目录

Data lineage model reference

Version 1.2 - 2021/7/7

https://www.sqlparser.com

lineage model summary

Objects in the output of a SQLFlow data lineage model. The output can be in XML or JSON format.

We try to make types defined in the SQLFlow data lineage model compatible with the Apache Atlas types.

So it will be easy to integrate the data lineage generated by the SQLFlow into the Apache Atlas.

1. table
2. view
3. resultset
4. relation
   - target element
   - source element
5. process
6. column
7. variable
   - scalar
   - cursor
   - record
8. procedure
   - argument
9. path
10. error

References

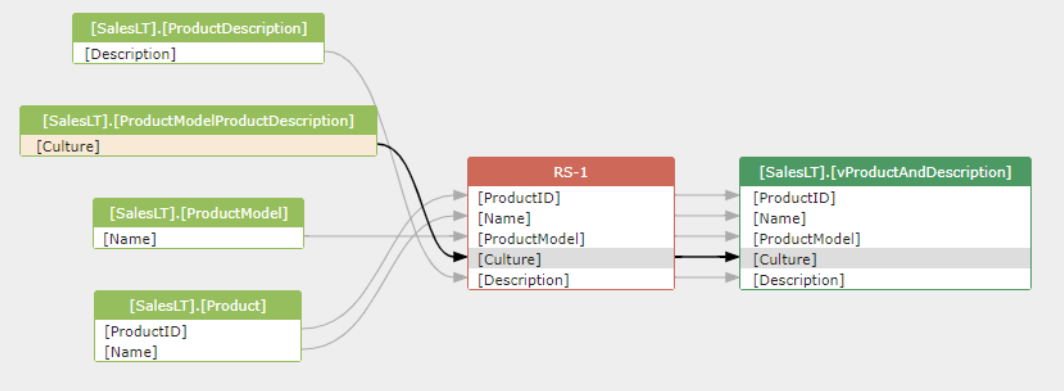
1. xml code used in this article is generated byDataFlowAnalyzer tools
2. digram used in this article is generated by theSQLFlow Cloud version

column-level lineage

SQLFlow generate column-level lineage by analyzing the SQL query and stored procedure.

The relationship built in the lineage model is based on the column, show the dataflow from one column to the other.

digram



xml output

```
<relation id="1" type="fdd" effectType="select">
 <target id="22" column="[ProductID]" parent_id="21" parent_name="RS-1" coordinate="[7,5,0],[7,18,0]"/>
 <source id="4" column="[ProductID]" parent_id="2" parent_name="[SalesLT].[Product]" coordinate="[7,5,0],[7,18,0]"/>
</relation>
```

simulate table-level lineage

Table to table transformations are also included in the column-level lineage model by using `PseudoRows` for 2 reasons:

1. This pseudo column lineage data will be used to generate the table-level lineage later if user need a table-level lineage model.
2. If a table is used in both column-level and table-level lineage, this pseudo column lineage will make it possible that use one table in a picture that appears in both the column-level and table-level lineage.

sample sql

```
create view v1 as select f1 from t2;
alter table t2 rename to t3;
```

column-level diagram

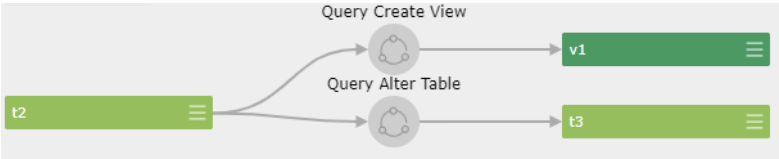
As you can see, Table `t2` involved in the column-level lineage of the create view statement, It also involved in a table-level lineage of the alter table statement. So, we use a pseudo column lineage to represent this table-level lineage in the column-level lineage model like this:

```
<relation id="3" type="fdd" effectType="rename_table">
 <target id="11" column="PseudoRows" parent_id="12" parent_name="t3" coordinate="[2,26,0],[2,28,0]" source="system"/>
 <source id="1" column="PseudoRows" parent_id="2" parent_name="t2" coordinate="[1,34,0],[1,36,0]" source="system"/>
</relation>
```

This pseudo column lineage will turn into table-level lineage when generate a table-level lineage model later like this:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<dlineage>
  <process id="9" name="Query Create View" type="Create View" coordinate="[1,1,0],[1,37,0]"/>
  <process id="13" name="Query Alter Table" type="Alter Table" coordinate="[2,1,0],[2,29,0]"/>
  <table id="2" name="t2" type="table" coordinate="[1,34,0],[1,36,0]"/>
  <table id="12" name="t3" type="table" processIds="13" coordinate="[2,26,0],[2,28,0]"/>
  <view id="8" name="v1" type="view" processIds="9" coordinate="[1,13,0],[1,15,0]"/>
  <relation id="307" type="fdd">
    <target id="308" target_id="9" target_name="Query Create View"/>
    <source id="302" source_id="2" source_name="t2"/>
  </relation>
  <relation id="309" type="fdd">
    <target id="301" target_id="8" target_name="v1"/>
    <source id="310" source_id="9" source_name="Query Create View"/>
  </relation>
  <relation id="311" type="fdd">
    <target id="312" target_id="13" target_name="Query Alter Table"/>
    <source id="305" source_id="2" source_name="t2"/>
  </relation>
  <relation id="313" type="fdd">
    <target id="304" target_id="12" target_name="t3"/>
    <source id="314" source_id="13" source_name="Query Alter Table"/>
  </relation>
</dlineage>
```

table-level diagram



SQLFlow UI

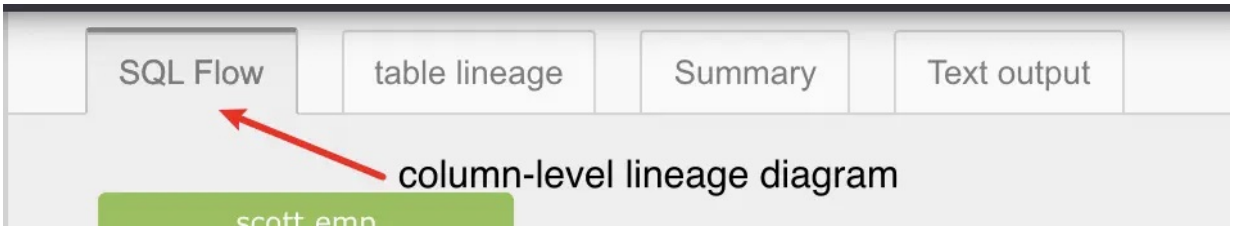


table-level lineage

The table-level lineage provides a higher level view for the dataflow in the data warehouse environment.

with the table-level lineage, you can grasp the data dataflow in a single picture.

The table-level lineage model is built on the data of the column-level model.

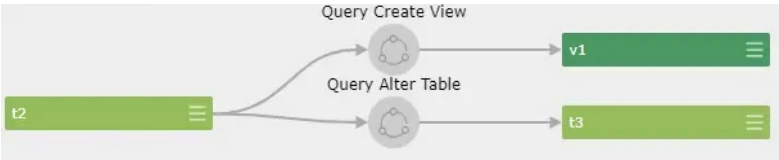
1. The table id and process id in the table-level model is the same as the one in column-level model.
2. The new table-level model uses table and process element from the column-level model and generate the new relation between the table and process.
3. Iterate target and source table in the column-level model, ignore all intermediate dataset such as resutlset, variable, and build relation between tables.
4. Iterate table-level realtion built in step 3 and according to the processId property in the table element, create the new relation by inserting the process between 2 tables.

