# 目录

# Data lineage model reference

Version 1.2.1 - 2021/7/12

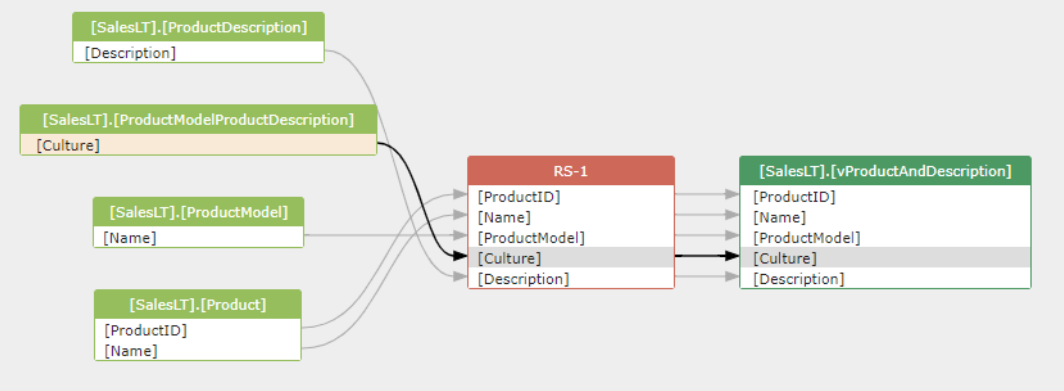


https://www.sqlparser.com

# lineage model summary

SQLFlow generates data lineage by analyzing SQL queries and stored procedures.

The entity in the data lineage model includes table, column, function, RESULTSET, relation and other entities . The combination of the entity and relation shows the lineage from one table/column to the others.

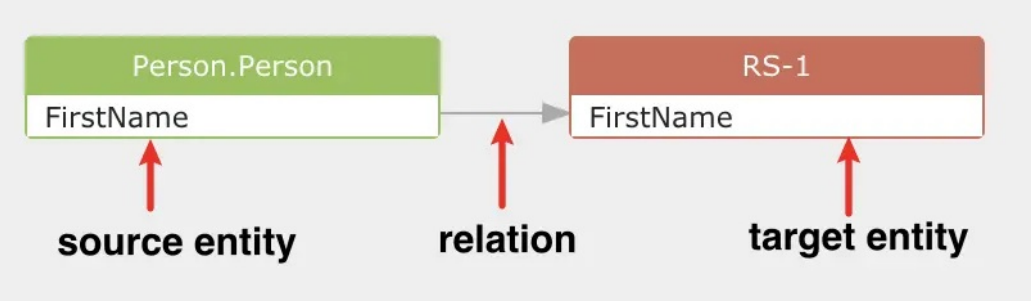



## An atom dataflow unit

An atom dataflow unit includes a source entity, a target entity and a relation between them.

```
SELECT p.FirstName
from Person.Person AS p
```

This is the dataflow generate for the above SQL query.

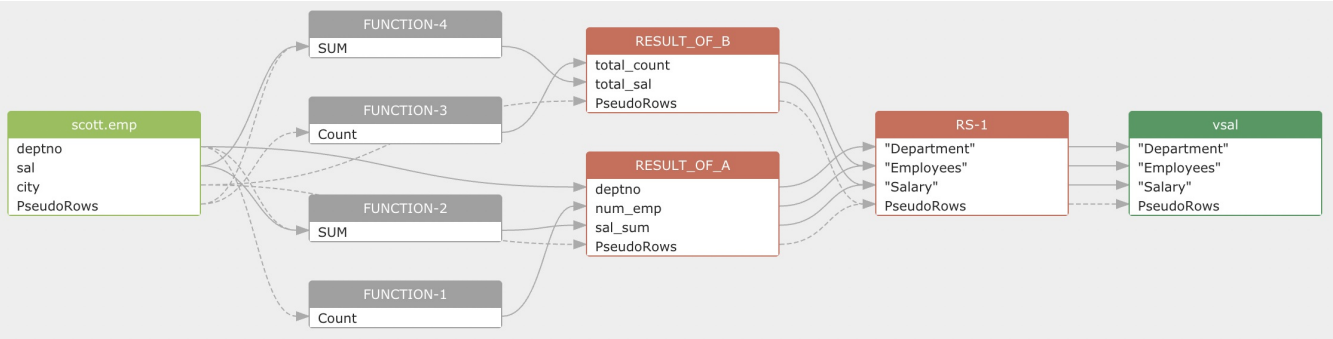



## Data lineage

A data lineage consist of lots of atom dataflow units.

```
CREATE VIEW vsal
AS
  SELECT a.deptno                "Department",
         a.num_emp / b.total_count "Employees",
         a.sal_sum / b.total_sal   "Salary"
  FROM   (SELECT deptno,
                 Count()  num_emp,
                 SUM(sal) sal_sum
          FROM   scott.emp
          WHERE  city = 'NYC'
          GROUP  BY deptno) a,
         (SELECT Count()  total_count,
                 SUM(sal) total_sal
          FROM   scott.emp
          WHERE  city = 'NYC') b
```

The data lineage diagram:




The output can be in XML or JSON format.

All the entities in the data lineage have predefined types. 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

# 3 types of data lineage models

In order to meet the user's various requirements about data lineage analysis, it is necessary to divide the SQLFlow data lineage model into several levels, each fitting a specific requirement.

## 1. The complete data lineage model

In this model, SQLFlow generates the data lineage includes all detailed information such as the RESULT SET generated during a SELECT statement, FUNCTION CALL used to calculate the new column value based on the input column, CASE EXPRESSION used to transform the data from one column to another, and so on.

This complete lineage model is the base of all other higher level lineage models which only includes some lineages in this complete model by omitting or aggregating some relations and entities in this model.

The higher level model is not only remove some relations and entities but also merge some relations to create a new relation. The most important entity introduced in the higher level model is PROCESS which is a SQL query/statement that do the transformation. The higher level model use the SQL query as the smallest unit to tells you how the data is transffered from one table/column to the other. On the other hand, the complete model tells you how data is transferred inside a SQL query.

When analyzing data lineage, the complete model is always generated since all other higher level models are based on this model. However, the complete model is not suitable to present to the user in a diagram if it includes too many entities and relations. The reason is:

1. Diagram includes thousands of entities and relations is a chaos and almost impossible to navigate in a single picture.
2. It's time comsuing and maybe impossible for the SQLFlow to layout the complete model with thousands of realtions.

The complete model is good when analyzing the SQL query or stored procedure less than 1000 thousands code lines. In this model, you can see all detailed information you need. For project includes thousands of stored procedures, It is much better to use the higher level model to visualize the dataflow for a specific table/column.

## 2. The column-level lineage model

As it name implied, this model traces the dataflow from column to column based on the SQL statement. In other words, from this model, you can see what SQL statement is used to move/impact data from one column to the other.

This model only includes 3 kinds of entity: the source column, the target column and the SQL statement( we call it PROCESS in the model) and the relation among them.

If you want to see how data is moved/impacted inside the SQL statement, you can use the complete model of this SQL statement to find more.

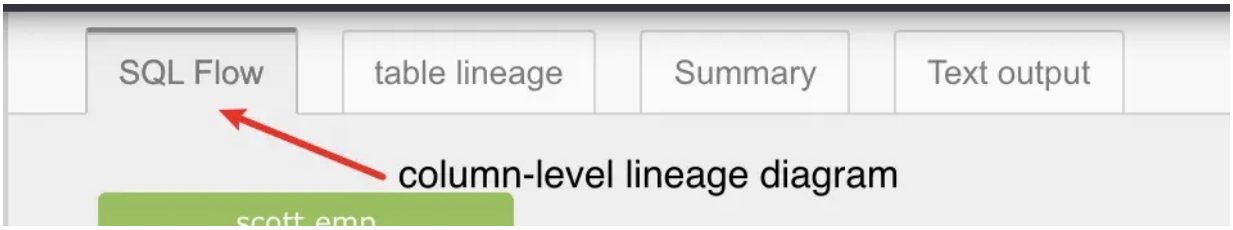
## 3. The table-level lineage model

As it name implied, this model traces the dataflow from table totable based on the SQL statement. In other words, from this model, you can see what SQL statement is used to move/impact data from onetable to the other.

This model only includes 3 kinds of entity: the source table, the target table and the SQL statement( we call it PROCESS in the model) and the relation among them.

If you want to see how data is moved inside the SQL statement, you can use the complete model of this SQL statement to find more.

## 4. SQLFlow UI



# the complete data lineage

When analyzing data lineage, the complete model is always generated since all other higher level models are based on this model.

## 1. Types of entity

Table, view, column, process(SQL statement), resultset, function, variale, path.

