# Ставим PIP и DBT:

sudo apt-get install python3-pip

pip install --upgrade setuptools wheel

pip install dbt-core

pip install dbt-clickhouse

dbt --version

mkdir /imdb

cd /imdb

dbt init imdb

# В clickhouse-client создаем таблицы:

CREATE DATABASE imdb;

CREATE TABLE imdb.actors

(

id UInt32,

first\_name String,

last\_name String,

gender FixedString(1)

) ENGINE = MergeTree ORDER BY (id, first\_name, last\_name, gender);

CREATE TABLE imdb.directors

(

id UInt32,

first\_name String,

last\_name String

) ENGINE = MergeTree ORDER BY (id, first\_name, last\_name);

CREATE TABLE imdb.genres

(

movie\_id UInt32,

genre String

) ENGINE = MergeTree ORDER BY (movie\_id, genre);

CREATE TABLE imdb.movie\_directors

(

director\_id UInt32,

movie\_id UInt64

) ENGINE = MergeTree ORDER BY (director\_id, movie\_id);

CREATE TABLE imdb.movies

(

id UInt32,

name String,

year UInt32,

rank Float32 DEFAULT 0

) ENGINE = MergeTree ORDER BY (id, name, year);

CREATE TABLE imdb.roles

(

actor\_id UInt32,

movie\_id UInt32,

role String,

created\_at DateTime DEFAULT now()

) ENGINE = MergeTree ORDER BY (actor\_id, movie\_id);

# Далее грузим данные:

INSERT INTO imdb.actors

SELECT \*

FROM s3('https://datasets-documentation.s3.eu-west-3.amazonaws.com/imdb/imdb\_ijs\_actors.tsv.gz',

'TSVWithNames');

INSERT INTO imdb.directors

SELECT \*

FROM s3('https://datasets-documentation.s3.eu-west-3.amazonaws.com/imdb/imdb\_ijs\_directors.tsv.gz',

'TSVWithNames');

INSERT INTO imdb.genres

SELECT \*

FROM s3('https://datasets-documentation.s3.eu-west-3.amazonaws.com/imdb/imdb\_ijs\_movies\_genres.tsv.gz',

'TSVWithNames');

INSERT INTO imdb.movie\_directors

SELECT \*

FROM s3('https://datasets-documentation.s3.eu-west-3.amazonaws.com/imdb/imdb\_ijs\_movies\_directors.tsv.gz',

'TSVWithNames');

INSERT INTO imdb.movies

SELECT \*

FROM s3('https://datasets-documentation.s3.eu-west-3.amazonaws.com/imdb/imdb\_ijs\_movies.tsv.gz',

'TSVWithNames');

INSERT INTO imdb.roles(actor\_id, movie\_id, role)

SELECT actor\_id, movie\_id, role

FROM s3('https://datasets-documentation.s3.eu-west-3.amazonaws.com/imdb/imdb\_ijs\_roles.tsv.gz',

'TSVWithNames');

# Проверяем, что все загрузилось:

SELECT id,

any(actor\_name) as name,

uniqExact(movie\_id) as num\_movies,

avg(rank) as avg\_rank,

uniqExact(genre) as unique\_genres,

uniqExact(director\_name) as uniq\_directors,

max(created\_at) as updated\_at

FROM (

SELECT imdb.actors.id as id,

concat(imdb.actors.first\_name, ' ', imdb.actors.last\_name) as actor\_name,

imdb.movies.id as movie\_id,

imdb.movies.rank as rank,

genre,

concat(imdb.directors.first\_name, ' ', imdb.directors.last\_name) as director\_name,

created\_at

FROM imdb.actors

JOIN imdb.roles ON imdb.roles.actor\_id = imdb.actors.id

LEFT OUTER JOIN imdb.movies ON imdb.movies.id = imdb.roles.movie\_id

LEFT OUTER JOIN imdb.genres ON imdb.genres.movie\_id = imdb.movies.id

LEFT OUTER JOIN imdb.movie\_directors ON imdb.movie\_directors.movie\_id = imdb.movies.id