```
create view v1 as select f1 from t2;
alter table t2 rename to t3;
```

column-level lineage

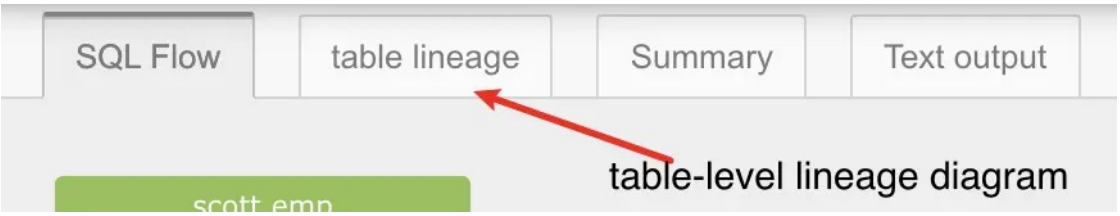

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<dlineage>
   <process id="9" name="Query Create View" type="Create View" coordinate="[1,1,0],[1,37,0]"/>
   <process id="13" name="Query Alter Table" type="Alter Table" coordinate="[2,1,0],[2,29,0]"/>
   <table id="2" name="t2" type="table" coordinate="[1,34,0],[1,36,0]">
      <column id="3" name="f1" coordinate="[1,26,0],[1,28,0]"/>
      <column id="1" name="PseudoRows" coordinate="[1,34,0],[1,36,0]" source="system"/>
   </table>
   <table id="12" name="t3" type="table" processIds="13" coordinate="[2,26,0],[2,28,0]">
      <column id="11" name="PseudoRows" coordinate="[2,26,0],[2,28,0]" source="system"/>
   </table>
   <view id="8" name="v1" type="view" processIds="9" coordinate="[1,13,0],[1,15,0]">
      <column id="10" name="f1" coordinate="[1,26,0],[1,28,0]"/>
   </view>
   <resultset id="5" name="RS-1" type="select_list" coordinate="[1,26,0],[1,28,0]">
      <column id="6" name="f1" coordinate="[1,26,0],[1,28,0]"/>
   </resultset>
   <relation id="1" type="fdd" effectType="select">
      <target id="6" column="f1" parent_id="5" parent_name="RS-1" coordinate="[1,26,0],[1,28,0]"/>
      <source id="3" column="f1" parent_id="2" parent_name="t2" coordinate="[1,26,0],[1,28,0]"/>
   </relation>
   <relation id="2" type="fdd" effectType="create_view">
      <target id="10" column="f1" parent_id="8" parent_name="v1" coordinate="[1,26,0],[1,28,0]"/>
      <source id="6" column="f1" parent_id="5" parent_name="RS-1" coordinate="[1,26,0],[1,28,0]"/>
   </relation>
   <relation id="3" type="fdd" effectType="rename_table">
      <target id="11" column="PseudoRows" parent_id="12" parent_name="t3" coordinate="[2,26,0],[2,28,0]" source="system"/>
      <source id="1" column="PseudoRows" parent_id="2" parent_name="t2" coordinate="[1,34,0],[1,36,0]" source="system"/>
   </relation>
</dlineage>
```

table-level lineage




```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<dlineage>
  <process id="9" name="Query Create View" type="Create View" coordinate="[1,1,0],[1,37,0]"/>
  <process id="13" name="Query Alter Table" type="Alter Table" coordinate="[2,1,0],[2,29,0]"/>
  <table id="2" name="t2" type="table" coordinate="[1,34,0],[1,36,0]"/>
  <table id="12" name="t3" type="table" processIds="13" coordinate="[2,26,0],[2,28,0]"/>
  <view id="8" name="v1" type="view" processIds="9" coordinate="[1,13,0],[1,15,0]"/>
  <relation id="307" type="fdd">
    <target id="308" target_id="9" target_name="Query Create View"/>
    <source id="302" source_id="2" source_name="t2"/>
  </relation>
  <relation id="309" type="fdd">
    <target id="301" target_id="8" target_name="v1"/>
    <source id="310" source_id="9" source_name="Query Create View"/>
  </relation>
  <relation id="311" type="fdd">
    <target id="312" target_id="13" target_name="Query Alter Table"/>
    <source id="305" source_id="2" source_name="t2"/>
  </relation>
  <relation id="313" type="fdd">
    <target id="304" target_id="12" target_name="t3"/>
    <source id="314" source_id="13" source_name="Query Alter Table"/>
  </relation>
</dlineage>
```

SQLFlow UI



lineage model - table

table

Table type represents the table object in a relational database.

It also represents the derived table such as function table.

struct definition


```
{
  "elementName" : "table",
  "attributeDefs": [
    {
      "name": "id",
      "typeName": "int",
      "isOptional": false,
      "isUnique": true
    },
    {
      "name": "name",
      "typeName": "string",
      "isOptional": false
    },
    {
      "name": "alias",
      "typeName": "string",
      "isOptional": true
    },
    {
      "name": "type",
      "typeName": "string",
      "isOptional": false
    },
    {
      "name": "subType",
      "typeName": "string",
      "isOptional": false
    },
    {
      "name": "database",
      "typeName": "string",
      "isOptional": true
    },
    {
      "name": "schema",
      "typeName": "string",
      "isOptional": true
    },
    {
      "name": "coordinate",
      "typeName": "string",
      "isOptional": true
    },
    {
      "name": "processIds",
      "typeName": "int",
      "isOptional": true
    },
    {
      "name": "columns",
      "typeName": "array<column>",
      "isOptional": true
    }
  ]
}
```

id

unique id in the output.

name

table name in the original SQL query.

alias

alias of the table in the original SQL query.

type

type of the table, available value: `table` , `pseudoTable`

- table

This means a base table found in the SQL query.

```
create view v123 as select a,b
from employee a, name b
where employee.id = name.id
```

```
<table id="2" name="employee" alias="a" type="table">
```

- pseudoTable

Due to the lack of metadata information, some columns can't be linked to a table correctly. Those columns will be assigned to a pseudo table with name: `pseudo_table_include_orphan_column` . The type of this table is `pseudoTable` .

In the following sample sql, columm `a` , `b` can't be linked to a specific table without enough information, so a pseudo table with name `pseudo_table_include_orphan_column` is created to contain those orphan columns.

```
create view v123 as
select a,b from employee a, name b where employee.id = name.id
```

```
<table id="11" name="pseudo_table_include_orphan_column" type="pseudoTable" coordinate="[1,1,f904f8312239df09d5e008bb9d69b466],[1,
 <column id="12" name="a" coordinate="[1,28,f904f8312239df09d5e008bb9d69b466],[1,29,f904f8312239df09d5e008bb9d69b466]"/>
 <column id="14" name="b" coordinate="[1,30,f904f8312239df09d5e008bb9d69b466],[1,31,f904f8312239df09d5e008bb9d69b466]"/>
</table>
```

subType

In the most case of SQL query, the table used is a base table. However, derived tables are also used in the from clause or other places.

The `subType` property in the `table` element tells you what kind of the derived table this table is.

Take the following sql for example, `WarehouseReporting.dbo.fnListToTable` is a function that used as a derived table. So, the value of `subType` is `function` .

Currently(GSP 2.2.0.6), `function` is the only value of `subType` . More value of `tableType` will be added in the later version such as `JSON_TABLE` for JSON_TABLE.

```
select entry as Account FROM WarehouseReporting.dbo.fnListToTable(@AccountList)
```

```
<table id="2" database="WarehouseReporting" schema="dbo" name="WarehouseReporting.dbo.fnListToTable" type="table" tableType="functi
 <column id="3" name="entry" coordinate="[1,8,15c3ec5e6df0919bb570c4d8cdd66651],[1,13,15c3ec5e6df0919bb570c4d8cdd66651]"/>
</table>
```

database

The database this table belongs to.

schema

The schema this table belongs to.

coordinate

Indicates the positions the table occurs in the SQL script.

`coordinate="[1,37,0],[1,47,0]"`

the first number is line， the second number is column, the third number is **?**

processIds

The Id of the process which is doing the transformation related to this table. This `processIds` is used when generate table-level lineage model.

columns

Array of `column` beblogs to this table.

lineage model - view

view

struct definition


```
{
  "elementName" : "view",
  "attributeDefs": [
    {
      "name": "id",
      "typeName": "int",
      "isOptional": false,
      "isUnique": true
    },
    {
      "name": "name",
      "typeName": "string",
      "isOptional": false
    },
    {
      "name": "alias",
      "typeName": "string",
      "isOptional": true
    },
    {
      "name": "type",
      "typeName": "string",
      "isOptional": false
    },
    {
      "name": "database",
      "typeName": "string",
      "isOptional": true
    },
    {
      "name": "schema",
      "typeName": "string",
      "isOptional": true
    },
    {
      "name": "processIds",
      "typeName": "int",
      "isOptional": true
    },
    {
      "name": "coordinate",
      "typeName": "string",
      "isOptional": true
    },
    {
      "name": "columns",
      "typeName": "array<column>",
      "isOptional": true
    }
  ]
}
```


id

unique id in the output.

name

view name in the original SQL query.

alias

alias of the view in the original SQL query.

type

type of the view, available value: `view`

processIds

lineage model - resultset

resultset

This is the intermediate recordset generated during the process of SQL query such as a select list.

struct definition

```
{
  "elementName" : "resultset",
  "attributeDefs": [
    {
      "name": "id",
      "typeName": "int",
      "isOptional": false,
      "isUnique": true
    },
    {
      "name": "name",
      "typeName": "string",
      "isOptional": false
    },
    {
      "name": "alias",
      "typeName": "string",
      "isOptional": true
    },
    {
      "name": "type",
      "typeName": "string",
      "isOptional": false
    },
    {
      "name": "coordinate",
      "typeName": "string",
      "isOptional": true
    },
    {
      "name": "columns",
      "typeName": "array<column>",
      "isOptional": true
    }
  ]
}
```

id

the unique id in the output.

name

name of this resultset.

alias

alias of this resultset.

type

type of the reseultset, available value: `select_list` , `merge-insert` , `merge_update` , `update_set` , `update-select` , `insert-select` .

lineage model - relation

relation

Relation represents the column-level lineage. It includes one target column, one or more source columns.

struct definition

```
  "elementName" : "relation",
  "attributeDefs": [
    {
      "name": "id",
      "typeName": "int",
      "isOptional": false,
      "isUnique": true
    },
    {
      "name": "type",
      "typeName": "string",
      "isOptional": false
    },
    {
      "name": "effectType",
      "typeName": "string",
      "isOptional": true
    },
    {
      "name": "target",
      "typeName": "targetElement",
      "isOptional": false
    },
    {
      "name": "source",
      "typeName": "array<sourceElement>",
      "isOptional": false
    }
  ]
}
```

id

unique id in the output.

type

type of the column-lineage, available value: `fdd` , `fdr` .