## 2. Types of relation

### fdd

The `fdd` relation means the data of the target entity comes from the source entity. Take this SQL query for example:

```
SELECT a.empName "eName"
FROM scott.emp a
```

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

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


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

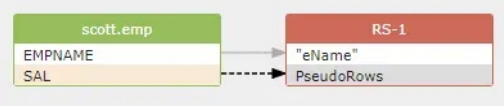
### fdr

The `fdr` relation means the data of the source column will impact the row number of the resultset in the select list, or will impact the result value of an anggreate function.

```
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:

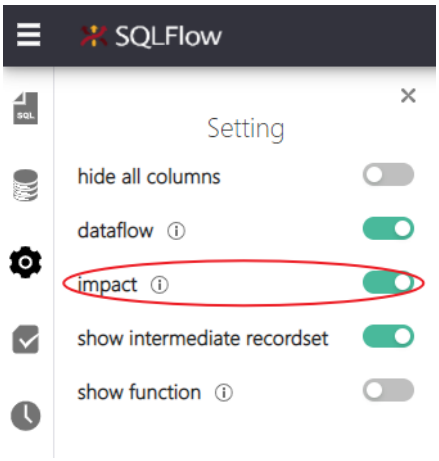
```
sal -> fdr -> resultset.PseudoRows
```




**PseudoRows column**

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

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.



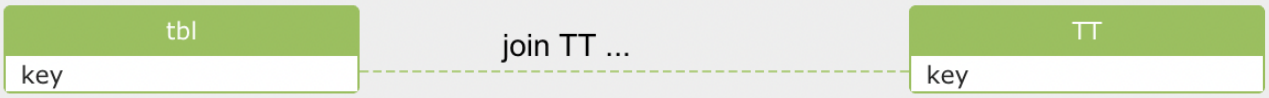

You may find more examples about the `fdr` relation in the ***lineage in real SQL*** section.

### join

The `join` relation build a link between 2 or more columns in the join condition. Stricly speaking, the relation is not a dataflow relation.

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

A join relation will be created after analzye the above SQL. It indicates a join relation betwee `tbl.key` and `TT.key`.




## 3. Connect the entity using relation

When build relation between 2 entities, the source and target entity can be column to column, or, table to table.

### column to column

This is the most often case that both entity in a relation are columns.

### table to table

Sometimes, there will be a relation between 2 tables. For example, in an alter table rename SQL statement, a table to table relation will be created. Acutally, a table to table relation is represented by a column to column relation using the `PseudoRows` column.

Table to table relation is included in the complete lineage model by using `PseudoRows` for 2 reasons:

1. This pseudo column to column relation will be used to generate a table-level lineage later if user need a table-level lineage model.
2. If a column in this table is used in a column to column relation while the table itself is in a table to table relation, then, this pseudo column will make it possible for a single table to includes both the column to column relation and table to table relation.

take this SQL for example

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

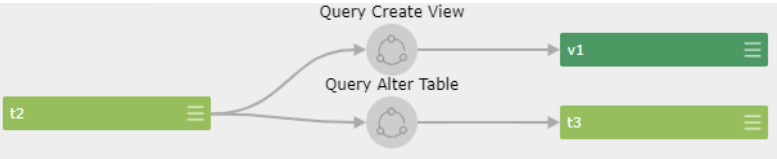
**column to column relation**

the complete data lineage


As you can see, Table `t2` involved in the column to column relation generated by the create view statement, It also involved in a table to table relation generated by the alter table statement. A single table `t2` in the above digram show that it includes both the column to column relation and the table to table reation.

#### table-level lineage

With the table to table relation generated in the complete data lineage model, we can later use it to generate a table-level lineage like this:



# the column-level lineage

1. How to get column-level model from the complete model
2. The export format of the column-level model

# the table-level lineage

The table-level lineage provides a table 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.

## How to get table-level model from the complete lineage model

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

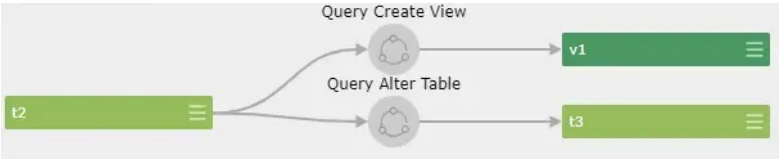
1. The table id and process id in the table-level model is the same as the one in complete lineage model.
2. The new table-level model uses table and process element from the complete lineage model and generate the new relation between the table and process.
3. Iterate target and source table in the complete lineage 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;
```

### The complete data 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>
```

### The 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>
```

## The export format of the table-level model

## SQLFlow UI

# data lineage in multi queries

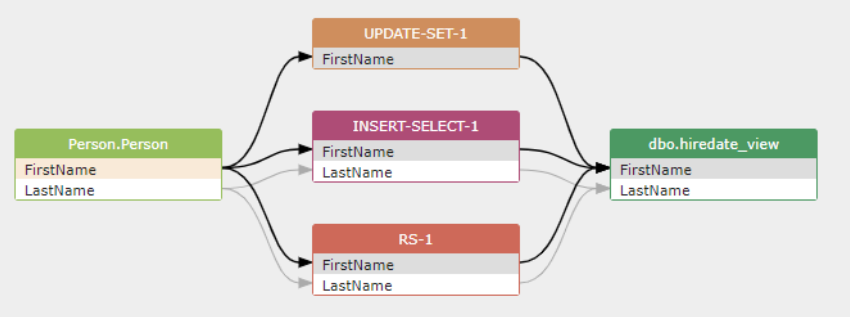
The same column in different SQL statements can create different type column-level lineage. Those lineages should be picked up separately.

```
CREATE VIEW dbo.hiredate_view(FirstName,LastName)
AS
SELECT p.FirstName, p.LastName
from Person.Person AS p
GO

update dbo.hiredate_view h
set  h.FirstName =  p.FirstName
from h join Person.Person p
on h.id = p.id;

insert into  dbo.hiredate_view (FirstName,LastName)
SELECT p.FirstName, p.LastName
from Person.Person AS p ;
```

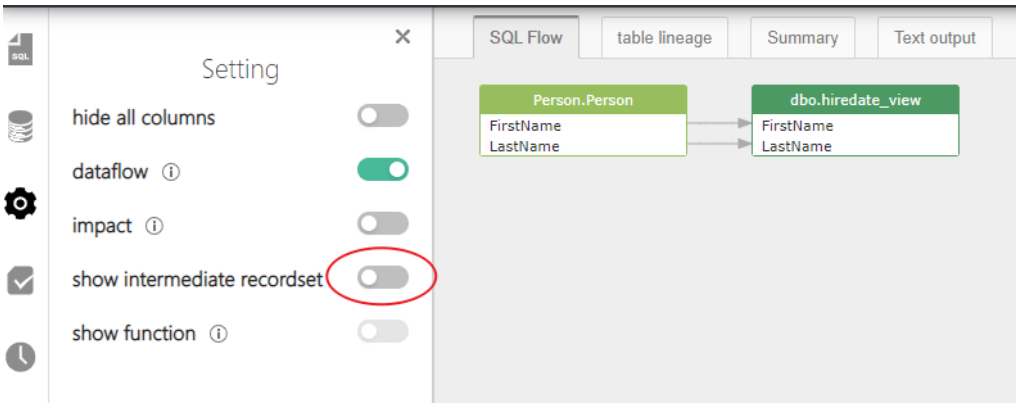
## column to column relations

As you can see, the column: `FirstName` involves in the three SQL statements: create view, update and insert statement.

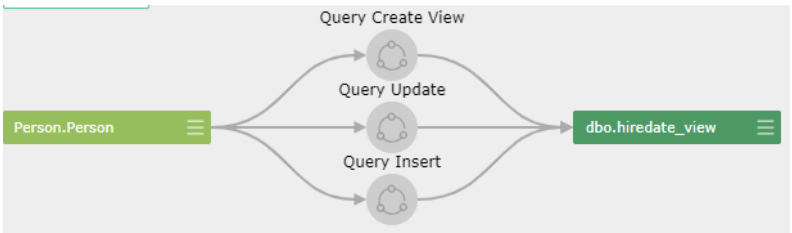
While the column `LastName` involves in the two SQL statement: create view, insert statement.




**In the complete lineage mode**, if we turn off the `show intermediate recordset` option, you may find that although it gives you a higher level overview of the table to table relation, but some SQL statement related information such as how one column impact another column are missing.




If we **check lineage in the table-level** via `table lineage` tab, you may find diagram like this:

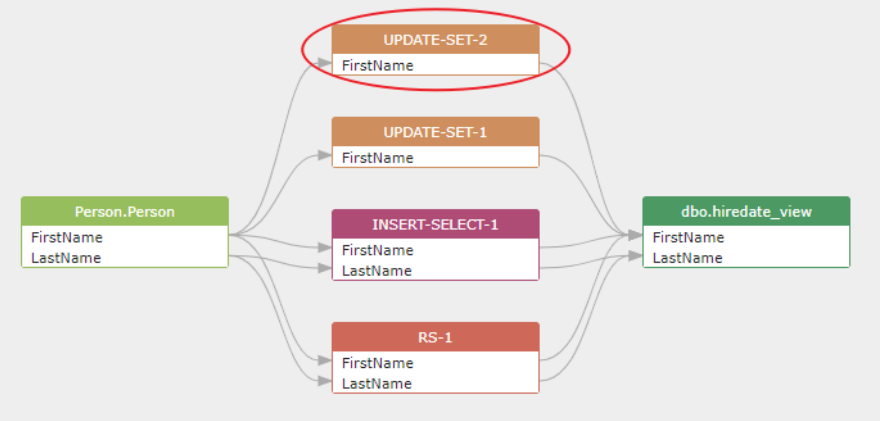


You can see that the statements that involved in the data transformation is persisted, but of course, since it's a table-level lineage, the columns involved in the lineage are hidden. So, it's your choice to use what's kind level of the lineage based on your requirements.

## duplicated SQL query

```
CREATE VIEW dbo.hiredate_view(FirstName,LastName)
AS
SELECT p.FirstName, p.LastName
from Person.Person AS p
GO

update dbo.hiredate_view h
set  h.FirstName =  p.FirstName
from h join Person.Person p
on h.id = p.id;

insert into  dbo.hiredate_view (FirstName,LastName)
SELECT p.FirstName, p.LastName
from Person.Person AS p ;

update dbo.hiredate_view h
set  h.FirstName =  p.FirstName
from h join Person.Person p
on h.id = p.id;
```

If the update statement is executed twice in the SQL batch as illustrated above, then you will see the update column-level lineage is showing twice in the diagram. These may not we want to see.




## how to avoid duplicate column-level lineage

# 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 SQL script index of task. SqlInfoHelper uses the third number to position SQL script.

### 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": "hashId",
      "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` .

### hashId

The hashId is generated by calcualting the MD5 value of string: `type+name of columns` in the resultset. The hashId of a resultset is used when export a column-level lineage, it is used as a table name of the column in this resultset.

# 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": "processId",
      "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`, `join`.

Please check dbobjects_relationship for the detailed information.

### effectType

This is the SQL statement that generate this relation. Available values: `select, insert, update, merge_update, merge_insert, create_view, create_table, merge, delete, function, rename_table, swap_table, like_table, cursor, trigger, create_view`

### processId

This is the SQL query that build this relation.

### queryHashId

Use `processId` instead.

