Information Integration project

Integrating heterogeneous data sources into a unified global schema

Goal of the project

Accomplishing a **domain-based integration of multiple and heterogenous data sources**, merging them into a single global view that can be queried by the users.

The global schema should be able to satisfy queries not trivially satisfiable by accessing directly the data sources.

GAV – Mappings (1)

In GAV approach, the (sound) mappings are assertions as such:

$\forall x.\ \phi S(x) \rightarrow g(x)$

Where:

- S is the source schema
- ϕS is a CQ over the source DB
- g is a predicate of arity n that indicates an element of G

GAV (Global As View): the global schema is a view of the set of sources

GAV – Retrieved global database (2)

Given I = <G, S, M> and source database C for S, we denote the retrieved global database as M(C), that is the database obtained by:

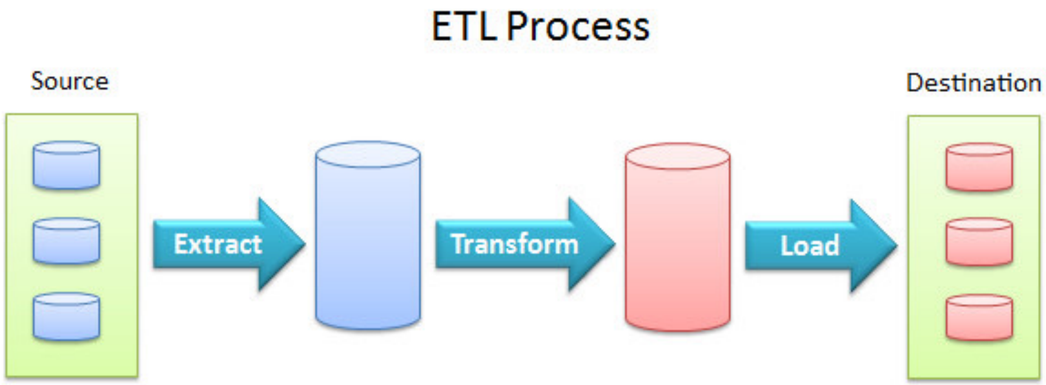
- Applying the CQs specified in the mappings
- Transferring the tuples from the sources to the global DB

Data integration using Pentaho (1)

The end result of the project is a **one-shot materialization** of the global schema, achieved by mappings in the form of Pentaho transformations.

Data integration using Pentaho (2)

Pentaho Kettle is a tool that allows loading input data in **heterogeneous formats and from different sources**, performing the transformations needed in order to map the information as requested.



Finally, the data is materialized in output as a SQL database in the example shown for this project.

Domain and data sources (1)

Critic reviews

Directors

Actors

Movies

Users Ratings

Production studios

Energy scores

Oscar awards

Domain and data sources (2)

- 7 Data sources

  kaggle IMDb m

  Filmtv

- 13 Files (TSV, CSV, JSON)

Domain and data sources (3)

IMDb

- title.principals.tsv
- title.basics.tsv
- name.basics.tsv

kaggle

- the_oscar_award.csv

kaggle m

- ratings.csv
- movies_metadata.csv
- links.csv

- movies.json

Domain and data sources (4)

- studios.csv
- movies.csv

- filmtv_movies.csv

- rotten_tomatoes_critic_reviews.csv
- rotten_tomatoes_movies.csv

Domain and data sources – Reconciliation issues (5)

- **Physical heterogeneity**

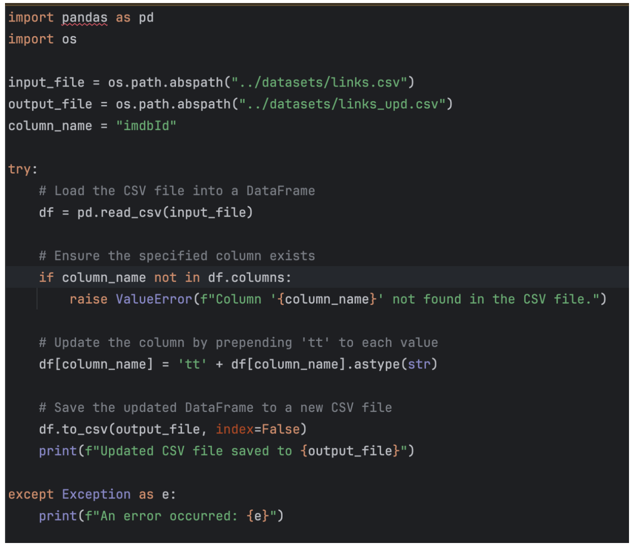
  Data stored in different formats: CSV, TSV and JSON
  Easily mitigated by Pentaho transformations that can handle multiple input formats

- **Conceptual heterogeneity**
  - Different datasets uniquely identify movies in diverse formats: IDs both in string and number data types
  - Some datasets refer to movies with their original title, some with their English title, some have both fields

Domain and data sources – Reconciliation issues (6)

- **Contextual heterogeneity**

  Datasets can refer to the same field using different representation: both IMDB and Movielens sources contain the IMDB ID, but the latter lacks a prefix in the string.



```python
import pandas as pd
import os

input_file = os.path.abspath("../datasets/links.csv")
output_file = os.path.abspath("../datasets/links_upd.csv")
column_name = "imdbId"

try:
    # Load the CSV file into a DataFrame
    df = pd.read_csv(input_file)

    # Ensure the specified column exists
    if column_name not in df.columns:
        raise ValueError(f"Column '{column_name}' not found in the CSV file.")

    # Update the column by prepending 'tt' to each value
    df[column_name] = 'tt' + df[column_name].astype(str)

    # Save the updated DataFrame to a new CSV file
    df.to_csv(output_file, index=False)
    print(f"Updated CSV file saved to {output_file}")

except Exception as e:
    print(f"An error occurred: {e}")
```

