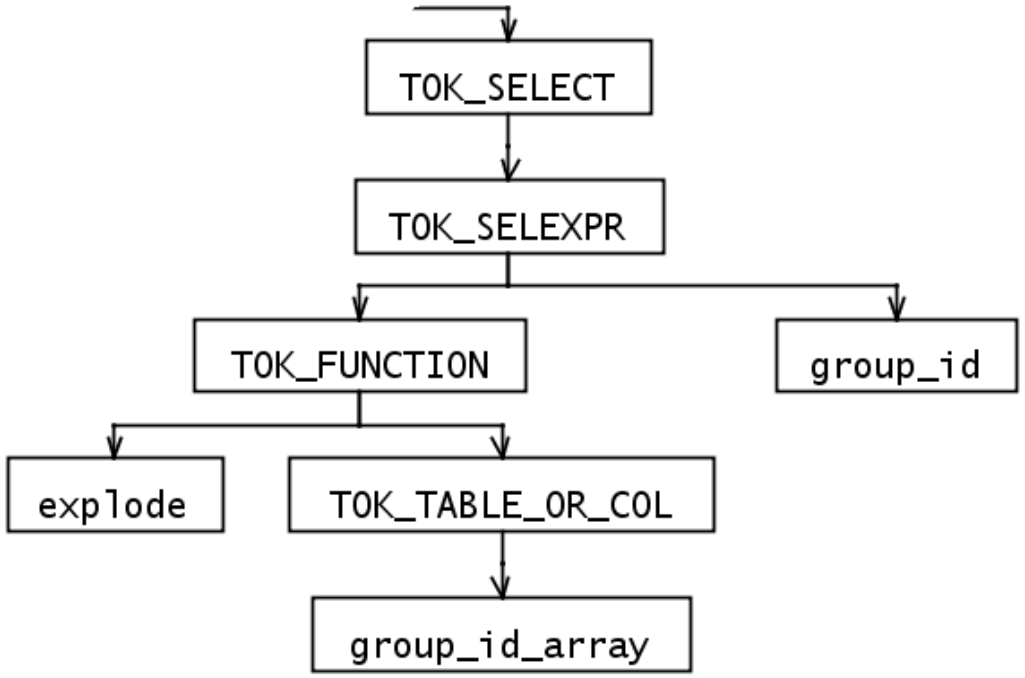
Outline

- UDTF Description and Usage
- Execution Phase
- **Compile Phase**

facebook

Compile Phase - Transform

- **`SELECT explode(group_id_array) AS group_id FROM src`**



facebook

Compile Time - Transform

- UDTF detected in genSelectPlan()

SemanticAnalyzer::genSelectPlan()

```
...

if (udtfExpr.getType() == HiveParser.TOK_FUNCTION) {
  String funcName =
    TypeCheckProcFactory.DefaultExprProcessor.getFunctionText(
        udtfExpr, true);
  FunctionInfo fi = FunctionRegistry.getFunctionInfo(funcName);
  if (fi != null) {
    genericUDTF = fi.getGenericUDTF();
  }
  isUDTF = (genericUDTF != null);
}

...
```

facebook

Compile Time - Transform

- If a UDTF is present
  - Generate a select operator to get the columns needed for UDTF
  - Attach UDTFOperator to SelectOperator

SemanticAnalyzer::genSelectPlan()

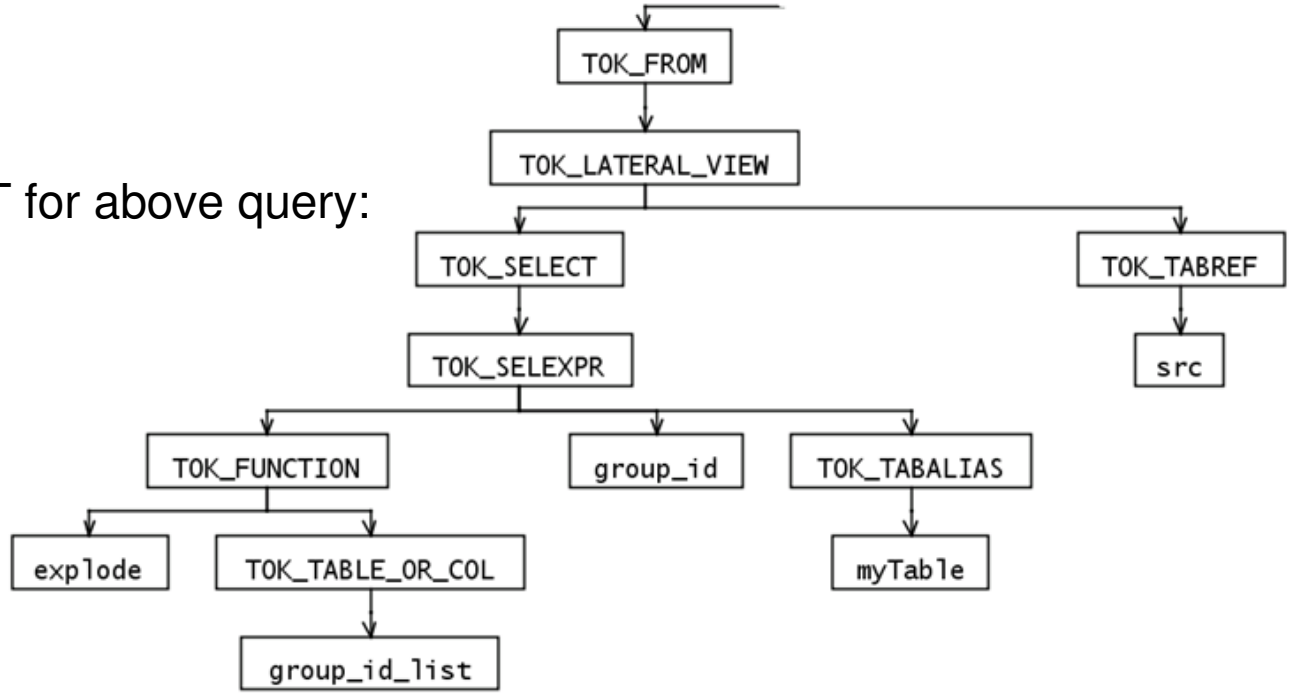
```
...
if (isInTransform) {
  output = genScriptPlan(trfm, qb, output);
}

if (isUDTF) {
  output = genUDTFPlan(genericUDTF,
udtfTableAlias, udtfColAliases, qb,
                       output);
}
```