~~This is the hash code of the SQL query text from which this relation is generated. The `queryHashId` combined with target and source columns can be used to determine a unique relation in the lineage model. It's useful when export the lineage into the data catalog such as the Apache Atals to avoid the duplicated relation been inserted.~~

~~The SQL query with the same queryHashId is treated as the same query. This is usually happened when a SQL query been executed multi times.~~

## 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": "name",
      "typeName": "string",
      "isOptional": false
    },
    {
      "name": "type",
      "typeName": "string",
      "isOptional": false
    },
    {
      "name": "queryHashId",
      "typeName": "string",
      "isOptional": false
    },
    {
      "name": "procedureName",
      "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:

- Create Table
- Create External Table
- Create View
- Create Stage
- Alter Table
- Update
- Merge
- Insert

- Select Into
- Hive Load

### queryHashId

This is the MD5 hash id that uniquely identify this SQL query. This `queryHashId` will be used when update a column or table-level lineage in the Atlas or other data catalog.

### procedureName

If this query statement is inside a stored procedure, this `procedureName` is the fully qualified name of the stored procedure. Otherwise, the `procedureName` should always be the `batchQueries`

# 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": "name",
      "typeName": "string",
      "isOptional": false
    },
    {
      "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": "name",
      "typeName": "string",
      "isOptional": false
    },
    {
      "name": "type",
      "typeName": "string",
      "isOptional": false
    },
    {
      "name": "uri",
      "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

the name of the path.

### type

type of the path, one of `hdfs`, Amazon `s3`, BigQuery `GS`

### uri

the path where the object is stored.

### columns

Path doesn't has columns in fact. We add columns here in order to make path available in column-level lineage model by using the pseudo column.

# Lineage model elements on UI

lineage model elements on UI

## Entity

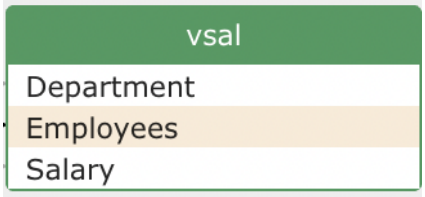
path in the json: `data->sqlflow->dbobjs`

### 1. Permanent entity

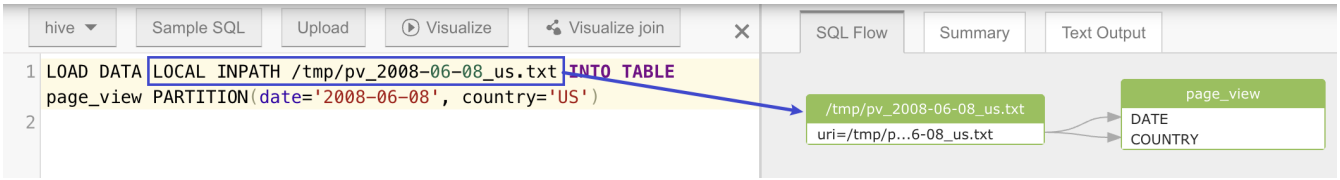
#### 1. table


#### 2. external table

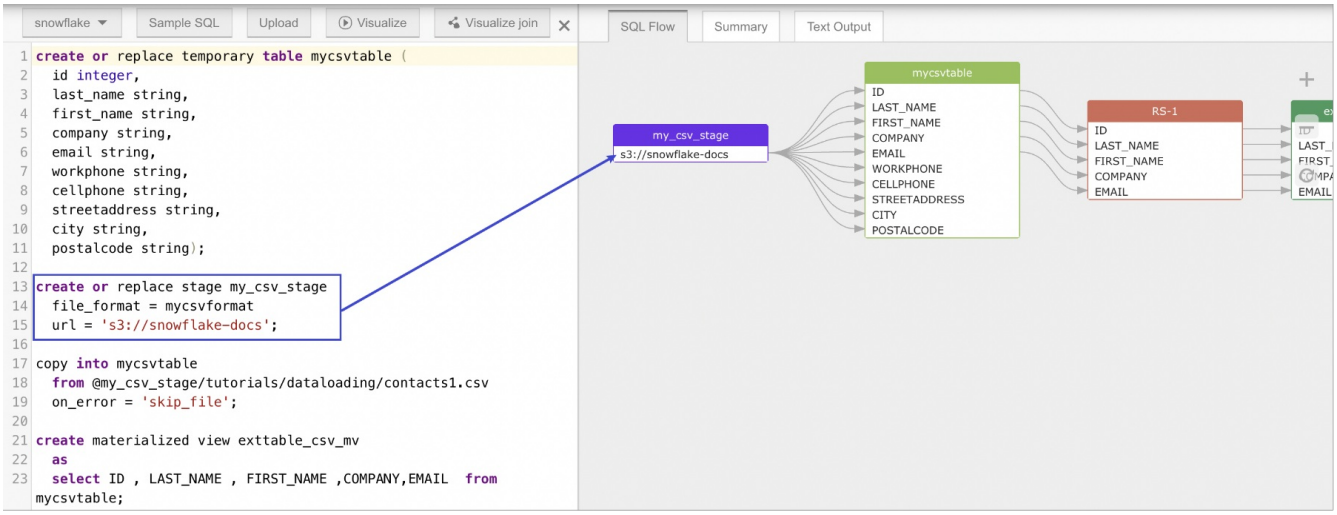
#### 3. view




#### 4. hive local directory/inpath

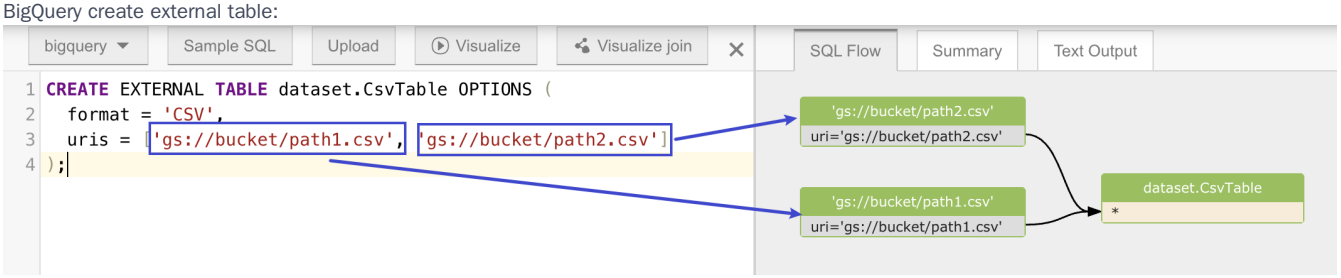



#### 5. snowflake stage



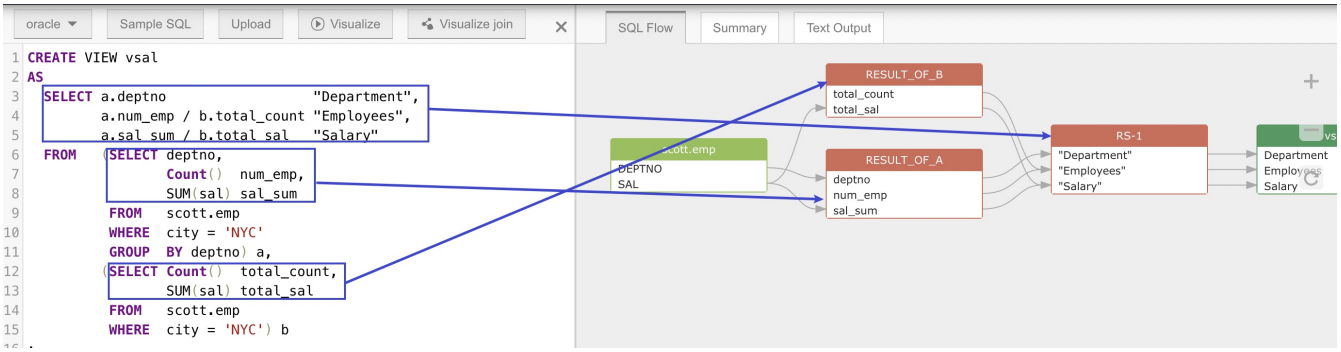

#### 6. bigquery file uri

BigQuery create external table:

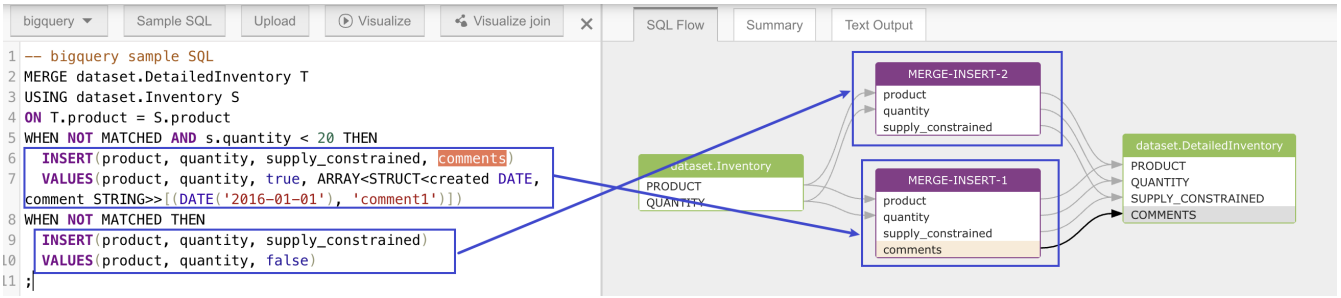



## 2. temporary entity

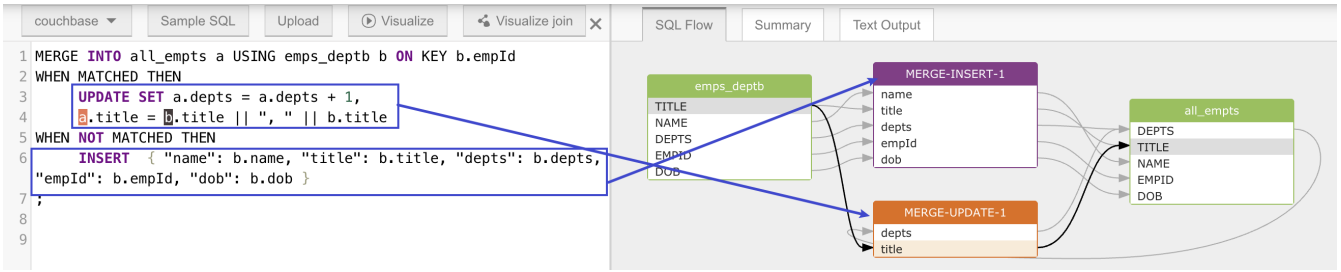
### 1. select_list



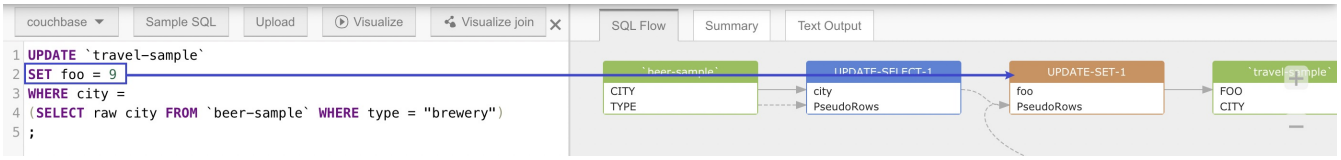

### 2. merge_insert



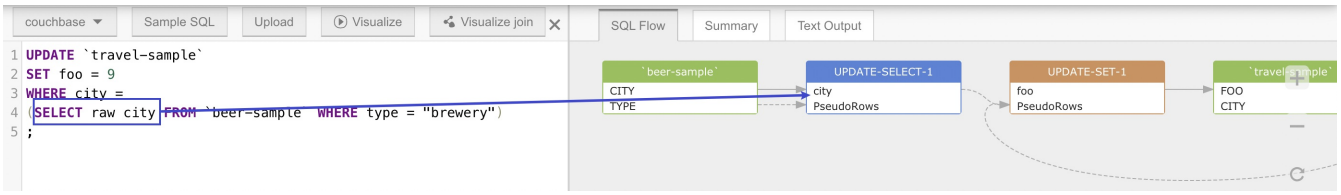

### 3. merge_update



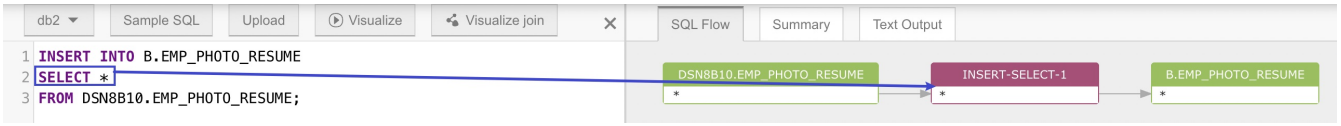

### 4. update_set




### 5. update-select

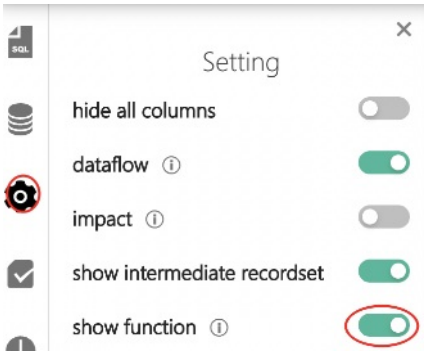



### 6. insert-select



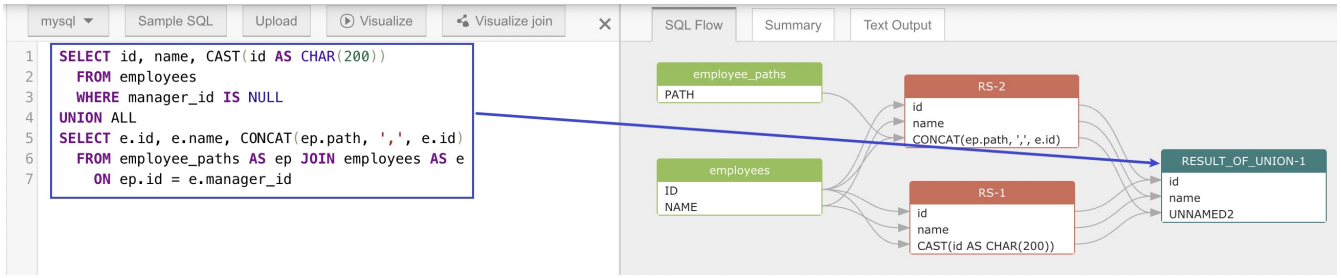

### 7. function

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

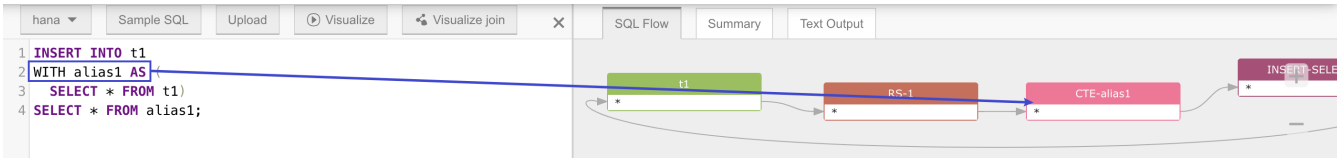




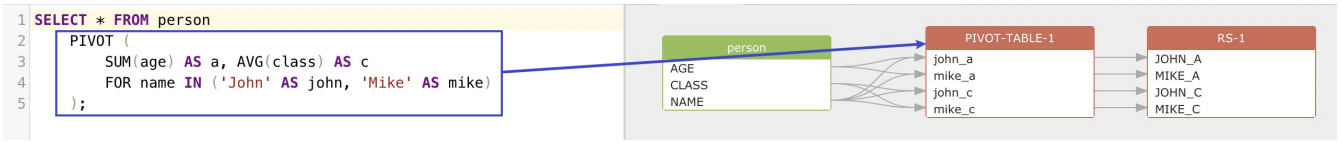
### 8. union



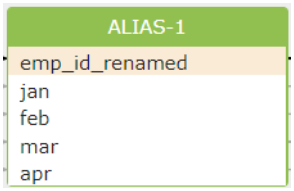

### 9. cte



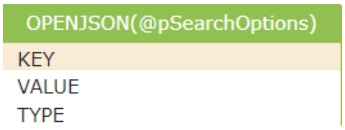

### 10. pivot table



11. snowflake pivot alias




12. mssql open json

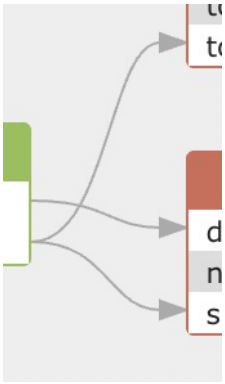



13. mssql json property


## relationship

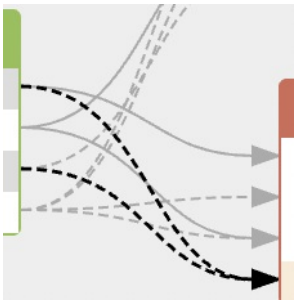
path in the json: `data->sqlflow->relations`

1. fdd, data flow



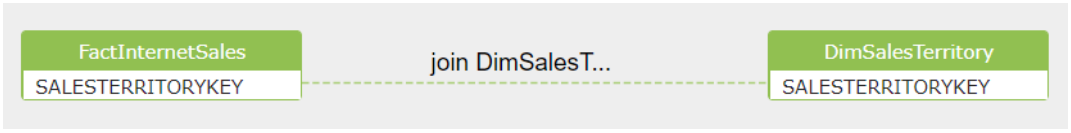
2. fdr, frd data impact

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.

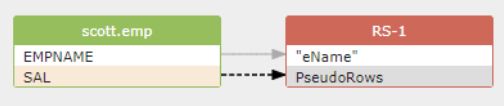
## 1. represents the total number of columns in a table/resultset

```
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;
```

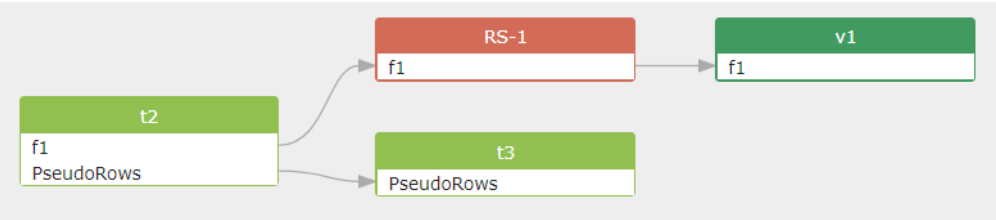
The first create view statement will generate a column-level lineage of the table `t2`,

```
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

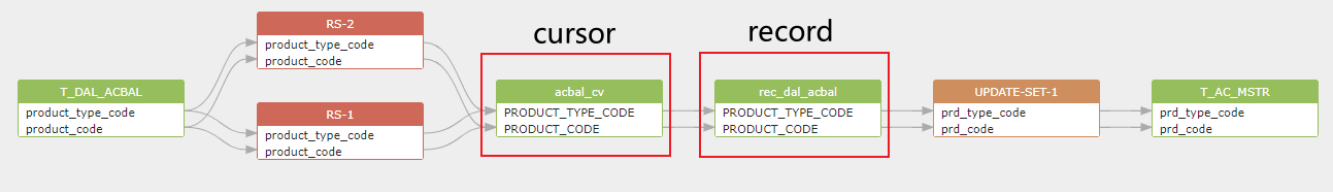
This is an Oracle PLSQL.

```
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

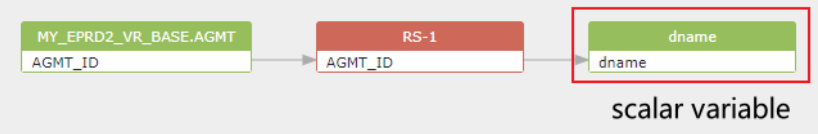
This is a Teradata stored procedure

```
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;
```

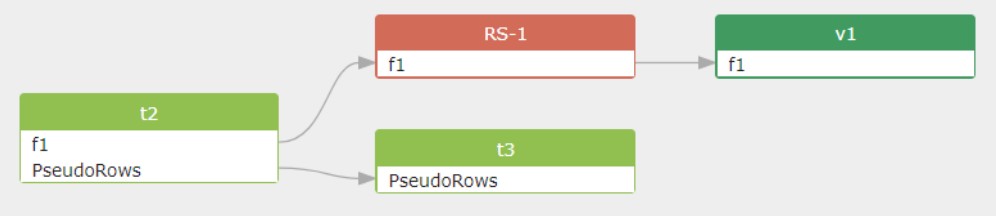
### column-level lineage mode

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.