Please check dbobjects_relationship for the detailed information.

effectType

This is the SQL statement that generate this relation. Available values: `select` , `create_view`

target,source element

```
{
  "elementName" : "target",
  "attributeDefs": [
    {
      "name": "id",
      "typeName": "int",
      "isOptional": false,
      "isUnique": true
    },
    {
```

```
        "name": "column",
        "typeName": "string",
        "isOptional": false
      },
      {
        "name": "parent_id",
        "typeName": "int",
        "isOptional": false
      },
      {
        "name": "parent_name",
        "typeName": "string",
        "isOptional": false
      },
      {
        "name": "source",
        "typeName": "string",
        "isOptional": true
      },
      {
        "name": "clauseType",
        "typeName": "string",
        "isOptional": true
      },
      {
        "name": "coordinate",
        "typeName": "string",
        "isOptional": true
      }
    ]
}
```

id

the unique id in the output.

column

The name of the column.

There is a specific column name: `PseudoRows`, which represents the number of rows in the table/view/resultset. Check here for more information.

parent_id

This is usually the id of a table that this columns belongs.

parent_name

This is usually the name of a table that this columns belongs.

source

If the value of source is `system`, this means the column doesn't comes from the SQL query. It's generated by SQLFlow.

clauseType

Where this column comes from, such as where clause.

lineage model - process

process

This is the SQL statement that transforms the data.

struct definition

```
{
  "elementName" : "process",
  "attributeDefs": [
    {
      "name": "id",
      "typeName": "int",
      "isOptional": false,
      "isUnique": true
    },
    {
      "name": "type",
      "typeName": "string",
      "isOptional": false
    },
    {
      "name": "coordinate",
      "typeName": "string",
      "isOptional": false
    }
  ]
}
```

id

the unique id in the output.

name

table name in the original SQL query.

type

type of the process, usually, it's the type of SQL statement that do the data transformation. Available value: `insert`

lineage model - column

column

struct definition

```
{
   "elementName" : "column",
   "attributeDefs": [
      {
         "name": "id",
         "typeName": "int",
         "isOptional": false,
         "isUnique": true
      },
      {
         "name": "name",
         "typeName": "string",
         "isOptional": false
      },
      {
         "name": "coordinate",
         "typeName": "string",
         "isOptional": false
      }
   ]
}
```

id

the unique id in the output.

name

column name in the original SQL query.

coordinate

Indicates the positions of the occurences of the column in the SQL script.

lineage model - variable

variable

the variable used in the SQL especially in the stored procedure.

struct definition

```
{
  "elementName" : "variable",
  "attributeDefs": [
    {
      "name": "id",
      "typeName": "int",
      "isOptional": false,
      "isUnique": true
    },
    {
      "name": "type",
      "typeName": "string",
      "isOptional": false
    },
    {
      "name": "subType",
      "typeName": "string",
      "isOptional": false
    },
    {
      "name": "coordinate",
      "typeName": "string",
      "isOptional": false
    },
    {
      "name": "columns",
      "typeName": "array<column>",
      "isOptional": true
    }
  ]
}
```

id

the unique id in the output.

name

variable name in the original SQL query.

type

This value is always be `type`

subType

type of the variable, one of those values: `scalar` , `cursor` , `record`

columns

Array of column name in the cursor/record variable. Or the variable name of the scalar variable.

lineage model - procedure

procedure

Represents a stored procedure.

struct definition

```
{
  "elementName" : "procedure",
  "attributeDefs": [
    {
      "name": "id",
      "typeName": "int",
      "isOptional": false,
      "isUnique": true
    },
    {
      "name": "name",
      "typeName": "string",
      "isOptional": false
    },
    {
      "name": "type",
      "typeName": "string",
      "isOptional": false
    },
    {
      "name": "coordinate",
      "typeName": "string",
      "isOptional": false
    },
    {
      "name": "arguments",
      "typeName": "array<argument>",
      "isOptional": true
    }
  ]
}
```

id

the unique id in the output.

name

procedure name in the original SQL query.

type

One of those values: `createprocedure`

coordinate

Indicates the positions of the occurrences in the SQL script.

argument

argument of the stored procedure

struct definition

```
{
  "elementName" : "argument",
  "attributeDefs": [
    {
```

```
        "name": "id",
        "typeName": "int",
        "isOptional": false,
        "isUnique": true
      },
      {
        "name": "name",
        "typeName": "string",
        "isOptional": false
      },
      {
        "name": "datatype",
        "typeName": "string",
        "isOptional": false
      },
      {
        "name": "coordinate",
        "typeName": "string",
        "isOptional": false
      },
      {
        "name": "inout",
        "typeName": "string",
        "isOptional": true
      }
    ]
}
```

lineage model - path

path

This is the path such as hdfs path, Amazon S3 path, BigQuery GS path.

struct definition

```
{
  "elementName" : "path",
  "attributeDefs": [
    {
      "name": "id",
      "typeName": "int",
      "isOptional": false,
      "isUnique": true
    },
    {
      "name": "uri",
      "typeName": "string",
      "isOptional": false
    },
    {
      "name": "coordinate",
      "typeName": "string",
      "isOptional": false
    }
  ]
}
```

id

the unique id in the output.

uri

the path where the object is stored.

Lineage model elements on UI

lineage model elements on UI

Entity

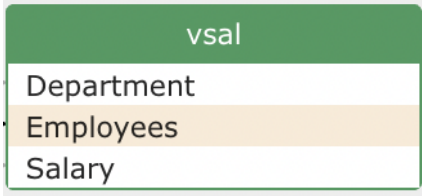
path in the json: `data->sqlflow->dbobjs`