Python preprocessing required to solve the heterogeneity between sources

Source schema (1)

IMDBTitleBasics$_{/10}$ *(tconst, titleType, primaryTitle, originalTitle, isAdult, startYear, endYear, runtimeMinutes, genres)*

IMDBTitlePrincipals$_{/6}$ *(tconst, ordering, nconst, category, job, characters)*

IMDBNameBasics$_{/6}$ *(nconst, primaryName, birthYear, deathYear, primaryProfession, knownForTitles)*

MovielensRatings$_{/4}$ *(userId, movieId, rating, timestamp)*

MovielensMoviesMetadata$_{/24}$ *(adult, belongsToCollection, budget, genres, homepage, id, imdb_id, original_language, originalTitle, overview, popularity, poster_path, production_companies, production_countries, release_date, revenue, runtime, spoken_languages, status, tagline, title, video, vote_average, vote_count)*

MovielensLinks$_{/3}$ *(movieId, imdbId, tmdbId)*

Source schema (2)

TheOscarAward$_{/7}$ *(year_film, year_ceremony, ceremony, category, name, film, winner)*

LetterboxStudios$_{/2}$ *(id, studio)*

LetterboxMovies$_{/7}$ *(id, name, date, tagline, description, minute, rating)*

RottenTomatoesCriticReviews$_{/8}$ *(rotten_tomatoes_link, critic_name, top_critic, publisher_name, review_type, review_score, review_date, review_content)*

RottenTomatoesMovies$_{/22}$ *(rotten_tomatoes_link, movie_title, movie_info, critics_consensus, content_rating, genres, directors, authors, actors, original_release_date, streaming_release_date, runtime, production_company, tomatometer_status, tomatometer_rating, tomatometer_count, audience_status, audience_rating, audience_count, tomatometer_top_critics_count, tomatometer_fresh_critics_count, tomatometer_rotten_critics_count)*

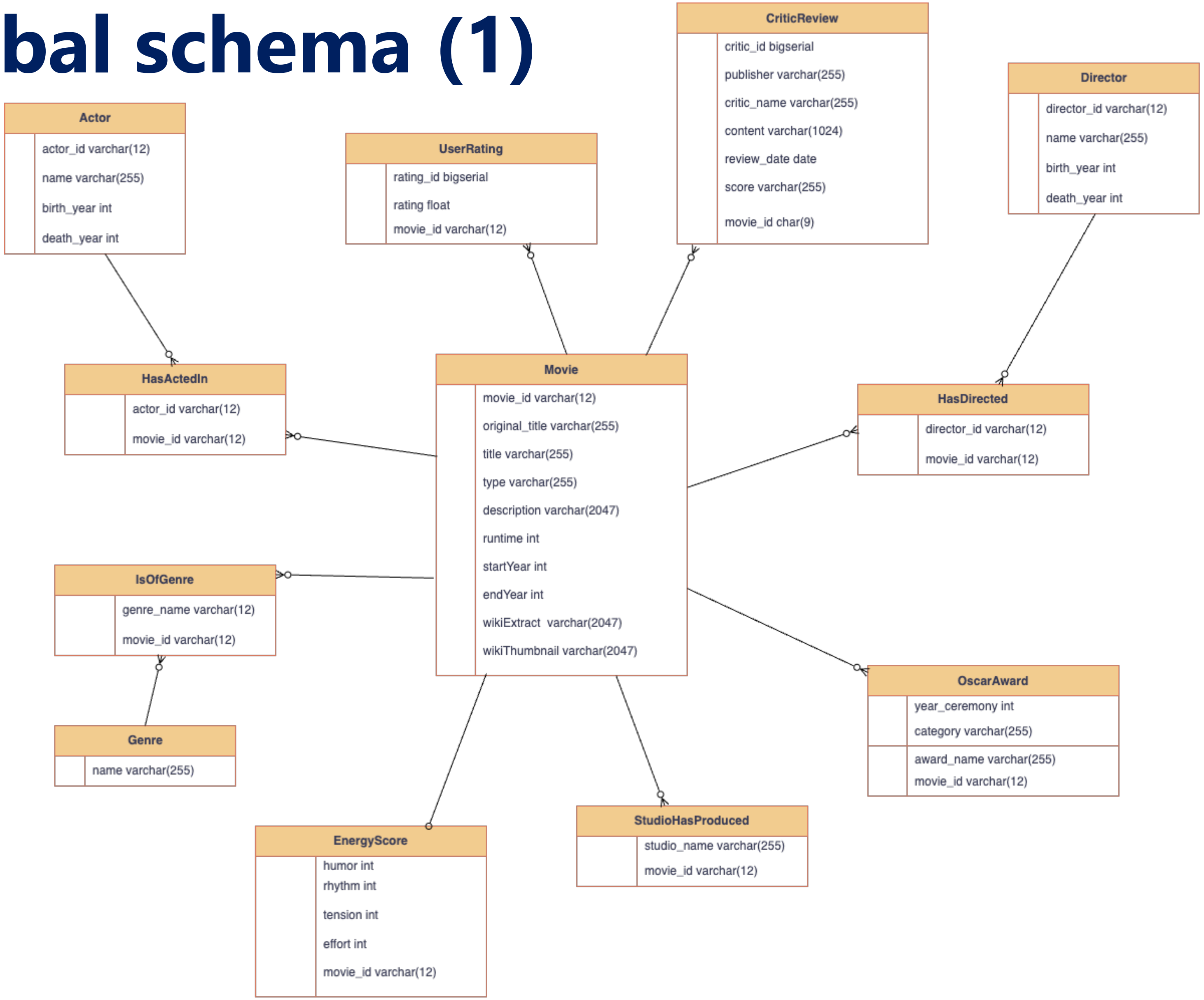
Source schema (3)