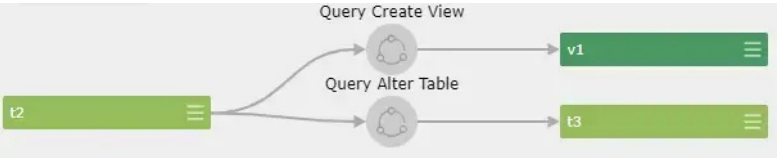

### table-level lineage mode

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

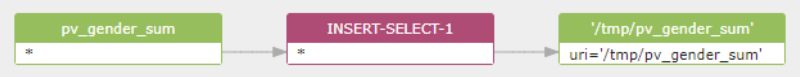
The data flow is:

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

#### dataflow in xml

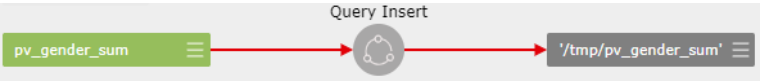
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<dlineage>
   <path id="2" name="'/tmp/pv_gender_sum'" uri="'/tmp/pv_gender_sum'" type="path" processIds="3" coordinate="[1,34,0],[1,54,0]">
      <column id="4" name="uri='/tmp/pv_gender_sum'" coordinate="[-1,-1,0],[-1,-1,0]"/>
   </path>
   <process id="3" name="Query Insert" type="Insert" coordinate="[1,1,0],[3,20,0]"/>
   <table id="6" name="pv_gender_sum" type="table" coordinate="[3,6,0],[3,19,0]">
      <column id="7" name="*" coordinate="[2,8,0],[2,23,0]"/>
   </table>
   <resultset id="9" name="INSERT-SELECT-1" type="insert-select" coordinate="[2,8,0],[2,23,0]">
      <column id="10" name="*" coordinate="[2,8,0],[2,23,0]"/>
   </resultset>
   <relation id="1" type="fdd" effectType="select">
      <target id="10" column="*" parent_id="9" parent_name="INSERT-SELECT-1" coordinate="[2,8,0],[2,23,0]"/>
      <source id="7" column="*" parent_id="6" parent_name="pv_gender_sum" coordinate="[2,8,0],[2,23,0]"/>
   </relation>
   <relation id="2" type="fdd" effectType="insert">
      <target id="4" column="uri='/tmp/pv_gender_sum'" parent_id="2" parent_name="'/tmp/pv_gender_sum'" coordinate="[-1,-1,0],[-1,-1,0]"/>
      <source id="10" column="*" parent_id="9" parent_name="INSERT-SELECT-1" coordinate="[2,8,0],[2,23,0]"/>
   </relation>
</dlineage>
```

#### 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)
)
```

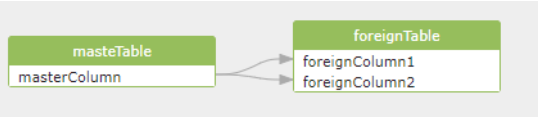
The data flow is:

```
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)

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);
```

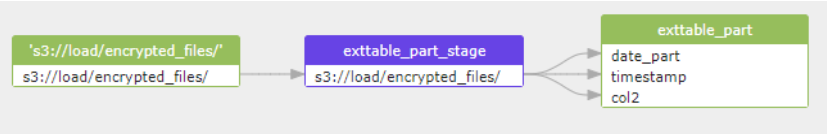
The data of the external table `exttable_part` comes from the `path ('s3://load/encrypted_files/')` via the stage: `exttable_part_stage`

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

#### dataflow in xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<dlineage>
   <stage id="5" name="exttable_part_stage" type="stage" processIds="6" coordinate="[1,25,0],[1,44,0]">
      <column id="7" name="s3://load/encrypted_files/" coordinate="[-1,-1,0],[-1,-1,0]"/>
   </stage>
   <path id="2" name="'s3://load/encrypted_files/'" uri="'s3://load/encrypted_files/'" type="path" coordinate="[-1,-1,0],[-1,-1,0]">
      <column id="3" name="s3://load/encrypted_files/" coordinate="[-1,-1,0],[-1,-1,0]"/>
   </path>
   <process id="6" name="Query Create Stage" type="Create Stage" coordinate="[1,1,0],[4,58,0]"/>
   <process id="13" name="Query Create External Table" type="Create External Table" coordinate="[6,1,0],[15,33,0]"/>
   <table id="9" name="exttable_part" type="table" processIds="13" coordinate="[6,23,0],[6,36,0]">
      <column id="10" name="date_part" coordinate="[7,2,0],[7,11,0]"/>
      <column id="11" name="timestamp" coordinate="[10,2,0],[10,11,0]"/>
      <column id="12" name="col2" coordinate="[11,2,0],[11,6,0]"/>
   </table>
   <relation id="1" type="fdd">
      <target id="7" column="'s3://load/encrypted_files/'" parent_id="5" parent_name="exttable_part_stage" coordinate="[-1,-1,0],[-1,-1,0]"/>
      <source id="3" column="'s3://load/encrypted_files/'" parent_id="2" parent_name="'s3://load/encrypted_files/'" coordinate="[-1,-1,0],[-1,-1,0
   </relation>
   <relation id="2" type="fdd">
      <target id="10" column="date_part" parent_id="9" parent_name="exttable_part" coordinate="[7,2,0],[7,11,0]"/>
      <source id="7" column="'s3://load/encrypted_files/'" parent_id="5" parent_name="exttable_part_stage" coordinate="[-1,-1,0],[-1,-1,0]"/>
   </relation>
   <relation id="3" type="fdd">
      <target id="11" column="timestamp" parent_id="9" parent_name="exttable_part" coordinate="[10,2,0],[10,11,0]"/>
      <source id="7" column="'s3://load/encrypted_files/'" parent_id="5" parent_name="exttable_part_stage" coordinate="[-1,-1,0],[-1,-1,0]"/>
   </relation>
   <relation id="4" type="fdd">
      <target id="12" column="col2" parent_id="9" parent_name="exttable_part" coordinate="[11,2,0],[11,6,0]"/>
      <source id="7" column="'s3://load/encrypted_files/'" parent_id="5" parent_name="exttable_part_stage" coordinate="[-1,-1,0],[-1,-1,0]"/>
   </relation>
</dlineage>
```

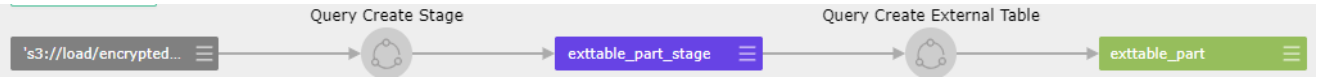
#### diagam




#### 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']
);
```

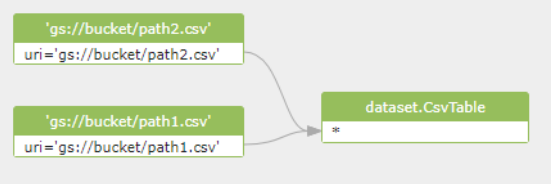
The data of the external table `dataset.CsvTable` comes from the csv file: `gs://bucket/path1.csv, gs://bucket/path2.csv`

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

#### dataflow in xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<dlineage>
   <path id="6" name="'gs://bucket/path1.csv'" uri="'gs://bucket/path1.csv'" type="path" fileFormat="CSV" coordinate="[-1,-1,0],[-1,-1,0]">
      <column id="7" name="uri='gs://bucket/path1.csv'" coordinate="[-1,-1,0],[-1,-1,0]"/>
   </path>
   <path id="9" name="'gs://bucket/path2.csv'" uri="'gs://bucket/path2.csv'" type="path" fileFormat="CSV" coordinate="[-1,-1,0],[-1,-1,0]">
      <column id="10" name="uri='gs://bucket/path2.csv'" coordinate="[-1,-1,0],[-1,-1,0]"/>
   </path>
   <process id="3" name="Query Create External Table" type="Create External Table" coordinate="[1,1,0],[4,3,0]"/>
   <table id="2" schema="dataset" name="dataset.CsvTable" type="table" processIds="3" coordinate="[1,23,0],[1,39,0]">
      <column id="4" name="*" coordinate="[1,1,0],[1,2,0]"/>
      <column id="4_0" name="URI='GS://BUCKET/PATH1.CSV'" coordinate="[1,1,0],[1,2,0]"/>
      <column id="4_1" name="URI='GS://BUCKET/PATH2.CSV'" coordinate="[1,1,0],[1,2,0]"/>
   </table>
   <relation id="1" type="fdd">
      <target id="4" column="*" parent_id="2" parent_name="dataset.CsvTable" coordinate="[1,1,0],[1,2,0]"/>
      <source id="7" column="uri='gs://bucket/path1.csv'" parent_id="6" parent_name="'gs://bucket/path1.csv'" coordinate="[-1,-1,0],[-1,-1,0]"/
   </relation>
   <relation id="2" type="fdd">
      <target id="4" column="*" parent_id="2" parent_name="dataset.CsvTable" coordinate="[1,1,0],[1,2,0]"/>
      <source id="10" column="uri='gs://bucket/path2.csv'" parent_id="9" parent_name="'gs://bucket/path2.csv'" coordinate="[-1,-1,0],[-1,-1,0]'
   </relation>
</dlineage>
```

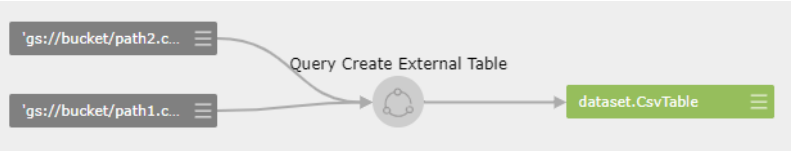
#### diagam



#### 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')
```

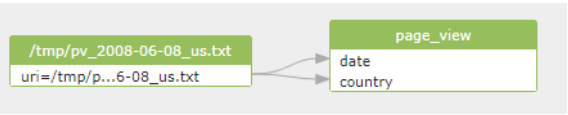
The data flow is:

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

#### dataflow in xml

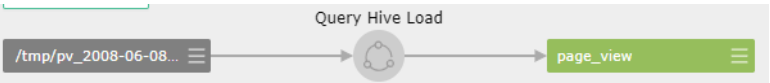
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<dlineage>
   <path id="2" name="/tmp/pv_2008-06-08_us.txt " uri="/tmp/pv_2008-06-08_us.txt " type="path" coordinate="[1,24,0],[1,50,0]">
      <column id="3" name="uri=/tmp/pv_2008-06-08_us.txt " coordinate="[-1,-1,0],[-1,-1,0]"/>
   </path>
   <process id="6" name="Query Hive Load" type="Hive Load" coordinate="[1,1,0],[1,113,0]"/>
   <table id="5" name="page_view" type="table" processIds="6" coordinate="[1,61,0],[1,113,0]">
      <column id="7" name="date" coordinate="[1,81,0],[1,85,0]"/>
      <column id="8" name="country" coordinate="[1,100,0],[1,107,0]"/>
   </table>
   <relation id="1" type="fdd">
      <target id="7" column="date" parent_id="5" parent_name="page_view" coordinate="[1,81,0],[1,85,0]"/>
      <source id="3" column="uri=/tmp/pv_2008-06-08_us.txt " parent_id="2" parent_name="/tmp/pv_2008-06-08_us.txt " coordinate="[-1,-1,0]
   </relation>
   <relation id="2" type="fdd">
      <target id="8" column="country" parent_id="5" parent_name="page_view" coordinate="[1,100,0],[1,107,0]"/>
      <source id="3" column="uri=/tmp/pv_2008-06-08_us.txt " parent_id="2" parent_name="/tmp/pv_2008-06-08_us.txt " coordinate="[-1,-1,0]
   </relation>
</dlineage>
```

#### diagram




#### table-level lineage

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



# case expression (fdd)

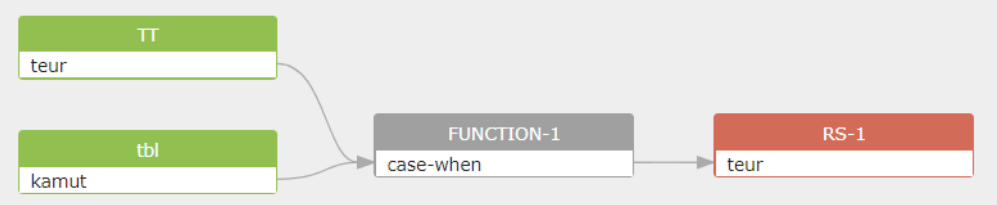
## case expression

```
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

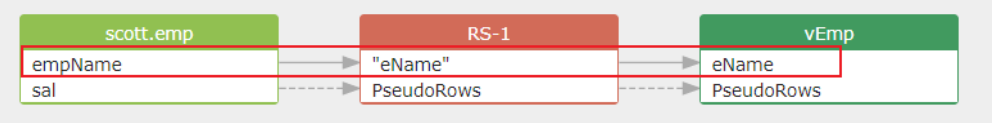
```
create view vEmp(eName) as
SELECT a.empName "eName"
FROM scott.emp a
Where sal > 1000
```

### 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

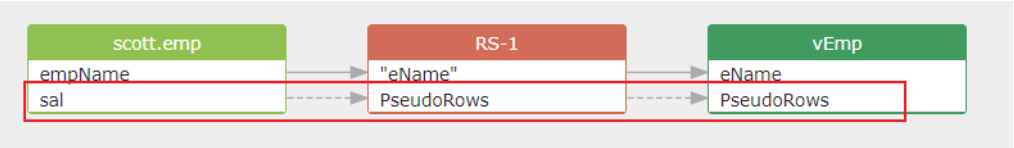
From this query, you will see how the column `sal` in where clause impact the number of rows in the top level view `vEmp`.

```
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 :

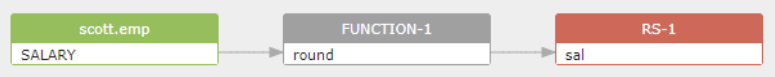
```
scott.emp.salary -> fdd -> round(salary) -> fdd -> sal
```

#### 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
```

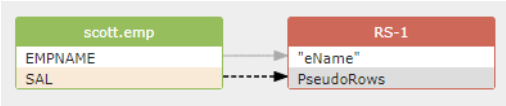
### PseudoRows column

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

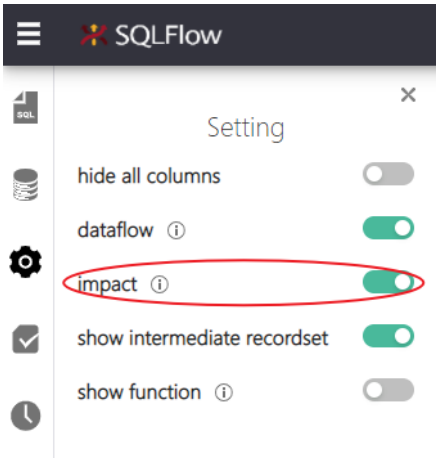
#### dataflow in xml

```
<?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
```

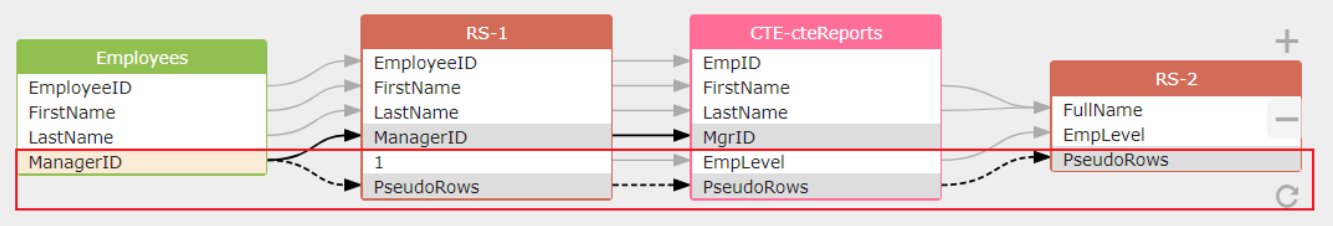
Since `cteReports` is used in the from clause of the upper-level statement, then the impact will carry on like this:

```
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.

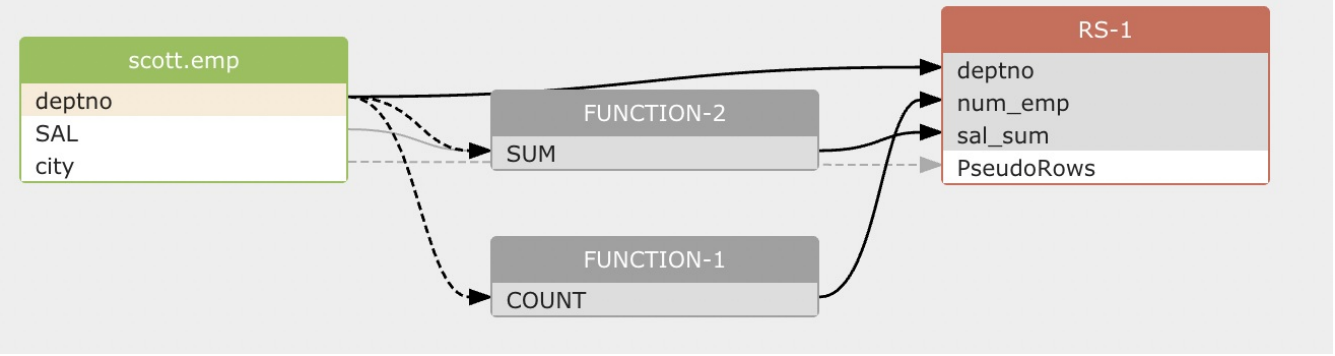
However, `deptno` column doesn't directly contribute the value to the `SUM()` function as column `SAL` does, So, the relation type is `fdr`:

```
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**



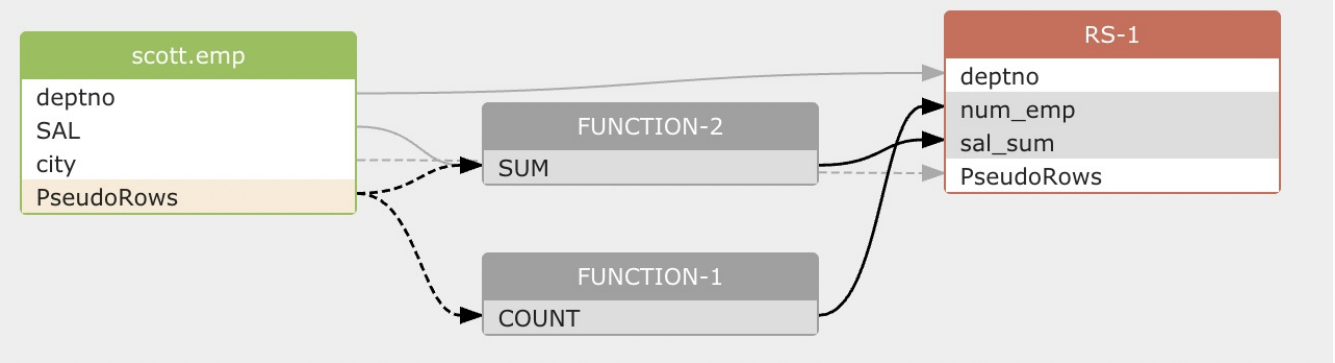

### Without group by clause

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)

### fdr relation

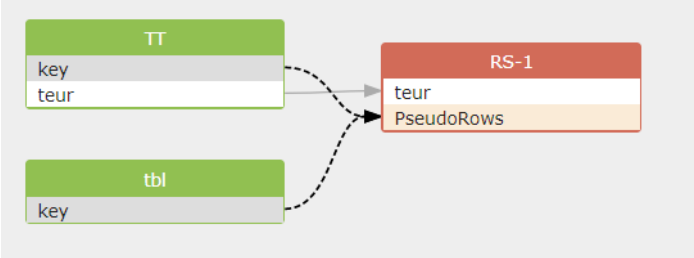
```
select b.teur
from tbl a left join TT b on (a.key=b.key)
```