1. Permanent entity

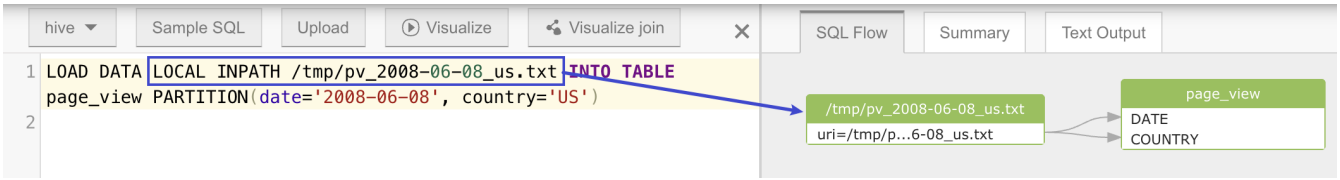
1. table

2. external table

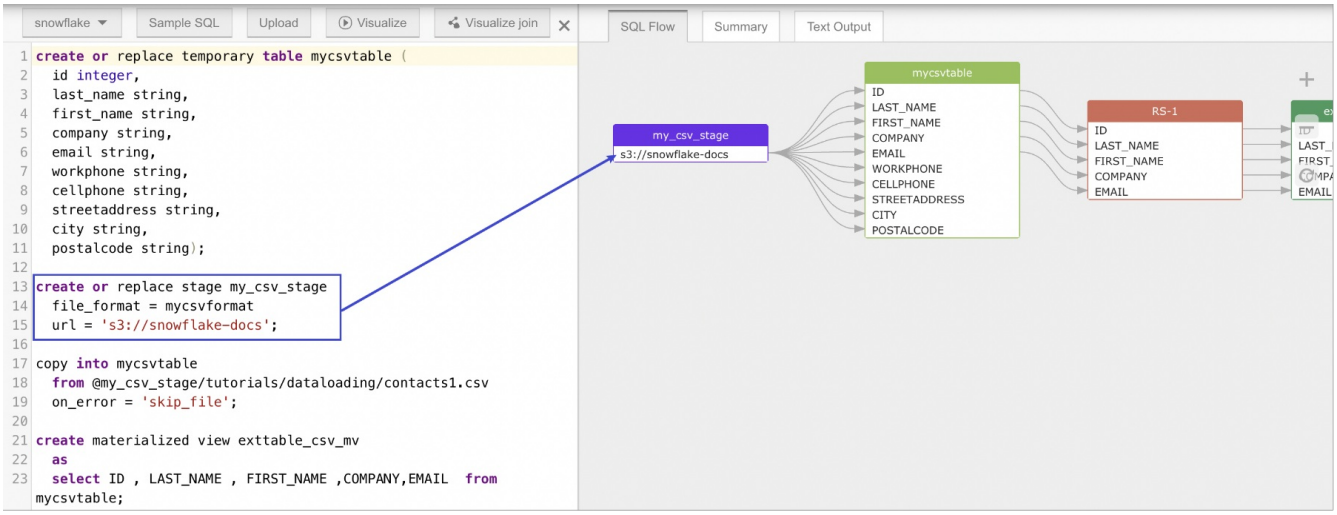
3. view



4. hive local directory/inpath

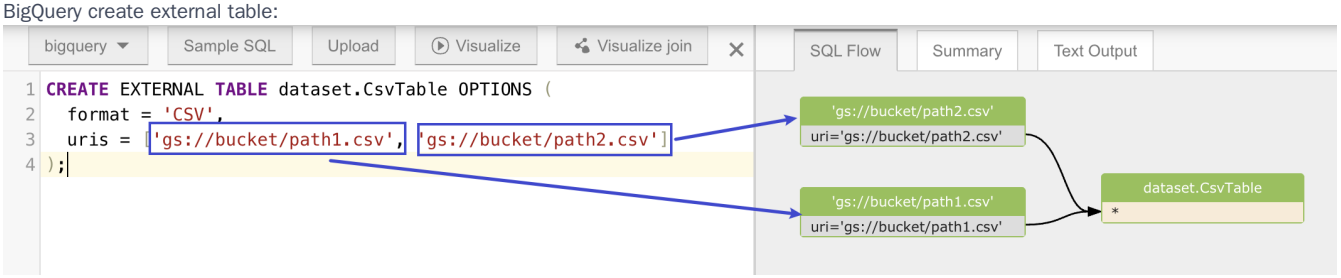


5. snowflake stage



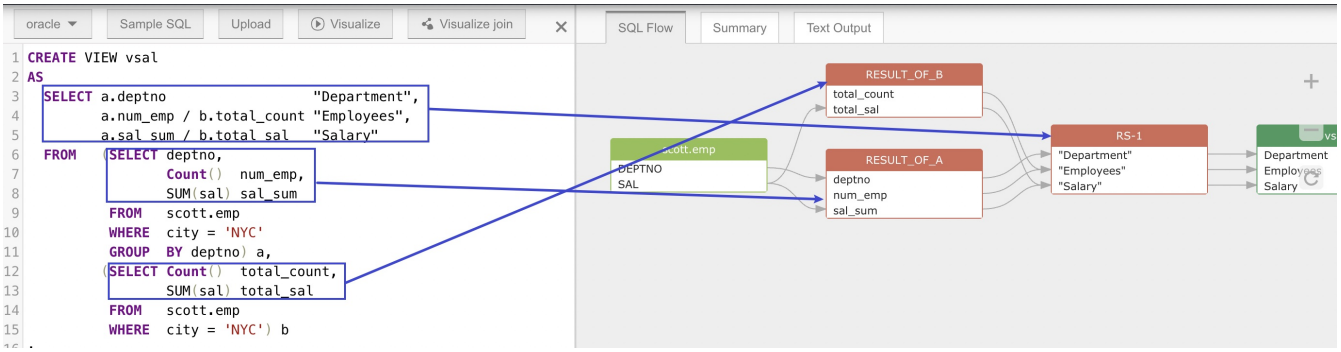
6. bigquery file uri

BigQuery create external table:

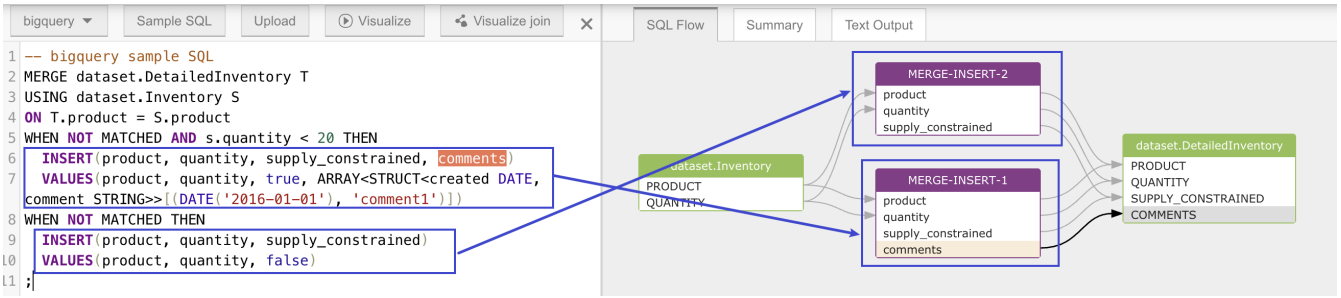


2. temporary entity

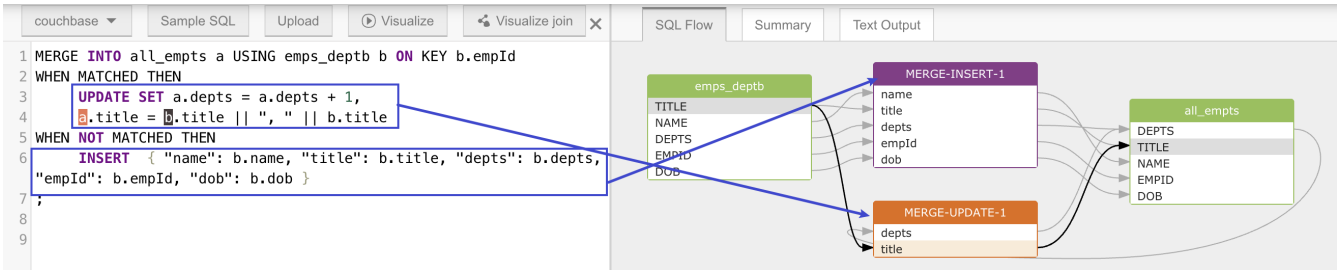
1. select_list



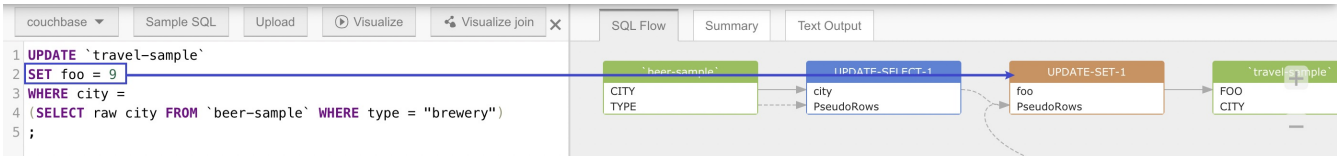
2. merge_insert



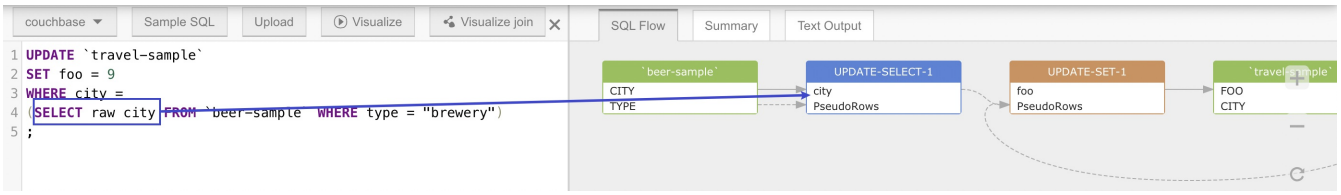
3. merge_update



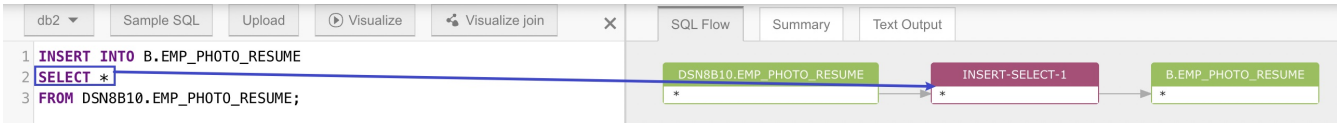
4. update_set



5. update-select

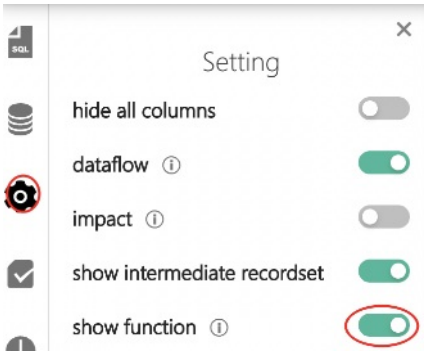


6. insert-select

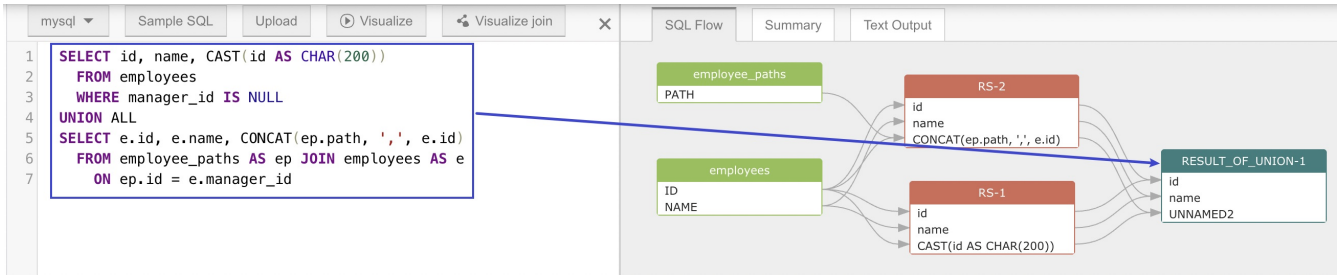


7. function

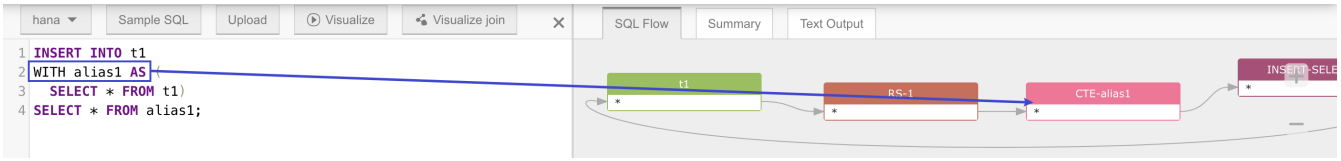
In order to show the function in the result, please turn on this setting:



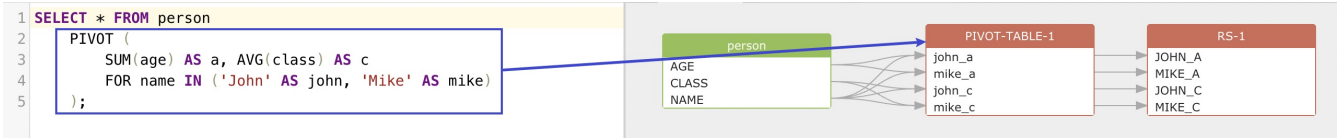
8. union



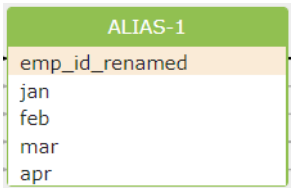
9. cte



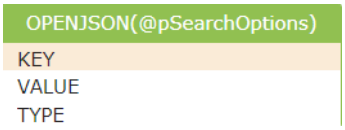
10. pivot table



11. snowflake pivot alias



12. mssql open json

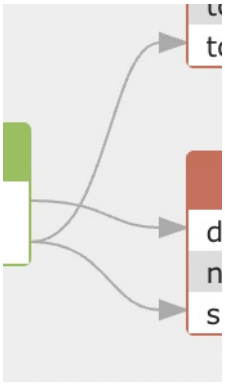


13. mssql json property

relationship

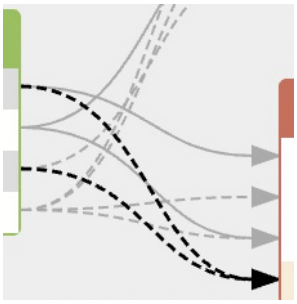
path in the json: `data->sqlflow->relations`

1. fdd, data flow



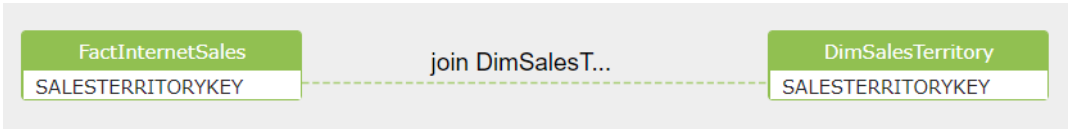
2. fdr, frd data impact

dash line



3. join

dash line



PseudoRows column

As it's name indicates, PseudoRows column doesn't exists in a table but created due to the following reasons.

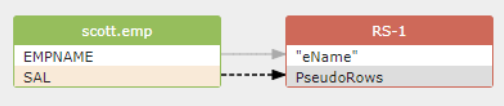
1. represents the total number of columns in a table/resultset

```
SELECT a.empName "eName"
FROM scott.emp a
Where sal > 1000
```

The total number of rows in the select list is impacted by the value of column `sal` in the where clause. So we have a dataflow relation like this:

```
sal -> fdr -> resultset.PseudoRows
```

diagram



2. In order to put a table involves in both column-level lineage and table-level lineage into one picture

```
create view v1 as select f1 from t2;
alter table t2 rename to t3;
```

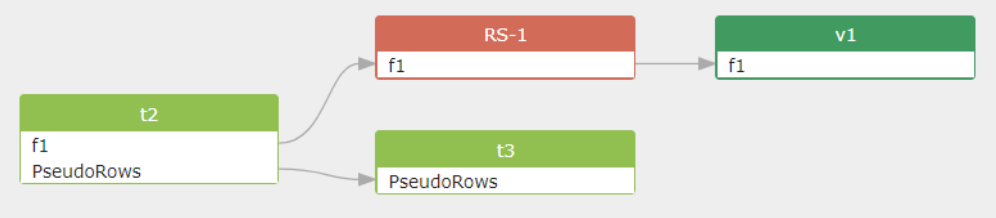
The first create view statement will generate a column-level lineage of the table `t2`,

```
t2.f1 -> fdd -> RS-1.f1 -> fdd -> v1.f1
```

while the second alter table statement will genereate a table-level lineage of the table t2.

```
t2.PseudoRows -> table-level lineage -> t3.PseudoRows
```

diagram



3. More use cases of PseudoRows column

1. where clause
2. group by and aggregate function
3. fdr via from clause
4. join condition
5. rename and swap table

Lineage in real SQL

Handle the dataflow chain

Handle the dataflow chain

Every relation in the SQL is picked up by the tool, and connected together to show the whole dataflow chain. Sometimes, we only need to see the end to end relation and ignore all the intermediate relations.

If we need to convert a fully chained dataflow to an `end to end` dataflow, we may consider the following rules:

1. A single dataflow chain with the mixed relation types: fdd and fdr.

   ```
   A -> fdd -> B -> fdr -> C -> fdd -> D
   ```

   the rule is: if any `fdr` relation appears in the chain, the relation from `A -> D` will be consider as type of `fdr`, otherwise, the final relation is `fdd` for the end to end relation of `A -> D`.

2. If there are multiple chains from `A -> D`

   ```
   A -> fdd -> B1 -> fdr -> C1 -> fdd -> D
   A -> fdd -> B2 -> fdr -> C1 -> fdd -> D
   A -> fdd -> B3 -> fdd -> C3 -> fdd -> D
   ```

   The final relation should choose the `fdd` chain if any.

variable

cursor, record variable

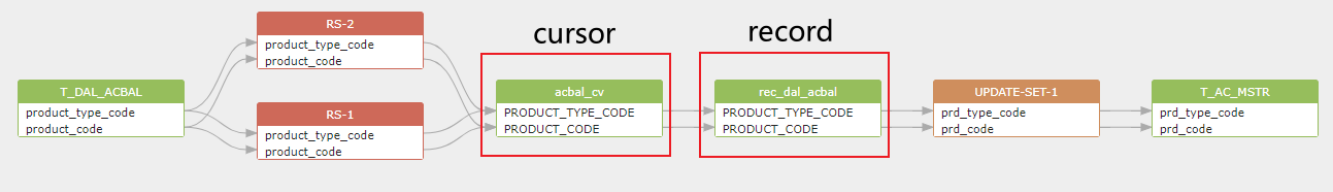
This is an Oracle PLSQL.

```
DECLARE
 p_run_ind VARCHAR2;
 TYPE acbal_cv IS REF CURSOR;
 rec_dal_acbal T_DAL_ACBAL%ROWTYPE;
BEGIN

IF p_run_ind = 'STEP1' THEN
 OPEN acbal_cv FOR
  SELECT product_type_code,product_code FROM T_DAL_ACBAL
   WHERE AC_CODE > ' ' AND UPDT_FLG != '0'
   AND UPDAT_FLG != '3' AND ROWNUM < 150001;

ELSIF p_run_ind = 'STEP2' THEN
 OPEN acbal_cv FOR
  SELECT product_type_code,product_code FROM T_DAL_ACBAL
   WHERE AC_CODE > ' ' AND UPDT_FLG != '0'
   AND UPDAT_FLG != '3';

END IF;

LOOP
 FETCH acbal_cv INTO rec_dal_acbal;
 EXIT WHEN cur_stclerk%NOTFOUND;

 UPDATE T_AC_MSTR
 SET prd_type_code = rec_dal_acbal.product_type_code,
  prd_code = rec_dal_acbal.product_code
 ;

END LOOP;

COMMIT;
END;
```