Additional sources added for Big Data Management:

WikiMovies$_{/9}$ *(title, year, cast, genres, href, extract, thumbnail, thumbnail_width, thumbnail_height)*

FilmTvMovies$_{/19}$ *(filmtv_id, title, year, genre, duration, country, directors, actors, avg_vote, critics_vote, public_vote, total_votes, description, notes, humor, rhythm, effort, tension, erotism)*

Global schema (1)



Actor
actor_id varchar(12)
name varchar(255)
birth_year int
death_year int

UserRating
rating_id bigserial
rating float
movie_id varchar(12)

CriticReview
critic_id bigserial
publisher varchar(255)
critic_name varchar(255)
content varchar(1024)
review_date date
score varchar(255)
movie_id char(9)

Director
director_id varchar(12)
name varchar(255)
birth_year int
death_year int

HasActedIn
actor_id varchar(12)
movie_id varchar(12)

Movie
movie_id varchar(12)
original_title varchar(255)
title varchar(255)
type varchar(255)
description varchar(2047)
runtime int
startYear int
endYear int
wikiExtract varchar(2047)
wikiThumbnail varchar(2047)

HasDirected
director_id varchar(12)
movie_id varchar(12)

IsOfGenre
genre_name varchar(12)
movie_id varchar(12)

Genre
name varchar(255)

OscarAward
year_ceremony int
category varchar(255)
award_name varchar(255)
movie_id varchar(12)

StudioHasProduced
studio_name varchar(255)
movie_id varchar(12)

EnergyScore
humor int
rhythm int
tension int
effort int
movie_id varchar(12)

Global schema (2)

Movie$_{/10}$ *(movie_id, original_title, title, type, description, runtime, startYear, endYear, wikiExtract, wikiThumbnail)*

UserRating$_{/3}$ *(rating_id, rating, movie_id)*

CriticReview$_{/7}$ *(critic_id, publisher, critic_name, content, review_date, score, movie_id)*

Actor$_{/4}$ *(actor_id, name, birth_year, death_year)*

HasActedIn$_{/2}$ *(actor_id, movie_id)*

StudioHasProduced$_{/2}$ *(studio_name, movie_id)*

Global schema (3)

Director$_{/4}$ *(director_id, name, birth_year, death_year)*

HasDirected$_{/2}$ *(director_id, movie_id)*

Genre$_{/1}$ *(name)*

IsOfGenre$_{/2}$ *(genre_name, movie_id)*

OscarAward$_{/4}$ *(year_ceremony, category, award_name, movie_id)*

EnergyScore$_{/5}$ *(humor, tension, effort, rhythm, movie_id)*

Conjunctive GAV FOL Mappings (1)

Movie:
∀ id, ot, ti, ty, dc, rt, sy, ey, ex, th.(∃ ia, g, ad, btc, bg, hp, mid, ol, ott, pop, pp, pcp,
pcn, rd, rev, rut, sl, st, tl, tit, va,vc, c, gr, hr,thw, thh.
(**IMDBTitleBasics** (id, ty, ti, ot, ia, sy, ey, rt, g) ∧
(**MovielensMovieMetadata**(ad, btc, bg, hp, mid, id, ol, ott, dc, pop, pp, pcp, pcn,
rd, rev, rut, sl, st, tl, tit, va, vc) ∨ dc = null) ∧
(**WikiMovies**((ti, sy, c, gr, hr, ex, th, thw, thh) ∨ (ex = null ∧ th = null)))) →
Movie(**id**, ot, ti, ty, dc, rt, sy, ey, ex, th)

UserRating:
∀rid, r, mid.(∃ uid, mid2, ts, tid.(**MovielensRatings** (uid, mid2, r, ts) ∧
MovielensLinks(mid2, **mid**, tid))) → UserRating(rid, r, **mid**)

Genre:
∀g.(∃ id, ty, ti, ot, ia, sy, ey, rt.(**IMDBTitleBasics** (id, ty, ti, ot, ia, sy, ey, rt, g))) →
Genre(g)

IsOfGenre:
∀g, id.(∃ id, ty, ti, ot, ia, sy, ey, rt.(**IMDBTitleBasics** (**id**, ty, ti, ot, ia, sy, ey, rt, g)))
→IsOfGenre(g, **id**)

Conjunctive GAV FOL Mappings (2)

Actor:
∀aid, n, by, dy.(∃ pp, kft.(IMDBNameBasics(aid, n, by, dy, pp, kft)))
→ Actor(aid, n, by, dy)

HasActedIn:
∀aid, mid.(∃ o, ca, j, ch.(IMDBTitlePrincipals(mid, o, aid,’$actor'$, j, ch))
→ HasActedIn(aid, mid)

Director:
∀did, n, by, dy.(∃ pp, kft.(IMDBNameBasics(did, n, by, dy, pp, kft)))
→ Actor(did, n, by, dy)

HasDirected:
∀did, mid.(∃ o, ca, j, ch.(IMDBTitlePrincipals(did, o, aid, ‘$director'$, j, ch))
→ HasDirected(did, mid)

Conjunctive GAV FOL Mappings (3)

CriticReview:
∀cid, p, cn, ct, rd, sc, mid.
(∃ rtl, tc, rt, mt, mi, cc, cr, g, d, a, ord, srd, r, pc, ts, tr, tc, as, ar, ac, ttcc, tfcc, trcc, ty, ot, ia, ey, rt2, g2. (RottenTomatoesCriticReviews (rtl, cn, tc, p, rt, sc, rd, ct) ∧
RottenTomatoesMovies(rtl, mt, mi, cc, cr, g, d, a, ord, srd, r, pc, ts, tr, tc, as, ar, ac, ttcc, tfcc, trcc) ∧
IMDBTitleBasics (**mid**, ty, mt, ot, ia, d, ey, rt2, g2))) →
CriticReview(cid, p, cn, ct, rd, sc, **mid**)