facebook

Compile Phase - Lateral View

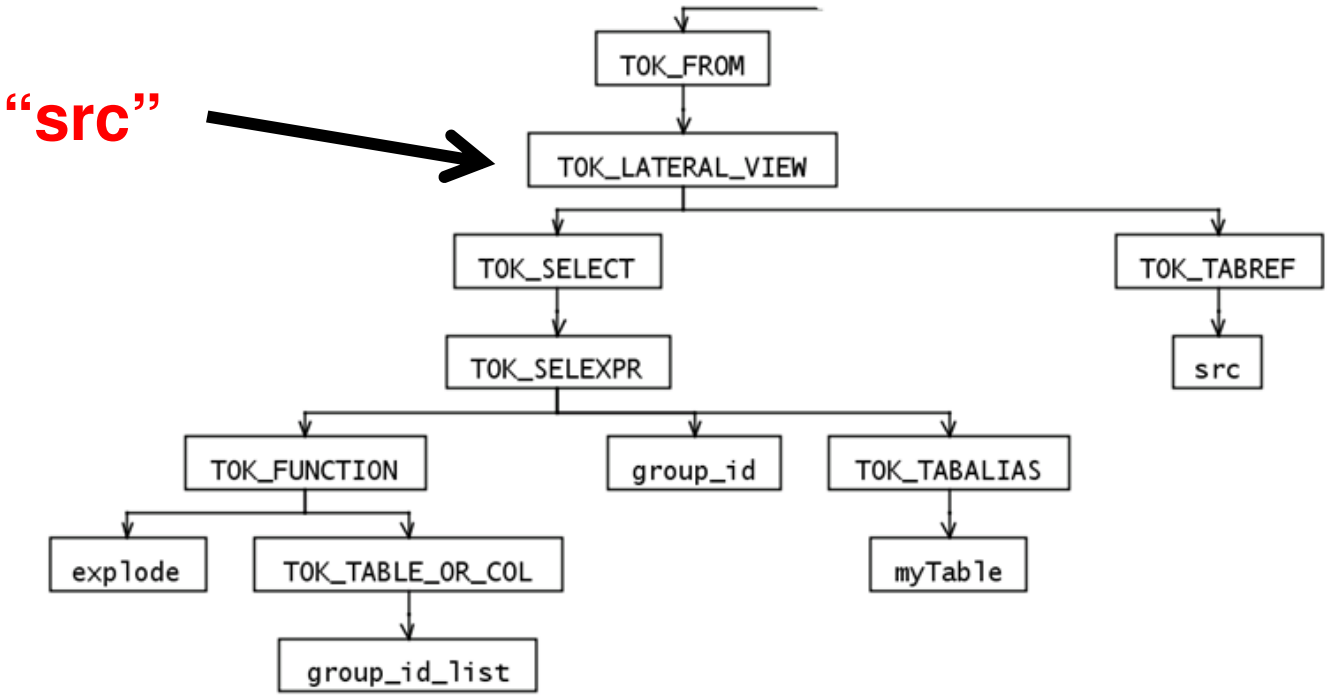
- **`SELECT * FROM src LATERAL VIEW explode(group_ids) myTable AS group_id`**

Partial AST for above query:



Compile Phase – Lateral View

- In SemanticAnalyzer::doPhase1()
  - make a mapping from the source table alias to TOK_LATERAL_VIEW



Compile Phase- Lateral View

- In SemanticAnalyzer:: genPlan()
  - Iterate through Map<String, Operator> aliasToOpInfo
  - Attach SELECT/UDTF Operator DAG with genLateralViewPlans()