dataflow in xml

```
<variable id="2" name="acbal_cv" type="variable" subType="cursor" coordinate="[9,7,0],[9,15,0]">
 <column id="14" name="*" coordinate="[1,1,0],[1,2,0]"/>
 <column id="14_0" name="PRODUCT_TYPE_CODE" coordinate="[1,1,0],[1,2,0]"/>
 <column id="14_1" name="PRODUCT_CODE" coordinate="[1,1,0],[1,2,0]"/>
</variable>
<variable id="25" name="rec_dal_acbal" type="variable" subType="record" coordinate="[23,22,0],[23,35,0]">
 <column id="26" name="*" coordinate="[1,1,0],[1,2,0]"/>
 <column id="26_0" name="PRODUCT_TYPE_CODE" coordinate="[1,1,0],[1,2,0]"/>
 <column id="26_1" name="PRODUCT_CODE" coordinate="[1,1,0],[1,2,0]"/>
</variable>
```

diagram



scalar variable

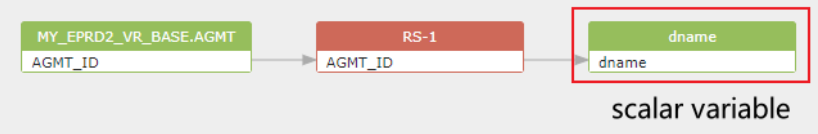
This is a Teradata stored procedure

```
CREATE PROCEDURE NewProc (IN id CHAR(12),
IN pname INTEGER,
IN pid INTEGER,
OUT dname CHAR(10))
BEGIN

 SELECT AGMT_ID
 INTO dname FROM MY_EPRD2_VR_BASE.AGMT
 WHERE PROCESS_ID = pid;
END;
```

dataflow in xml

```
<variable id="14" name="dname" type="variable" subType="scalar" coordinate="[8,7,0],[8,12,0]">
 <column id="15" name="dname" coordinate="[8,7,0],[8,12,0]"/>
</variable>
```

diagram



rename and swap table

```
create view v1 as select f1 from t2;
alter table t2 rename to t3;
```

column-level lineage mode

In order to put a table involved in both column-level lineage and table-level lineage into one picture,we use `PseudoRows` column in order to represent this relation.

```
t2.PseudoRows -> fdd -> t3.PseudoRows
```

diagram

This is the diagram show lineage in column-level mode.

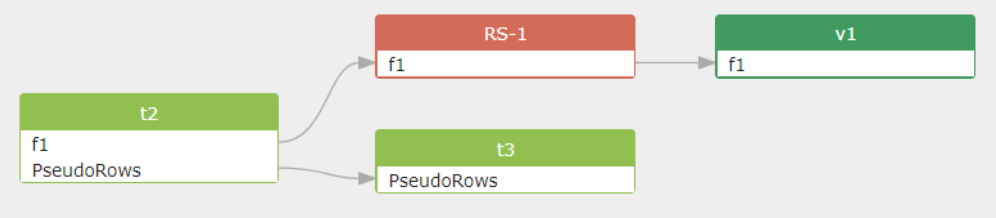


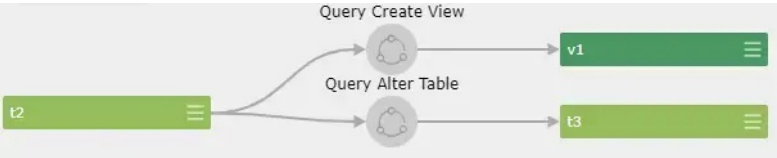
table-level lineage mode

If we want to show the table in above SQL in a table-level lineage mode, the relation between 2 tables should be represented by another form like this:

```
t2 -> query process (create view) -> v1
t2 -> query process (alter table rename) -> t3
```

diagram

This is the diagram show lineage in table-level mode.



dataflow in xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<dlineage>
  <process id="9" name="Query Create View" type="Create View" coordinate="[1,1,0],[1,37,0]"/>
  <process id="13" name="Query Alter Table" type="Alter Table" coordinate="[2,1,0],[2,29,0]"/>
  <table id="2" name="t2" type="table" coordinate="[1,34,0],[1,36,0]"/>
  <table id="12" name="t3" type="table" processIds="13" coordinate="[2,26,0],[2,28,0]"/>
  <view id="8" name="v1" type="view" processIds="9" coordinate="[1,13,0],[1,15,0]"/>
  <relation id="307" type="fdd">
    <target id="308" target_id="9" target_name="Query Create View"/>
    <source id="302" source_id="2" source_name="t2"/>
  </relation>
  <relation id="309" type="fdd">
    <target id="301" target_id="8" target_name="v1"/>
    <source id="310" source_id="9" source_name="Query Create View"/>
  </relation>
  <relation id="311" type="fdd">
    <target id="312" target_id="13" target_name="Query Alter Table"/>
    <source id="305" source_id="2" source_name="t2"/>
```

```
    </relation>
    <relation id="313" type="fdd">
       <target id="304" target_id="12" target_name="t3"/>
       <source id="314" source_id="13" source_name="Query Alter Table"/>
    </relation>
</dlineage>
```

insert overwrite (Hive)

```
INSERT OVERWRITE LOCAL DIRECTORY '/tmp/pv_gender_sum'
SELECT pv_gender_sum.*
FROM pv_gender_sum;
```

column-level lineage

The data flow is:

```
pv_gender_sum(*) ->  fdd ->  path ( uri='/tmp/pv_gender_sum')
```

dataflow in xml

diagram

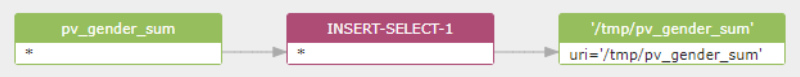


table-level lineage

```
pv_gender_sum ->  query process (insert overwrite) ->  path ( uri='/tmp/pv_gender_sum')
```

foreign key

The foreign key in create table statement will create a column-level lineage.

```
CREATE TABLE masteTable
(
 masterColumn        varchar(3)  Primary Key,
);


CREATE TABLE foreignTable
(
 foreignColumn1          varchar(3)  NOT NULL ,
 foreignColumn2          varchar(3)  NOT NULL
 FOREIGN KEY (foreignColumn1) REFERENCES masteTable(masterColumn),
 FOREIGN KEY (foreignColumn2) REFERENCES masteTable(masterColumn)
)
```

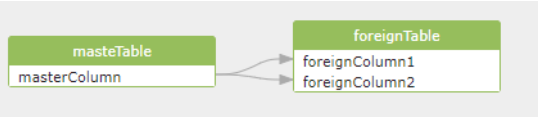
The data flow is:

```
masteTable.masterColumn -> fdd -> foreignTable.foreignColumn1
masteTable.masterColumn -> fdd -> foreignTable.foreignColumn2
```

dataflow in xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<dlineage>
   <table id="2" name="masteTable" type="table" coordinate="[1,14,0],[1,24,0]">
      <column id="3" name="masterColumn" coordinate="[3,2,0],[3,14,0]"/>
   </table>
   <table id="5" name="foreignTable" type="table" coordinate="[7,14,0],[7,26,0]">
      <column id="10" name="foreignColumn1" coordinate="[9,2,0],[9,16,0]"/>
      <column id="11" name="foreignColumn2" coordinate="[10,2,0],[10,16,0]"/>
   </table>
   <relation id="1" type="fdd">
      <target id="10" column="foreignColumn1" parent_id="5" parent_name="foreignTable" coordinate="[9,2,0],[9,16,0]"/>
      <source id="3" column="masterColumn" parent_id="2" parent_name="masteTable" coordinate="[3,2,0],[3,14,0]"/>
   </relation>
   <relation id="2" type="fdd">
      <target id="11" column="foreignColumn2" parent_id="5" parent_name="foreignTable" coordinate="[10,2,0],[10,16,0]"/>
      <source id="3" column="masterColumn" parent_id="2" parent_name="masteTable" coordinate="[3,2,0],[3,14,0]"/>
   </relation>
</dlineage>
```

diagram



create external table (path)

> This feature is working in process in current version 2021/7/7

create table external table usually will use `path` object.

snowflake create external

```
create or replace stage exttable_part_stage
  url='s3://load/encrypted_files/'
  credentials=(aws_key_id='1a2b3c' aws_secret_key='4x5y6z')
  encryption=(type='AWS_SSE_KMS' kms_key_id = 'aws/key');

create external table exttable_part(
 date_part date as to_date(split_part(metadata$filename, '/', 3)
   || '/' || split_part(metadata$filename, '/', 4)
   || '/' || split_part(metadata$filename, '/', 5), 'YYYY/MM/DD'),
 timestamp bigint as (value:timestamp::bigint),
 col2 varchar as (value:col2::varchar))
 partition by (date_part)
 location=@exttable_part_stage/logs/
 auto_refresh = true
 file_format = (type = parquet);
```