OscarAward:
∀ yc, ca, an, mid.
(∃ yf, ce, n, f, w, ty, ot, ia, ey, rt, g. TheOscarAward(yf, yc, ce, ca, n, f, w) ∧
IMDBTitleBasics (**mid**, ty, f, ot, ia, yf, ey, rt, g) ∧ w='true') →
OscarAward(yc, ca, an, **mid**)

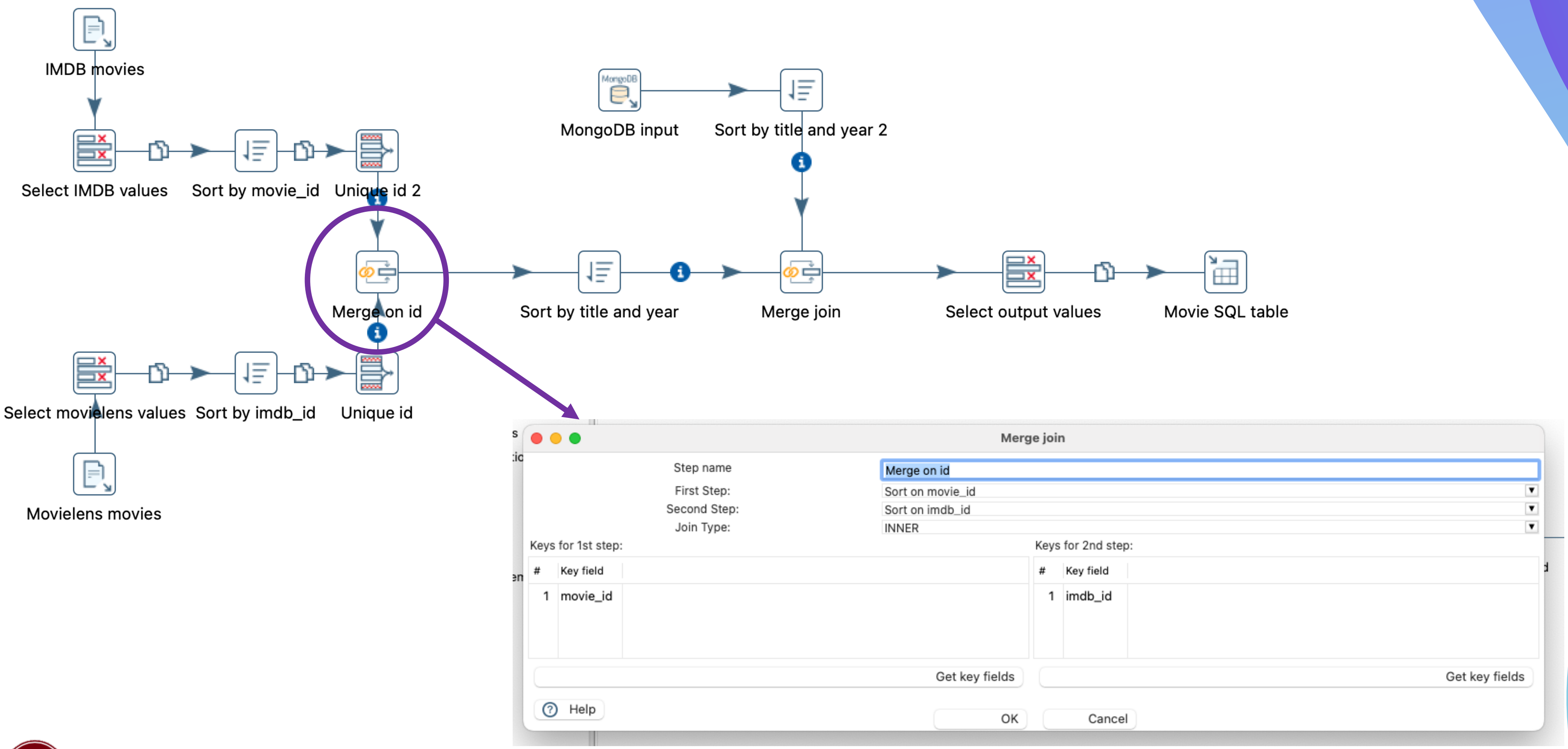
StudioHasProduced:
∀sn, mid.(∃ mid2, n, d, tl, dc, m , ty, ot, ia, ey, rt, g, r.(LetterboxStudios(mid2, sn) ∧
LetterboxMovies(mid2, n, d, tl, dc, m, r) ∧
IMDBTitleBasics (**mid**, ty, n, ot, ia, d, ey, rt, g))) →
StudioHasProduced(sn, **mid**)

Conjunctive GAV FOL Mappings (4)

EnergyScore:
∀aid, n, by, dy.(∃ ty, tot, ia, sy, ey, rt, g, fid, gr, dt, cy, ds, as, avg_v, crv, pbv, tvs, dp, ns, e.
(IMDBTitleBasics(id, ty, ti, ot, ia, sy, ey, rt, g) Λ
FilmTvMovies(fid, ti, sy, gr, dt, cy, ds, as, avg_v, crv, pbv, tvs, dp, ns, humor, rhythm, effort, tension, e))) →
EnergyScore(humor, rhythm, effort, tension, id)

Pentaho implementation (1)

Movie transformation



Step name: Merge on id

First Step: Sort on movie_id

Second Step: Sort on imdb_id

Join Type: INNER

Keys for 1st step:

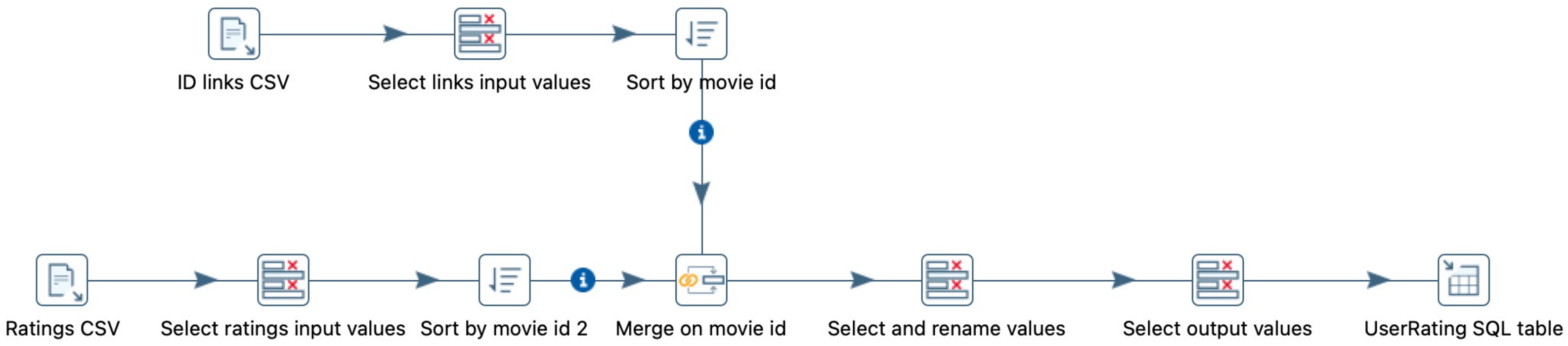
#	Key field
1	movie_id

Keys for 2nd step:

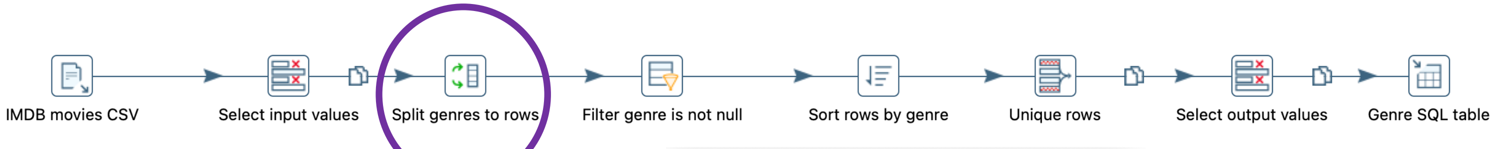
#	Key field
1	imdb_id

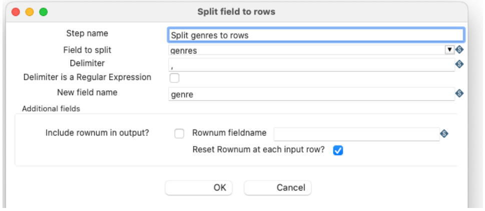
Pentaho implementation (2)

UserRating transformation



Genre transformation




Split field to rows

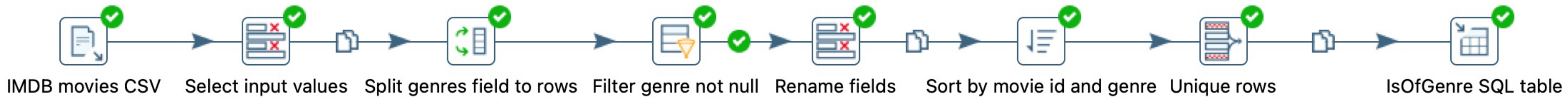
Step name: Split genres to rows
Field to split: genres
Delimiter: ,
Delimiter is a Regular Expression ☐
New field name: genre

Additional fields

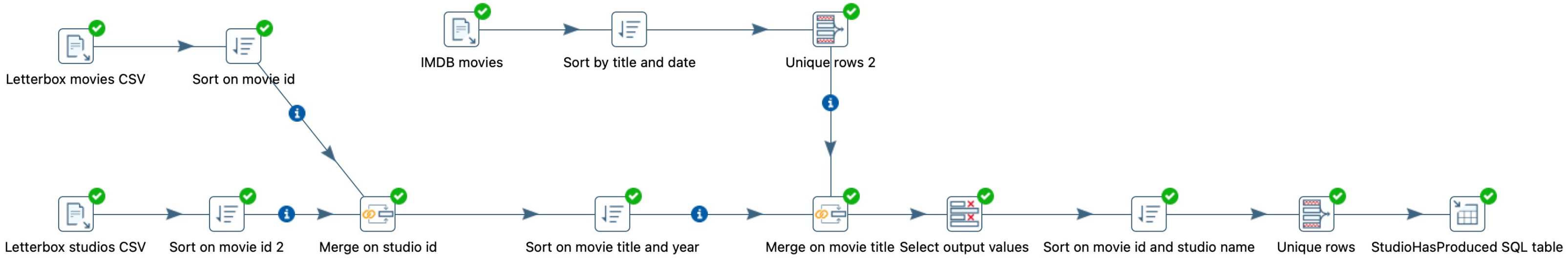
Include rownum in output? ☐ Rownum fieldname
Reset Rownum at each input row? ☑

Pentaho implementation (3)