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
```

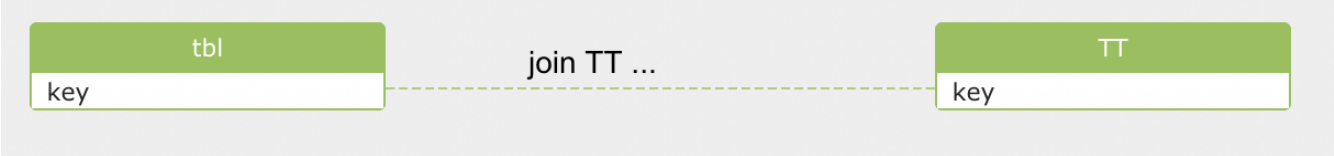
#### diagram




### join relation

A join relation will be created after analzye the above SQL. It indicates a join relation betwee `tbl.key` and `TT.key`.

#### diagram



# Export metadata and lineage

## export metadata

The source of metadata comes from 2 sources, one is extracted from the database, the other is collect from the SQL scripts.

In order to export the metadata collected by the SQLFlow, we need to know the structure that how SQLFlow save the

metadata in the data lineage model.

### 1. The database objects that need to be exported

1. cluster
2. db
3. table/view
4. column
5. process, this is usually the query that transform the data, such as a stored proceudure, an insert statement and etc.

#### defualt value for db and schema name

If a db or schema is not mentioned in a SQL query, we use `default` as the name of the missing db or schema.

The default schema for SQL Server database is `dbo`.

### metadata from the database

SQLFlow use the grabit tool to extract metadata from a database instance and save it in the format defined in this document.

https://e.gitee.com/gudusoft/docs/591884/file/1434789?sub_id=4091727

#### cluster

The related element in the exported json file is: `physicalInstance`

#### db

The related element in the exported json file is: `databases`

#### table/view

The related element in the exported json file is: `databases->tables`

#### column

The related element in the exported json file is: `databases->tables->column`

### 2. metadata from the SQL script

There is no `cluster` and `db` information in the json file generated by the SQLFlow after analyzing the SQL script and stored procedure.

The database objects is saved in a json array `dbobjs` with the `name` and `type` property.


```
{
 "dbvendor": "dbvoracle",
 "dbobjs": [
  {
   "id": "37",
   "name": "Query Create View",
   "type": "process"
  },
  {
   "id": "4",
   "schema": "scott",
   "name": "scott.emp",
   "type": "table",
   "columns": [
```

```
      {
       "id": "41",
       "name": "deptno"
      },
      {
       "id": "42",
       "name": "sal"
      },
      {
       "id": "43",
       "name": "city"
      }
     ]
    }
   ]
}
```

#### table/view

table/view can be located via `dbobjs[index]` with the `type` set to `table`.

#### column

column can be located via `dbobjs[index]->columns[index]`

### 3. uniquely identify a database object

#### cluster

1. Hive, the default cluster name is `primary`
2. Oracle, the cluster name is `physicalInstance`
3. SQL Server, the cluster name is the `servername`

#### db

The unique name of a database is in syntax like: `dbname@cluster`. Such as `ABCDPROD@xzy001db03.ddc.nba.com`

#### table/view

The unique name of a table is `dbname.tablename@cluster`, or `dbname.schemaname.tablename@cluster`.

Such as `ABCDPROD.HARDWARE.SUBRACK_I_TRG@xzy001db03.ddc.nba.com`

#### column

The unique name of a column is `dbname.tablename.columnname@cluster`, or `dbname.schemaname.tablename.columnname@cluster`.

#### process

The unique name of a process is `dbname.procedureName.queryHashId@cluster`.

The `proedureName` should be fully qualified. If the SQL query is not inside a stored proceure, then `procedureName` will use `batchQueries` as it name. `queryHashId` is the hash code of the SQL query that do the transformation, The `queryHashId` can be generated by calling the GSP library via `TParseTreeNode.getMd5()` method.

# export table-level lineage

## table-level lineage

The exported table-level linege should be in format like this:

```
source_db;source_schema;source_table;target_db;target_schema;target_table;procedure_names;query_hash_id
```

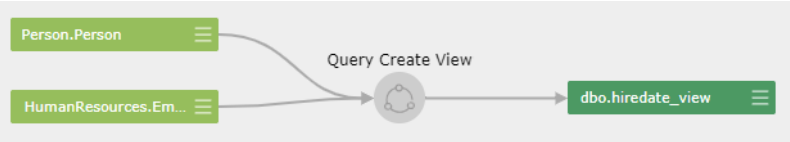
### sample sql

```
CREATE VIEW dbo.hiredate_view
AS
SELECT p.FirstName, p.LastName, e.BusinessEntityID, e.HireDate
FROM HumanResources.Employee e
JOIN Person.Person AS p ON e.BusinessEntityID = p.BusinessEntityID ;
```

### lineage in XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<dlineage>
   <process id="19" name="Query Create View" type="Create View" coordinate="[1,1,0],[5,69,0]"/>
   <table id="2" schema="HumanResources" name="HumanResources.Employee" alias="e" type="table" coordinate="[4,6,0],[4,31,0]"/>
   <table id="7" schema="Person" name="Person.Person" alias="p" type="table" coordinate="[5,6,0],[5,24,0]"/>
   <view id="18" schema="dbo" name="dbo.hiredate_view" type="view" processIds="19" coordinate="[1,13,0],[1,30,0]"/>
   <relation id="1007" type="fdd">
      <target id="1008" target_id="19" target_name="Query Create View"/>
      <source id="1002" source_id="7" source_name="Person.Person"/>
   </relation>
   <relation id="1009" type="fdd">
      <target id="1001" target_id="18" target_name="dbo.hiredate_view"/>
      <source id="1010" source_id="19" source_name="Query Create View"/>
   </relation>
   <relation id="1011" type="fdd">
      <target id="1012" target_id="19" target_name="Query Create View"/>
      <source id="1005" source_id="2" source_name="HumanResources.Employee"/>
   </relation>
   <relation id="1013" type="fdd">
      <target id="1004" target_id="18" target_name="dbo.hiredate_view"/>
      <source id="1014" source_id="19" source_name="Query Create View"/>
   </relation>
</dlineage>
```

### diagram




### exported lineage

```
source_db;source_schema;source_table;target_db;target_schema;target_table;procedure_names;query_hash_id
default;person;person;default;dbo;hiredate_view;batchQueries;xxxxx
default;HumanResources;Employee;default;dbo;hiredate_view;batchQueries;xxxxx
```

The table name in the exported lineage **shoudn't be qualified** , it must be like this `Employee` . But when it is written to the data catalog such as Atlas, it must be qualified like this: `default.HumanResources.Employee`

# export column-level lineage

## column-level lineage

The exported column-level linege should be in format like this:

```
source_db;source_schema;source_table;source_column;target_db;target_schema;target_table;target_column;procedure_names;query_hash_id
```

the exported column-level lineage **shouldn't include any intermediate recordset**, it only inclues the source and target table column, the hasd id of the query which does this transformation.

### sample sql

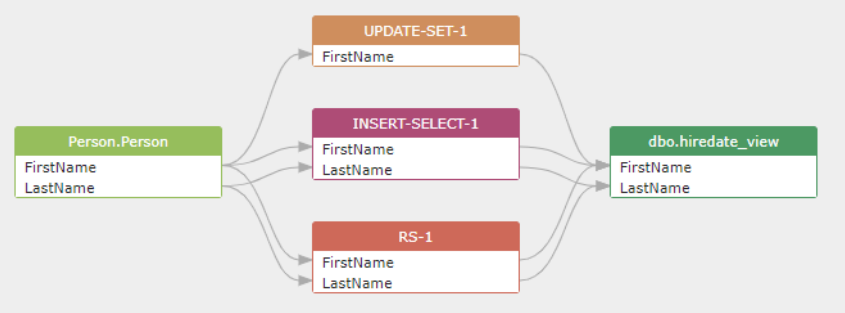
```
CREATE VIEW dbo.hiredate_view(FirstName,LastName)
AS
SELECT p.FirstName, p.LastName
from Person.Person AS p
GO

update dbo.hiredate_view h
set  h.FirstName =  p.FirstName
from h join Person.Person p
on h.id = p.id;

insert into  dbo.hiredate_view (FirstName,LastName)
SELECT p.FirstName, p.LastName
from Person.Person AS p ;
```

### column-level lineage in xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<dlineage>
   <process id="11" name="Query Create View" type="Create View" coordinate="[1,1,0],[4,27,0]"/>
   <process id="14" name="Query Update" type="Update" coordinate="[7,1,0],[10,16,0]"/>
   <process id="21" name="Query Insert" type="Insert" coordinate="[12,1,0],[14,26,0]"/>
   <table id="2" schema="Person" name="Person.Person" alias="p" type="table" coordinate="[4,6,0],[4,24,0]">
      <column id="28" name="FirstName" coordinate="[3,8,0],[3,19,0]"/>
      <column id="29" name="LastName" coordinate="[3,21,0],[3,31,0]"/>
      <column id="20" name="id" coordinate="[10,11,0],[10,15,0]"/>
   </table>
   <view id="10" schema="dbo" name="dbo.hiredate_view" type="view" processIds="11 14 21" coordinate="[1,13,0],[1,30,0]">
      <column id="12" name="FirstName" coordinate="[1,31,0],[1,40,0]"/>
      <column id="13" name="LastName" coordinate="[1,41,0],[1,49,0]"/>
      <column id="19" name="id" coordinate="[10,4,0],[10,8,0]"/>
   </view>
   <resultset id="6" name="RS-1" type="select_list" coordinate="[3,8,0],[3,31,0]">
      <column id="7" name="FirstName" coordinate="[3,8,0],[3,19,0]"/>
      <column id="8" name="LastName" coordinate="[3,21,0],[3,31,0]"/>
   </resultset>
   <resultset id="25" name="INSERT-SELECT-1" type="insert-select" coordinate="[13,8,0],[13,31,0]">
      <column id="26" name="FirstName" coordinate="[13,8,0],[13,19,0]"/>
      <column id="27" name="LastName" coordinate="[13,21,0],[13,31,0]"/>
   </resultset>
   <resultset id="16" name="UPDATE-SET-1" type="update-set" coordinate="[7,1,0],[10,16,0]">
      <column id="17" name="FirstName" coordinate="[8,6,0],[8,17,0]"/>
      <column id="15" name="PseudoRows" coordinate="[7,1,0],[10,16,0]" source="system"/>
   </resultset>
   <relation id="1" type="fdd" effectType="select">
      <target id="7" column="FirstName" parent_id="6" parent_name="RS-1" coordinate="[3,8,0],[3,19,0]"/>
      <source id="28" column="FirstName" parent_id="2" parent_name="Person.Person" coordinate="[3,8,0],[3,19,0]"/>
   </relation>
   <relation id="2" type="fdd" effectType="select">
      <target id="8" column="LastName" parent_id="6" parent_name="RS-1" coordinate="[3,21,0],[3,31,0]"/>
      <source id="29" column="LastName" parent_id="2" parent_name="Person.Person" coordinate="[3,21,0],[3,31,0]"/>
   </relation>
   <relation id="3" type="fdd" effectType="create_view">
```