The data of the external table `exttable_part` comes from the `path ('s3://load/encrypted_files/')` via the stage: `exttable_part_stage`

```
path('s3://load/encrypted_files/') -> fdd -> exttable_part_stage (url) -> fdd -> exttable_part(date_part,timestamp,col2)
```

dataflow in xml

diagam

table-level lineage

this SQL is able to create a table-level lineage like this:

```
path('s3://load/encrypted_files/') -> process(create stage) -> exttable_part_stage (url) -> process(create external table) -> exttable_part
```

bigquery create external table

```
CREATE EXTERNAL TABLE dataset.CsvTable OPTIONS (
  format = 'CSV',
  uris = ['gs://bucket/path1.csv', 'gs://bucket/path2.csv']
);
```

The data of the external table `dataset.CsvTable` comes from the csv file: `gs://bucket/path1.csv, gs://bucket/path2.csv`

```
path (uri='gs://bucket/path1.csv')  -> fdd ->  dataset.CsvTable
path (uri='gs://bucket/path2.csv')  -> fdd ->  dataset.CsvTable
```

dataflow in xml

diagam

table-level lineage

This SQL is able to create a table-level lineage like this:

```
path (uri='gs://bucket/path1.csv')  -> query process(create external table) ->  dataset.CsvTable
path (uri='gs://bucket/path2.csv')  -> query process(create external table) ->  dataset.CsvTable
```

Hive load data

```
LOAD DATA LOCAL INPATH /tmp/pv_2008-06-08_us.txt INTO TABLE page_view PARTITION(date='2008-06-08', country='US')
```

The data flow is:

```
path (uri='/tmp/pv_2008-06-08_us.txt')  ->  fdd -> page_view(date,country)
```

dataflow in xml

diagram

table-level lineage

```
path (uri='/tmp/pv_2008-06-08_us.txt')  -> query process(load data) -> page_view
```

case expression (fdd)

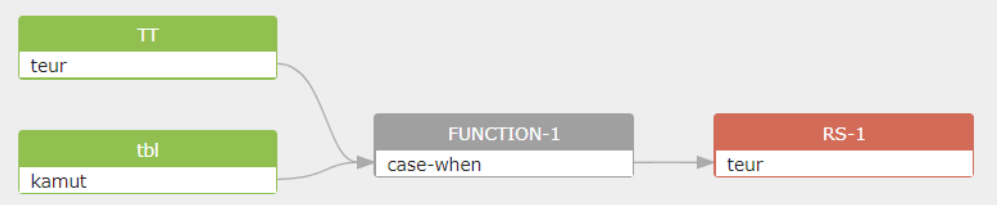
case expression

```
select
 case when a.kamut=1 and b.teur IS null
    then 'no locks'
   when a.kamut=1
   then b.teur
 else 'locks'
 end teur
from tbl a left join TT b on (a.key=b.key)
```

During the analyzing of dataflow, case expression is treated as a function. The column used inside the case expression will be treated like the arguments of a function. So for the above SQL, the following relation is discovered:

```
tbl.kamut -> fdd -> teur
TT.teur -> fdd -> teur
```

diagram



create view

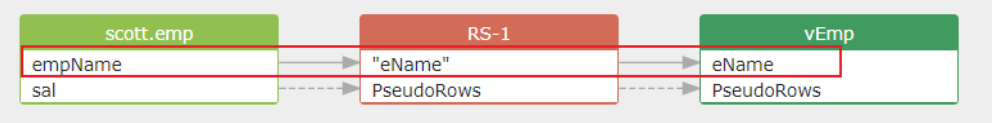
```
create view vEmp(eName) as
SELECT a.empName "eName"
FROM scott.emp a
Where sal > 1000
```

fdd

Data in the column `eName` of the view `vEmp` comes from column `empName` of the table `scott.emp` via the chain like this:

```
scott.emp.empName -> fdd -> RS-1."eName" -> vEmp.eName
```

diagram



fdr

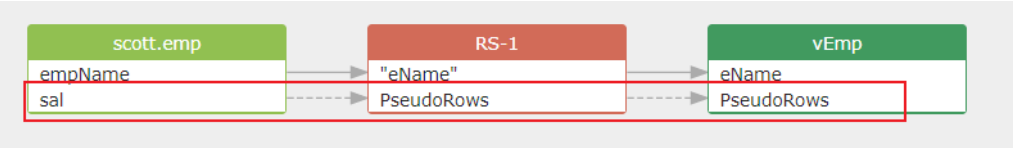
From this query, you will see how the column `sal` in where clause impact the number of rows in the top level view `vEmp`.

```
scott.emp.sal -> fdr -> resultset1.PseudoRows -> fdr -> vEmp.PseudoRows
```

So, from an end to end point of view, there will be a `fdr` relation between column `sal` and view `vEmp` like this:

```
scott.emp.sal -> fdr -> vEmp.PseudoRows
```

diagram



select list (fdd)

This article introduce a basic dataflow generated by GSP.

Column with alias

```
SELECT a.empName "eName"
FROM scott.emp a
Where sal > 1000
```

the data of target column `"eName"` comes from `scott.emp.empName` (represented by `fdd`), so we have a dataflow relation like this:

```
scott.emp.empName -> fdd -> "eName"
```

the result generated by the select list called: `resultset` likes a virtual table includes columns and rows.

dataflow in XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<dlineage>
   <table id="2" schema="scott" name="scott.emp" alias="a" type="table" coordinate="[2,6,0],[2,17,0]">
      <column id="3" name="empName" coordinate="[1,8,0],[1,17,0]"/>
   </table>
   <resultset id="5" name="RS-1" type="select_list" coordinate="[1,8,0],[1,25,0]">
      <column id="6" name=""eName"" coordinate="[1,8,0],[1,25,0]"/>
   </resultset>
   <relation id="1" type="fdd" effectType="select">
      <target id="6" column=""eName"" parent_id="5" parent_name="RS-1" coordinate="[1,8,0],[1,25,0]"/>
      <source id="3" column="empName" parent_id="2" parent_name="scott.emp" coordinate="[1,8,0],[1,17,0]"/>
   </relation>
</dlineage>
```

The `relation` represents a dataflow from source column with `id=3` to the target column with `id=6`

diagram

Column uses function

During the dataflow analyzing, `function` plays a key role. It accepts arguments which usually is column and generate resultset which maybe a scalar value or a set value.

```
select round(salary) as sal from scott.emp
```

The relation of the `round` function in the above SQL :

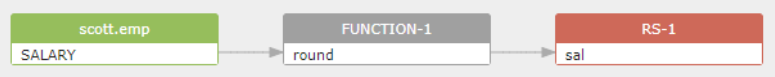
```
scott.emp.salary -> fdd -> round(salary) -> fdd -> sal
```

dataflow in xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<dlineage>
   <table id="2" schema="scott" name="scott.emp" type="table" coordinate="[1,34,0],[1,43,0]">
      <column id="3" name="salary" coordinate="[1,14,0],[1,20,0]"/>
   </table>
   <resultset id="5" name="RS-1" type="select_list" coordinate="[1,8,0],[1,28,0]">
      <column id="6" name="sal" coordinate="[1,8,0],[1,28,0]"/>
   </resultset>
   <resultset id="8" name="FUNCTION-1" type="function" coordinate="[1,8,0],[1,21,0]">
```

```
      <column id="9" name="round" coordinate="[1,8,0],[1,13,0]"/>
    </resultset>
    <relation id="1" type="fdd" effectType="select">
      <target id="6" column="sal" parent_id="5" parent_name="RS-1" coordinate="[1,8,0],[1,28,0]"/>
      <source id="9" column="round" parent_id="8" parent_name="FUNCTION-1" coordinate="[1,8,0],[1,13,0]"/>
    </relation>
    <relation id="2" type="fdd" effectType="function">
      <target id="9" column="round" parent_id="8" parent_name="FUNCTION-1" coordinate="[1,8,0],[1,13,0]"/>
      <source id="3" column="salary" parent_id="2" parent_name="scott.emp" coordinate="[1,14,0],[1,20,0]"/>
    </relation>
  </dlineage>
```

diagram



if you turn off the `show function` setting with `/if` option, the result is:

References

1. xml code used in this article is generated byDataFlowAnalyzer tools
2. digram used in this article is generated by theSQLFlow Cloud version

where clause (fdr)

fdr type

```
SELECT a.empName "eName"
FROM scott.emp a
Where sal > 1000
```

The total number of rows in the select list is impacted by the value of column `sal` in the where clause. So we have a dataflow relation like this:

```
sal -> fdr -> resultset.PseudoRows
```

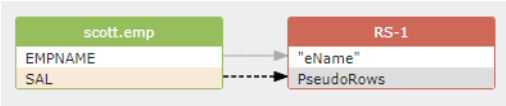
PseudoRows column

As you can see, we introduced a new pseudo column: `PseudoRows` to represents the number of rows in the resultset.