IsOfGenre transformation

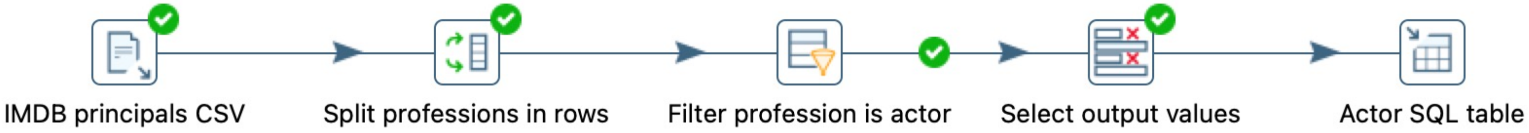


StudioHasProduced transformation

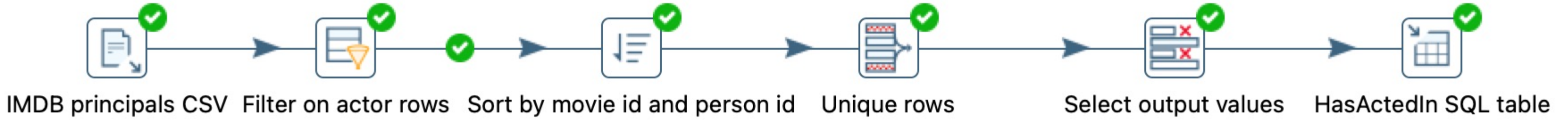


Pentaho implementation (4)

Actor transformation

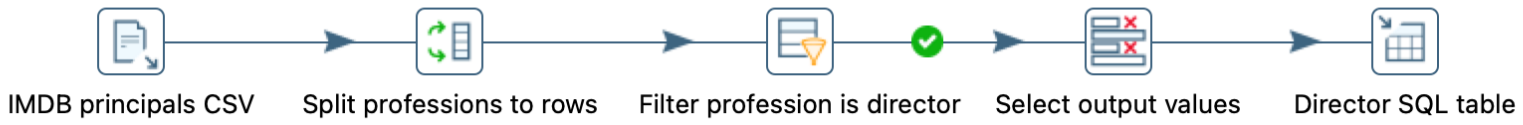


HasActedIn transformation

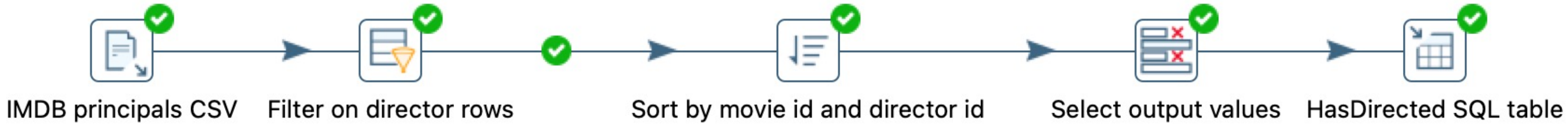


Pentaho implementation (5)