```
      <target id="12" column="FirstName" parent_id="10" parent_name="dbo.hiredate_view" coordinate="[1,31,0],[1,40,0]"/>
      <source id="7" column="FirstName" parent_id="6" parent_name="RS-1" coordinate="[3,8,0],[3,19,0]"/>
    </relation>
    <relation id="4" type="fdd" effectType="create_view">
      <target id="13" column="LastName" parent_id="10" parent_name="dbo.hiredate_view" coordinate="[1,41,0],[1,49,0]"/>
      <source id="8" column="LastName" parent_id="6" parent_name="RS-1" coordinate="[3,21,0],[3,31,0]"/>
    </relation>
    <relation id="5" type="fdd" effectType="update">
      <target id="17" column="FirstName" parent_id="16" parent_name="UPDATE-SET-1" coordinate="[8,6,0],[8,17,0]"/>
      <source id="28" column="FirstName" parent_id="2" parent_name="Person.Person" coordinate="[3,8,0],[3,19,0]"/>
    </relation>
    <relation id="6" type="fdd" effectType="update">
      <target id="12" column="FirstName" parent_id="10" parent_name="dbo.hiredate_view" coordinate="[1,31,0],[1,40,0]"/>
      <source id="17" column="FirstName" parent_id="16" parent_name="UPDATE-SET-1" coordinate="[8,6,0],[8,17,0]"/>
    </relation>
    <relation id="8" type="fdd" effectType="select">
      <target id="26" column="FirstName" parent_id="25" parent_name="INSERT-SELECT-1" coordinate="[13,8,0],[13,19,0]"/>
      <source id="28" column="FirstName" parent_id="2" parent_name="Person.Person" coordinate="[3,8,0],[3,19,0]"/>
    </relation>
    <relation id="9" type="fdd" effectType="select">
      <target id="27" column="LastName" parent_id="25" parent_name="INSERT-SELECT-1" coordinate="[13,21,0],[13,31,0]"/>
      <source id="29" column="LastName" parent_id="2" parent_name="Person.Person" coordinate="[3,21,0],[3,31,0]"/>
    </relation>
    <relation id="10" type="fdd" effectType="insert">
      <target id="12" column="FirstName" parent_id="10" parent_name="dbo.hiredate_view" coordinate="[1,31,0],[1,40,0]"/>
      <source id="26" column="FirstName" parent_id="25" parent_name="INSERT-SELECT-1" coordinate="[13,8,0],[13,19,0]"/>
    </relation>
    <relation id="11" type="fdd" effectType="insert">
      <target id="13" column="LastName" parent_id="10" parent_name="dbo.hiredate_view" coordinate="[1,41,0],[1,49,0]"/>
      <source id="27" column="LastName" parent_id="25" parent_name="INSERT-SELECT-1" coordinate="[13,21,0],[13,31,0]"/>
    </relation>
    <relation id="7" type="fdr" effectType="update">
      <target id="15" column="PseudoRows" parent_id="16" parent_name="UPDATE-SET-1" coordinate="[7,1,0],[10,16,0]" source="system"/>
      <source id="19" column="id" parent_id="10" parent_name="dbo.hiredate_view" coordinate="[10,4,0],[10,8,0]" clauseType="joinCondition
      <source id="20" column="id" parent_id="2" parent_name="Person.Person" coordinate="[10,11,0],[10,15,0]" clauseType="joinCondition"/>
    </relation>
  </dlineage>
```

diagram




exported lineage

```
source_db;source_schema;source_table;source_column;target_db;target_schema;target_table;target_column;procedure_names;query_hash_id
default;Person;Person;FirstName;default;default;hashId_of_update-set-1;FirstName;batchQueries;hashId_of_query
default;default;hashId_of_update-set-1;FirstName;default;dbo;hiredate_view;FirstName;batchQueries;hashId_of_query
```

The column name in the exported lineage **shoudn't be qualified** , it must be like this `FirstName` . But when it is written to the data catalog such as Atlas, it must be qualified like this: `default.dbo.hiredate_view.FirstName`.

The `hashId_of_update-set-1` is the pseduo name of the update-set resultset, it is MD5 value of string `type+column name in resultset`.

The `hashId_of_query` is the MD5 value of the SQL query text from which this lineage is generated.

Both of those hashId are used in order to make sure the resultset name or query from the same SQL statement is the same every time the SQL

statement is executed.

## export metadata to an RDBMS database

There are 2 tables created in the database in order to store the metadata.

### sqlflow_dbobjects

This table is used to store metadata of all database objects except column which is stored in `sqlflow_columns` table.

#### fields

1. guid, this is the unique identity number of the object
2. parent_id, the guid of the parent object.
3. qualified_name, the fully qualified object name
4. object_type, type of this object.
5. ddl, the SQL script used to define this object, such as create table statement.
6. ddl_hashId, hash code of the ddl to unique identify a ddl
7. comment, comment about this object

qualified_name should be unique in the same object_type.

So, there is a unique key of this table: (qualified_name, object_type).

Available value for object_type is: cluster, database, table, view, column, procedure, function, trigger.

The following predefined rows should be insert into this table:

```
insert into sqlflow_dbobjects(guid,parent_id,object_type,qualified_name) values ('101','1','sqldialect','bigquery');
insert into sqlflow_dbobjects(guid,parent_id,object_type,qualified_name) values ('102','1','sqldialect','couchbase');
insert into sqlflow_dbobjects(guid,parent_id,object_type,qualified_name) values ('103','1','sqldialect','dax');
insert into sqlflow_dbobjects(guid,parent_id,object_type,qualified_name) values ('104','1','sqldialect','db2');
insert into sqlflow_dbobjects(guid,parent_id,object_type,qualified_name) values ('105','1','sqldialect','exasol');
insert into sqlflow_dbobjects(guid,parent_id,object_type,qualified_name) values ('106','1','sqldialect','greenplum');
insert into sqlflow_dbobjects(guid,parent_id,object_type,qualified_name) values ('107','1','sqldialect','hana');
insert into sqlflow_dbobjects(guid,parent_id,object_type,qualified_name) values ('108','1','sqldialect','hive');
insert into sqlflow_dbobjects(guid,parent_id,object_type,qualified_name) values ('109','1','sqldialect','impala');
insert into sqlflow_dbobjects(guid,parent_id,object_type,qualified_name) values ('121','1','sqldialect','informix');
insert into sqlflow_dbobjects(guid,parent_id,object_type,qualified_name) values ('122','1','sqldialect','mdx');
insert into sqlflow_dbobjects(guid,parent_id,object_type,qualified_name) values ('123','1','sqldialect','mysql');
insert into sqlflow_dbobjects(guid,parent_id,object_type,qualified_name) values ('124','1','sqldialect','netezza');
insert into sqlflow_dbobjects(guid,parent_id,object_type,qualified_name) values ('125','1','sqldialect','odbc');
insert into sqlflow_dbobjects(guid,parent_id,object_type,qualified_name) values ('126','1','sqldialect','openedge');
insert into sqlflow_dbobjects(guid,parent_id,object_type,qualified_name) values ('127','1','sqldialect','oracle');
insert into sqlflow_dbobjects(guid,parent_id,object_type,qualified_name) values ('128','1','sqldialect','postgresql');
insert into sqlflow_dbobjects(guid,parent_id,object_type,qualified_name) values ('129','1','sqldialect','redshift');
insert into sqlflow_dbobjects(guid,parent_id,object_type,qualified_name) values ('121','1','sqldialect','snowflake');
insert into sqlflow_dbobjects(guid,parent_id,object_type,qualified_name) values ('122','1','sqldialect','sparksql');
insert into sqlflow_dbobjects(guid,parent_id,object_type,qualified_name) values ('123','1','sqldialect','sqlserver');
insert into sqlflow_dbobjects(guid,parent_id,object_type,qualified_name) values ('124','1','sqldialect','sybase');
insert into sqlflow_dbobjects(guid,parent_id,object_type,qualified_name) values ('125','1','sqldialect','teradata');
insert into sqlflow_dbobjects(guid,parent_id,object_type,qualified_name) values ('126','1','sqldialect','vertica');
```

#### insert a new cluster

insert a new hive cluster, with name `primary` and link to `hive` sql dialect which id is `108`

```
insert into sqlflow_dbobjects(guid,parent_id,object_type,qualified_name) values ('1001','108','cluster','primary');
```

#### insert a new database

```
insert into sqlflow_dbobjects(guid,parent_id,object_type,qualified_name) values ('2001','1001','database','sampledb@primary');
```

### insert a new table

```
insert into sqlflow_dbobjects(guid,parent_id,object_type,qualified_name) values ('3001','2001','table','sampledb.tableA@primary');
```

## sqlflow_columns

This table is used to store all columns.

### fields

1. guid, this is the unique identity number of the column
2. parent_id, the guid of the table which includes this column
3. qualified_name, the fully qualified object name
4. comment, comment about this column

### insert a column

```
insert into sqlflow_dbobjects(guid,parent_id,qualified_name,comment)
values ('3001','2001','sampledb.tableA.columnB@primary','this is the comment');
```

## export metadata to Atlas

.keep