LEFT OUTER JOIN imdb.directors ON imdb.directors.id = imdb.movie\_directors.director\_id

)

GROUP BY id

ORDER BY num\_movies DESC

LIMIT 5;

# Далее инициализируем проект:

dbt init imdb

cd imdb

# Создаем профиль dbt:

mcedit dbt\_project.yml

В него добавляем строку

profile: 'clickhouse\_imdb'

imdb:

actors:

+materialized: view

Далее редактируем файл

mcedit ~/.dbt/profiles.yml

И записываем в него параметры подключения к Clickhouse:

clickhouse\_imdb:

target: dev

outputs:

dev:

type: clickhouse

schema: imdb\_dbt

host: localhost

port: 8123

user: default

password: ''

secure: False  
Проверяем возможность подключения к Clickhouse:

dbt debug

# Материализация в виде создания представления

Удаляем каталог example:

rm -rf models/example

Создаем папку models/actors с файлами:

mkdir models/actors

touch models/actors/actor\_summary.sql

touch models/actors/schema.yml

Редактируем /actors/schema.yml:

mcedit models/actors/schema.yml

----------------

version: 2

sources:

- name: imdb

tables:

- name: directors

- name: actors

- name: roles

- name: movies

- name: genres

- name: movie\_directors

-----------------------

Далее записываем в actor\_summary:

mcedit models/actors/actors\_summary.sql

---------------

{{ config(materialized='view') }}

with actor\_summary as (

SELECT id,

any(actor\_name) as name,

uniqExact(movie\_id) as num\_movies,

avg(rank) as avg\_rank,

uniqExact(genre) as genres,

uniqExact(director\_name) as directors,

max(created\_at) as updated\_at

FROM (

SELECT {{ source('imdb', 'actors') }}.id as id,

concat({{ source('imdb', 'actors') }}.first\_name, ' ', {{ source('imdb', 'actors') }}.last\_name) as actor\_name,

{{ source('imdb', 'movies') }}.id as movie\_id,

{{ source('imdb', 'movies') }}.rank as rank,

genre,

concat({{ source('imdb', 'directors') }}.first\_name, ' ', {{ source('imdb', 'directors') }}.last\_name) as director\_name,

created\_at

FROM {{ source('imdb', 'actors') }}

JOIN {{ source('imdb', 'roles') }} ON {{ source('imdb', 'roles') }}.actor\_id = {{ source('imdb', 'actors') }}.id

LEFT OUTER JOIN {{ source('imdb', 'movies') }} ON {{ source('imdb', 'movies') }}.id = {{ source('imdb', 'roles') }}.movie\_id

LEFT OUTER JOIN {{ source('imdb', 'genres') }} ON {{ source('imdb', 'genres') }}.movie\_id = {{ source('imdb', 'movies') }}.id

LEFT OUTER JOIN {{ source('imdb', 'movie\_directors') }} ON {{ source('imdb', 'movie\_directors') }}.movie\_id = {{ source('imdb', 'movies') }}.id

LEFT OUTER JOIN {{ source('imdb', 'directors') }} ON {{ source('imdb', 'directors') }}.id = {{ source('imdb', 'movie\_directors') }}.director\_id

)

GROUP BY id

)

select \*

from actor\_summary

-------------------

В actor\_summary добавлен столбец updated\_at. Пока он не используется, понадобится позже.