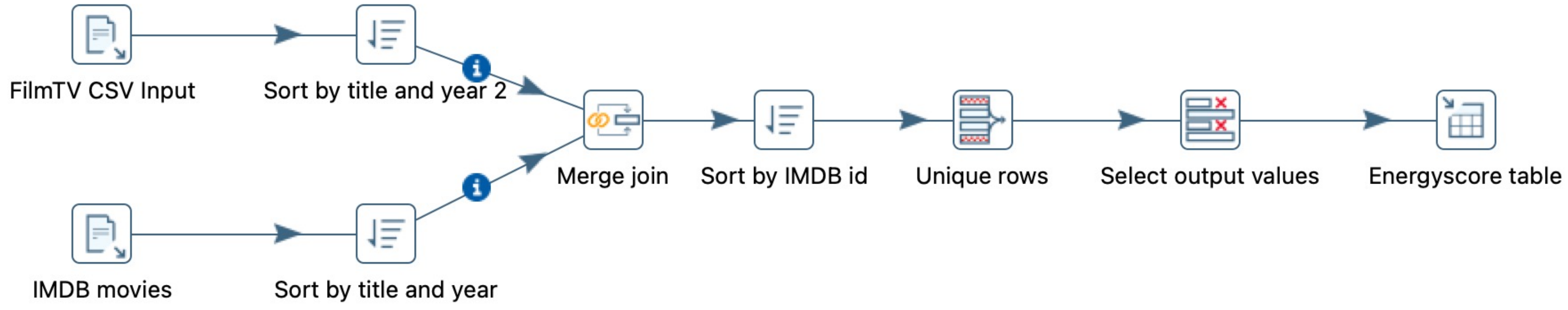
Director transformation



HasDirected transformation

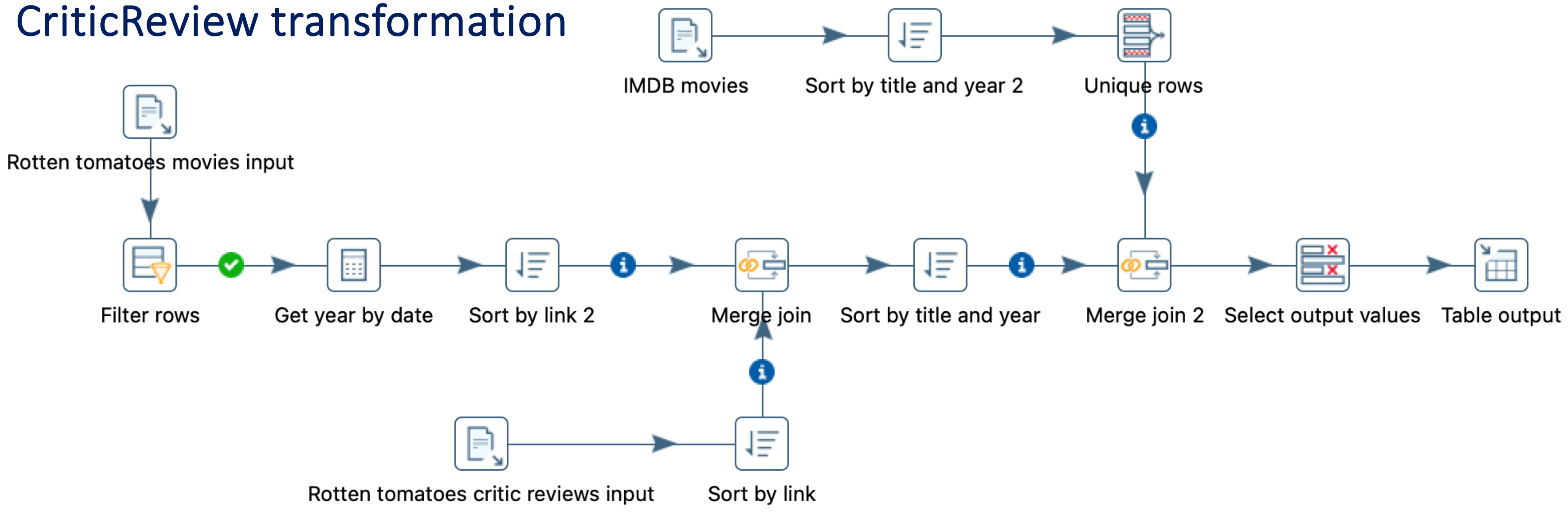


EnergyScore transformation

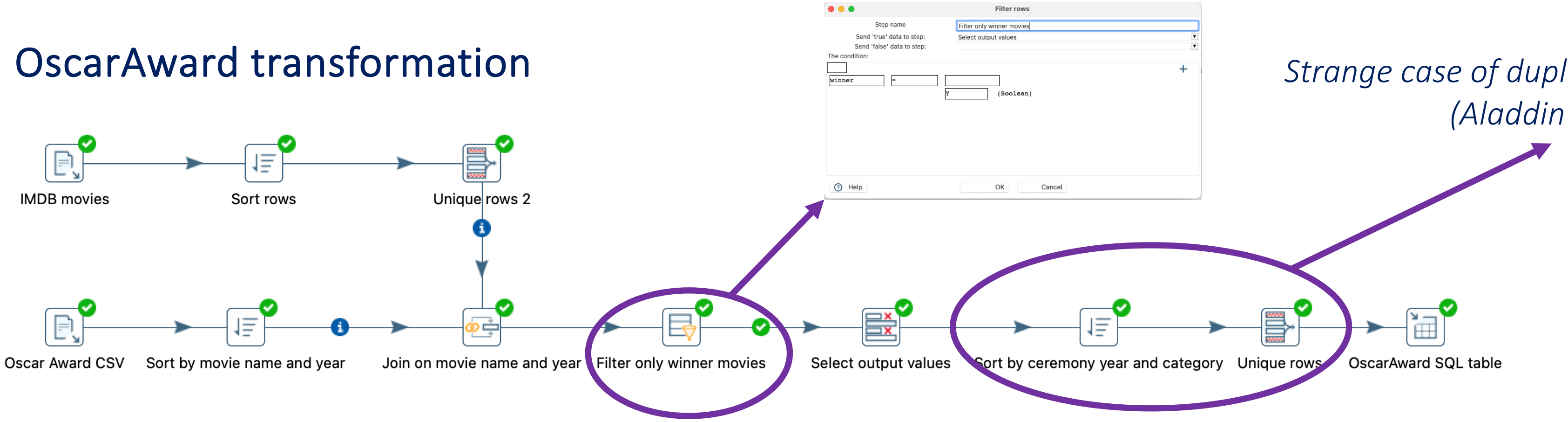


Pentaho implementation (6)

CriticReview transformation



OscarAward transformation



Step name: Filter only winner movies
Send 'true' data to step: Select output values
Send 'false' data to step:
The condition:
winner = Y (Boolean)

Strange case of duplicate Oscar Award (Aladdin 1993)

Some statistics…

- In the RGBD (Retrieved Global Database)
  - **>11.000.000** movies
  - **>26.000.000** reconciled user ratings
  - **>20.000.000** actor appearances

- **17 minutes each** for CriticRating, OscarAward, StudioHasProduced transformations (ran on M1 Pro)

- **7,09 GBs** of source data

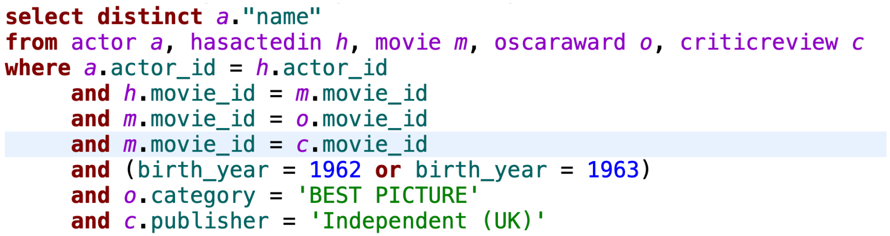
Query answering

Given the source database C, now that we have obtained the one-shot materialization of the **RGBD (Retrieved Global Database)** M(C), we can query it

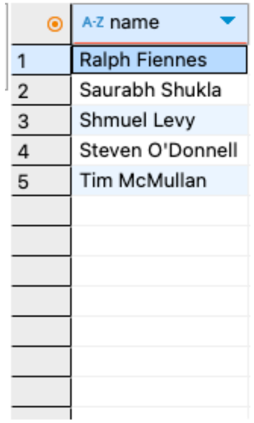
We have stated that, given a GAV setting without constraints, evaluating a query over M(C) is equivalent to unfolding it and evaluating it over the source database C, from the point of view of computing certain answers