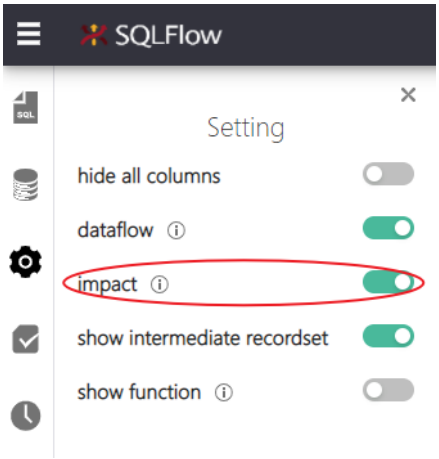
dataflow in xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<dlineage>
  <table id="2" schema="scott" name="scott.emp" alias="a" type="table" coordinate="[2,6,0],[2,17,0]">
    <column id="3" name="empName" coordinate="[1,8,0],[1,17,0]"/>
    <column id="4" name="sal" coordinate="[3,7,0],[3,10,0]"/>
  </table>
  <resultset id="6" name="RS-1" type="select_list" coordinate="[1,8,0],[1,25,0]">
    <column id="7" name=""eName"" coordinate="[1,8,0],[1,25,0]"/>
    <column id="5" name="PseudoRows" coordinate="[1,8,0],[1,25,0]" source="system"/>
  </resultset>
  <relation id="1" type="fdd" effectType="select">
    <target id="7" column=""eName"" parent_id="6" parent_name="RS-1" coordinate="[1,8,0],[1,25,0]"/>
    <source id="3" column="empName" parent_id="2" parent_name="scott.emp" coordinate="[1,8,0],[1,17,0]"/>
  </relation>
  <relation id="2" type="fdr" effectType="select">
    <target id="5" column="PseudoRows" parent_id="6" parent_name="RS-1" coordinate="[1,8,0],[1,25,0]" source="system"/>
    <source id="4" column="sal" parent_id="2" parent_name="scott.emp" coordinate="[3,7,0],[3,10,0]" clauseType="where"/>
  </relation>
</dlineage>
```

diagram



The fdr type dataflow is represented by a dash line. You can hide the `fdr` type dataflow by turn off the `impact` option in the SQLFlow.



References

1. xml code used in this article is generated by DataFlowAnalyzer tools
2. digram used in this article is generated by the SQLFlow Cloud version

fdr via from clause

From clause

If the resultset of a subquery or CTE is used in the from clause of the upper-level statement, then the impact of the lower level resultset will be transferred to the upper-level.

```
WITH
  cteReports (EmpID, FirstName, LastName, MgrID, EmpLevel)
  AS
  (
    SELECT EmployeeID, FirstName, LastName, ManagerID, 1  -- resultset1
    FROM Employees
    WHERE ManagerID IS NULL
  )
SELECT
  FirstName + ' ' + LastName AS FullName, EmpLevel  -- resultset2
FROM cteReports
```

In the CTE, there is an impact relation:

```
Employees.ManagerID -> fdr -> resultset1.pseudoRows
```

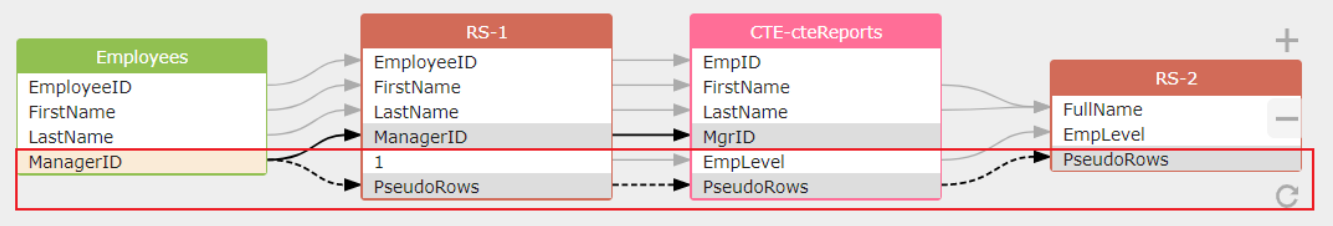
Since `cteReports` is used in the from clause of the upper-level statement, then the impact will carry on like this:

```
Employees.ManagerID -> fdr -> resultset1.pseudoRows -> fdd -> resultset2.pseudoRows
```

If we choose to ignore the intermediate resultset, the end to end dataflow is :

```
Employees.ManagerID -> fdr -> resultset2.pseudoRows
```

diagram



group by and aggregate function (fdr)

fdr and aggregate function

with group by clause

```
SELECT deptno, COUNT() num_emp, SUM(SAL) sal_sum
FROM scott.emp
Where city = 'NYC'
GROUP BY deptno
```

since `SUM()` is an aggregate function, so `deptno` column in the group by clause will be treated as an implict argument of the `SUM()` function.

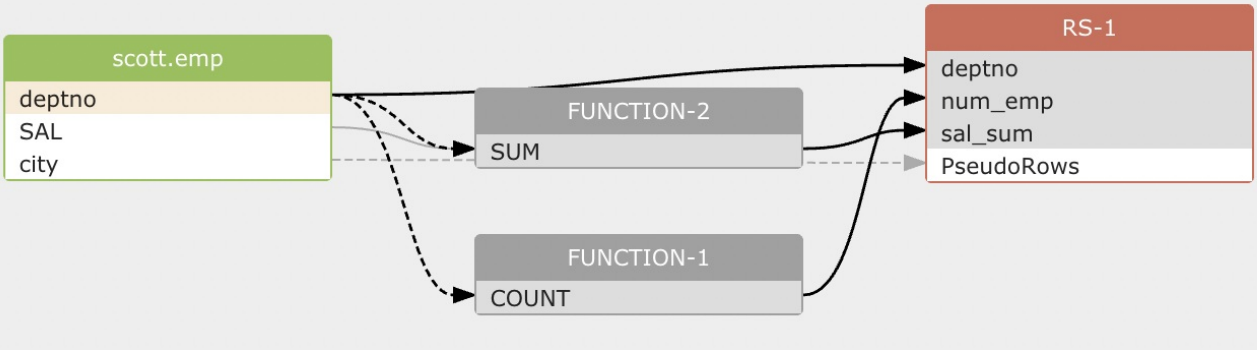
However, `deptno` column doesn't directly contribute the value to the `SUM()` function as column `SAL` does, So, the relation type is `fdr`:

```
scott.emp.deptno -> fdr -> SUM(SAL) -> fdd -> sal_sum
```

the columns in the having clause have the same relation as the columns in the group by clause as mentioned above.

The above rules apply to all aggregation functions, such as the `count()` function in the SQL.

diagram



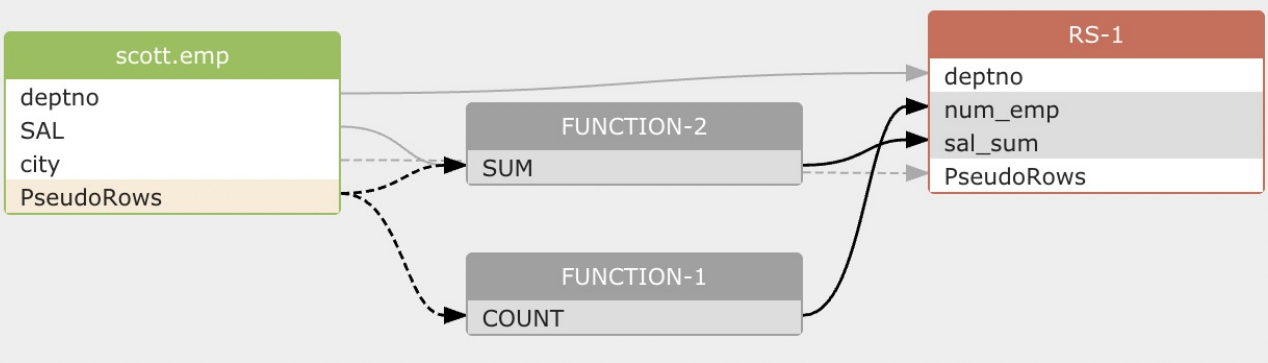
Without group by clause

If there is no group by clause but aggregate function used in the select like this:

```
SELECT deptno, COUNT() num_emp, SUM(SAL) sal_sum
FROM scott.emp
Where city = 'NYC'
```

This means all records in the table used as a group to the aggregate function, so we use PsedoRows as an impact argument of the aggregate function.

diagram



join condition (fdr)

join condition

```
select b.teur
from tbl a left join TT b on (a.key=b.key)
```

Columns in the join condition also effect the number of row in the resultset of the select list just like column in the where clause do.

So, the following relation will be discoverd in the above SQL.

```
tbl.key -> fdr -> resultset.PseudoRows
TT.key -> fdr -> resultset.PseudoRows
```

diagram