Запускаем dbt run в каталоге imdb:

dbt run

Появилась новая база данных imdb\_dbt

SHOW DATABASES;

И появилось представление actor\_summary:

SELECT \* FROM imdb\_dbt.actor\_summary ORDER BY num\_movies DESC LIMIT 5;

# Материализация в виде создания таблицы

Изменяем файл

mcedit models/actors/actor\_summary.sql

Меняем верхнюю строку на

{{ config(order\_by='(updated\_at, id, name)', engine='MergeTree()', materialized='table') }}

Далее

dbt run

Смотрим, какой код получился на создание таблицы:

SHOW CREATE TABLE imdb\_dbt.actor\_summary;

Можно проверить содержание таблицы:

SELECT \* FROM imdb\_dbt.actor\_summary ORDER BY num\_movies DESC LIMIT 5;

# Инкрементальная материализация

First, we modify our model to be of type incremental. This addition requires:

1. **unique\_key** - To ensure the plugin can uniquely identify rows, we must provide a unique\_key - in this case, the **id** field from our query will suffice. This ensures we will have no row duplicates in our materialized table. For more details on uniqueness constraints, see[here](https://docs.getdbt.com/docs/building-a-dbt-project/building-models/configuring-incremental-models" \l "defining-a-uniqueness-constraint-optional" \t "_blank).
2. **Incremental filter** - We also need to tell dbt how it should identify which rows have changed on an incremental run. This is achieved by providing a delta expression. Typically this involves a timestamp for event data; hence our updated\_at timestamp field. This column, which defaults to the value of now() when rows are inserted, allows new roles to be identified. Additionally, we need to identify the alternative case where new actors are added. Using the **{{this}}** variable, to denote the existing materialized table, this gives us the expression **where id > (select max(id) from {{ this }}) or updated\_at > (select max(updated\_at) from {{this}})**. We embed this inside the **{% if is\_incremental() %}** condition, ensuring it is only used on incremental runs and not when the table is first constructed.

Изменяем файл

mcedit models/actors/actor\_summary.sql

Добавляем в него контент:

{{ config(order\_by='(updated\_at, id, name)', engine='MergeTree()', materialized='incremental', unique\_key='id') }}

with actor\_summary as (

SELECT id,

any(actor\_name) as name,

uniqExact(movie\_id) as num\_movies,

avg(rank) as avg\_rank,

uniqExact(genre) as genres,

uniqExact(director\_name) as directors,

max(created\_at) as updated\_at

FROM (

SELECT {{ source('imdb', 'actors') }}.id as id,

concat({{ source('imdb', 'actors') }}.first\_name, ' ', {{ source('imdb', 'actors') }}.last\_name) as actor\_name,

{{ source('imdb', 'movies') }}.id as movie\_id,

{{ source('imdb', 'movies') }}.rank as rank,

genre,

concat({{ source('imdb', 'directors') }}.first\_name, ' ', {{ source('imdb', 'directors') }}.last\_name) as director\_name,

created\_at

FROM {{ source('imdb', 'actors') }}

JOIN {{ source('imdb', 'roles') }} ON {{ source('imdb', 'roles') }}.actor\_id = {{ source('imdb', 'actors') }}.id

LEFT OUTER JOIN {{ source('imdb', 'movies') }} ON {{ source('imdb', 'movies') }}.id = {{ source('imdb', 'roles') }}.movie\_id

LEFT OUTER JOIN {{ source('imdb', 'genres') }} ON {{ source('imdb', 'genres') }}.movie\_id = {{ source('imdb', 'movies') }}.id

LEFT OUTER JOIN {{ source('imdb', 'movie\_directors') }} ON {{ source('imdb', 'movie\_directors') }}.movie\_id = {{ source('imdb', 'movies') }}.id

LEFT OUTER JOIN {{ source('imdb', 'directors') }} ON {{ source('imdb', 'directors') }}.id = {{ source('imdb', 'movie\_directors') }}.director\_id

)

GROUP BY id

)

select \*

from actor\_summary

{% if is\_incremental() %}

-- this filter will only be applied on an incremental run

where id > (select max(id) from {{ this }}) or updated\_at > (select max(updated\_at) from {{this}})

{% endif %}

Запускаем dbt run, чтобы обновить модель:

dbt run

Проверяем актеров по количеству фильмов:

SELECT \* FROM imdb\_dbt.actor\_summary ORDER BY num\_movies DESC LIMIT 5;

Дальше добавляем значение в actors:

INSERT INTO imdb.actors VALUES (845466, 'Clicky', 'McClickHouse', 'M');

и в roles:

INSERT INTO imdb.roles

SELECT now() as created\_at, 845466 as actor\_id, id as movie\_id, 'Himself' as role

FROM imdb.movies

LIMIT 910 OFFSET 10000;

Проверяем, что он реально добавился:

SELECT id,

any(actor\_name) as name,

uniqExact(movie\_id) as num\_movies,

avg(rank) as avg\_rank,

uniqExact(genre) as unique\_genres,

uniqExact(director\_name) as uniq\_directors,

max(created\_at) as updated\_at

FROM (

SELECT imdb.actors.id as id,

concat(imdb.actors.first\_name, ' ', imdb.actors.last\_name) as actor\_name,

imdb.movies.id as movie\_id,

imdb.movies.rank as rank,

genre,

concat(imdb.directors.first\_name, ' ', imdb.directors.last\_name) as director\_name,

created\_at

FROM imdb.actors

JOIN imdb.roles ON imdb.roles.actor\_id = imdb.actors.id

LEFT OUTER JOIN imdb.movies ON imdb.movies.id = imdb.roles.movie\_id

LEFT OUTER JOIN imdb.genres ON imdb.genres.movie\_id = imdb.movies.id

LEFT OUTER JOIN imdb.movie\_directors ON imdb.movie\_directors.movie\_id = imdb.movies.id

LEFT OUTER JOIN imdb.directors ON imdb.directors.id = imdb.movie\_directors.director\_id

)

GROUP BY id

ORDER BY num\_movies DESC

LIMIT 2;

Запускаем dbt run и убеждаемся, что актер реально появился в actor\_summary:

dbt run

SELECT \* FROM imdb\_dbt.actor\_summary ORDER BY num\_movies DESC LIMIT 2;

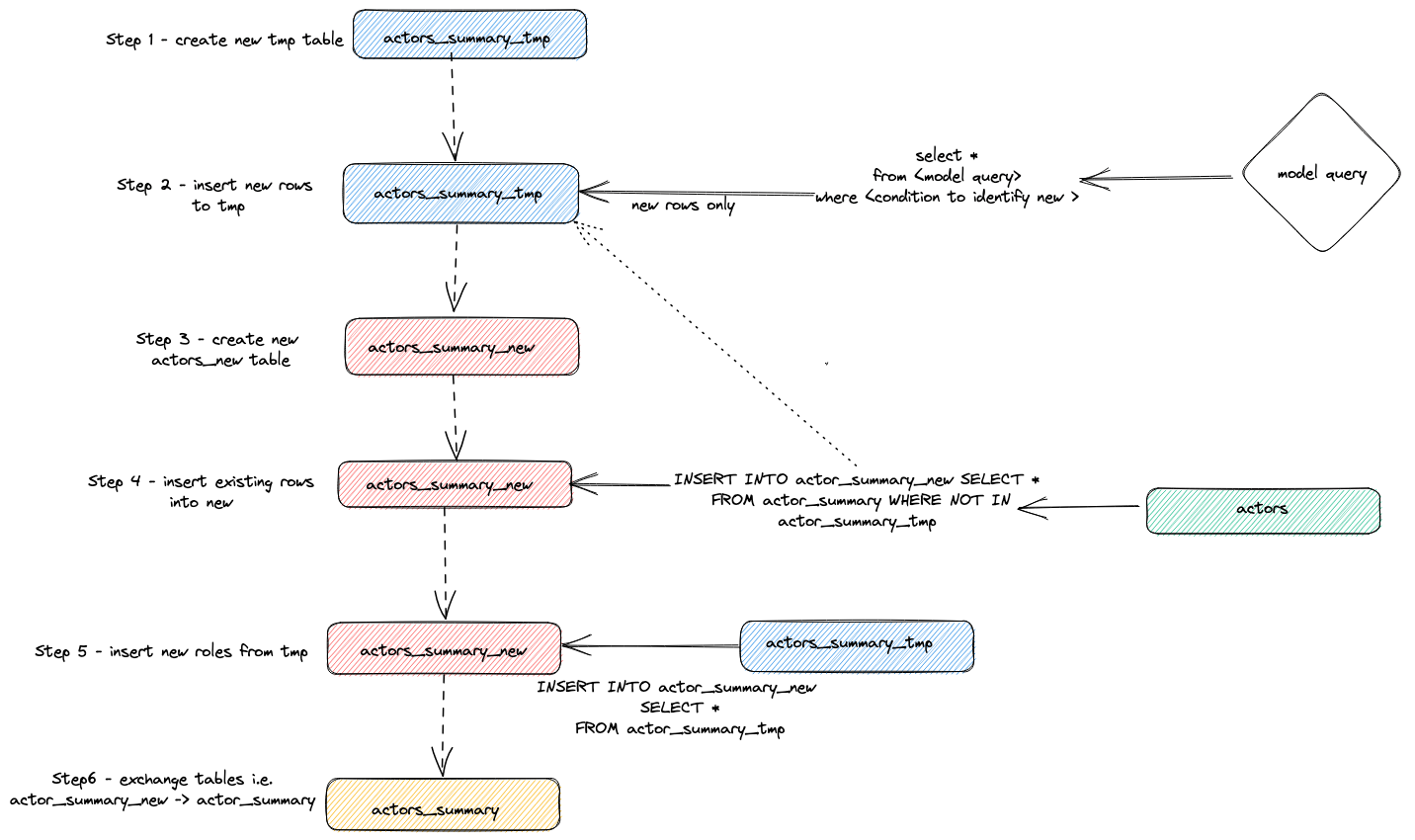
Реально произошло следующее:

SELECT event\_time, query FROM system.query\_log WHERE type='QueryStart' AND query LIKE '%dbt%'

AND event\_time > subtractMinutes(now(), 15) ORDER BY event\_time LIMIT 100;

1. The plugin creates a temporary table **actor\_sumary\_\_dbt\_tmp**. Rows that have changed are streamed into this table.
2. A new table, **actor\_summary\_new,** is created. The rows from the old table are, in turn, streamed from the old to new, with a check to make sure row ids do not exist in the temporary table. This effectively handles updates and duplicates.
3. The results from the temporary table are streamed into the new **actor\_summary** table:
4. Finally, the new table is exchanged atomically with the old version via an **EXCHANGE TABLES** statement. The old and temporary tables are in turn dropped.

This is visualized below:



# Режим добавления при инкрементальных обновлениях

To overcome the limitations of large datasets in incremental models, the plugin uses the dbt configuration parameter incremental\_strategy. This can be set to the value append. When set, updated rows are inserted directly into the target table (a.k.a imdb\_dbt.actor\_summary) and no temporary table is created. Note: Append only mode requires your data to be immutable or for duplicates to be acceptable. If you want an incremental table model that supports altered rows don't use this mode!

В actors\_summary.sql:

{{ config(order\_by='(updated\_at, id, name)', engine='MergeTree()', materialized='incremental', unique\_key='id', **incremental\_strategy='append'**) }}

Добавляем еще актера и роли:

INSERT INTO imdb.actors VALUES (845467, 'Danny', 'DeBito', 'M');

INSERT INTO imdb.roles

SELECT now() as created\_at, 845467 as actor\_id, id as movie\_id, 'Himself' as role

FROM imdb.movies

LIMIT 920 OFFSET 10000;

Запускаем dbt run:

dbt run

Смотрим, что получилось:

SELECT \* FROM imdb\_dbt.actor\_summary ORDER BY num\_movies DESC LIMIT 3;

Смотрим опять логи:

SELECT event\_time, query FROM system.query\_log WHERE type='QueryStart' AND query LIKE '%dbt%'

AND event\_time > subtractMinutes(now(), 15) ORDER BY event\_time LIMIT 100;

И видим, что промежуточных таблиц нет, вставка идет напрямую в actor\_summary.