Query 1

Name of actors born in 1962 or 1963 participating in a "Best Picture" Oscar-winning movie, criticized by the Independent (UK) publisher



```
select distinct a."name"
from actor a, hasactedin h, movie m, oscaraward o, criticreview c
where a.actor_id = h.actor_id
     and h.movie_id = m.movie_id
     and m.movie_id = o.movie_id
     and m.movie_id = c.movie_id
     and (birth_year = 1962 or birth_year = 1963)
     and o.category = 'BEST PICTURE'
     and c.publisher = 'Independent (UK)'
```



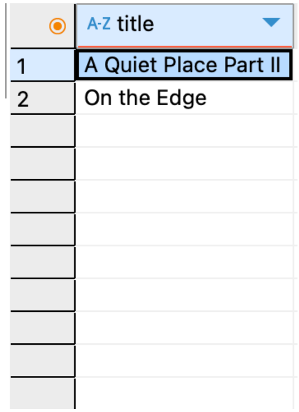
	A-Z name
1	Ralph Fiennes
2	Saurabh Shukla
3	Shmuel Levy
4	Steven O'Donnell
5	Tim McMullan

q = {(name) |
∃ aid, mid, name, by, dy, ot, ti, ty, dc, r, sy, ey, yc, cat, aw, cid, pub, cn, co, rd,sc.
(Actor(aid, name, 1962, dy) ⋁ Actor(aid, name, 1963, dy)) ⋀
HasActedIn(aid, mid) ⋀ Movie(mid, ot, ti, ty, dc, r, sy, ey) ⋀
OscarAward(yc, ‘BEST PICTURE’, aw, mid) ⋀ CriticReview(cid, ‘Independent (UK)’,
cn, co, rd, sc, mid)}

Query 2

Name of movies of genre Drama produced by "Paramount Pictures" where Cillian Murphy acted

```
select m.title
from genre g, isofgenre i, movie m, hasactedin h, studiohasproduced s, actor a
where g."name" = i.genre_name
     and i.movie_id = m.movie_id
     and m.movie_id = s.movie_id
     and i.movie_id = h.movie_id
     and h.actor_id = a.actor_id
     and s.studio_name = 'Paramount'
     and a."name" = 'Cillian Murphy'
     and g."name" = 'Drama'
```



	title
1	A Quiet Place Part II
2	On the Edge

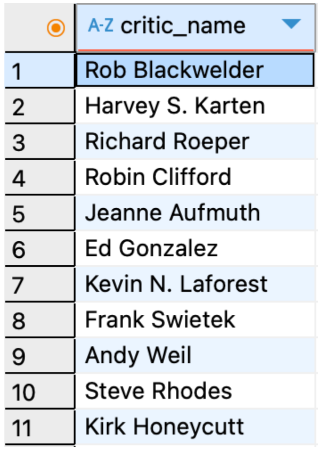
q = {(title) |
∃ aid, mid, by, dy, ot, ty, dc, r, sy, ey, .
Actor(aid, ‘Cillian Murphy’, by, dy) Ʌ HasActedIn(aid, mid)
Ʌ Movie(mid, ot, title, ty, dc, r, sy, ey) Ʌ IsOfGenre(‘Drama’, mid) Ʌ
StudioHasProduced(‘Paramount’, 'Cillian Murphy’)}

Query 3

Name of critics that have criticized a movie directed by Cristopher Nolan where Al Pacino acted

query elapsed time to join actors, movies and directors: ~5 minutes

```
select c.critic_name
from actor a, hasactedin h, movie m, hasdirected h2, criticreview c, director d
where a.actor_id = h.actor_id
     and h.movie_id = m.movie_id
     and m.movie_id = h2.movie_id
     and h2.director_id = d.director_id
     and m.movie_id = c.movie_id
     and d.name = 'Christopher Nolan'
     and a."name" = 'Al Pacino'
```

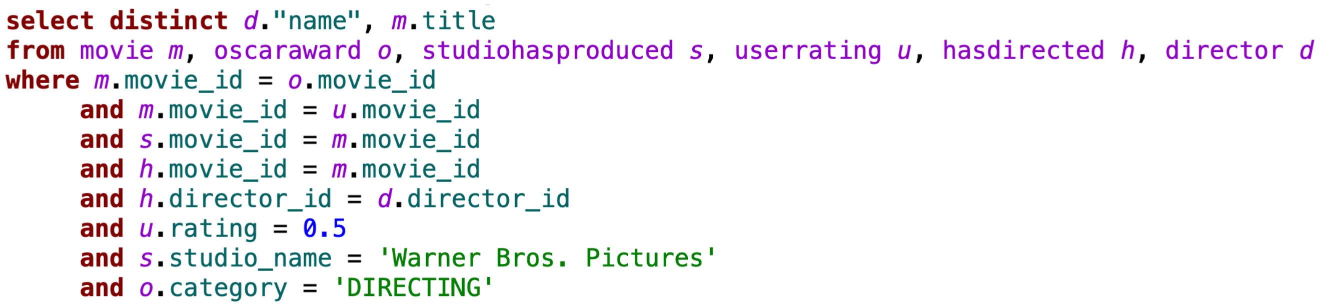


	A-Z critic_name
1	Rob Blackwelder
2	Harvey S. Karten
3	Richard Roeper
4	Robin Clifford
5	Jeanne Aufmuth
6	Ed Gonzalez
7	Kevin N. Laforest
8	Frank Swietek
9	Andy Weil
10	Steve Rhodes
11	Kirk Honeycutt

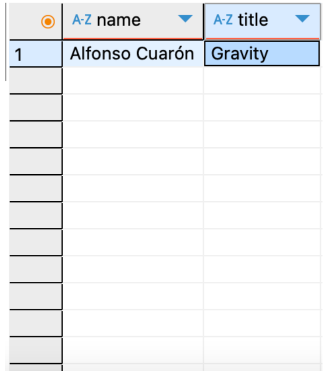
q = {(name) |
∃ aid, mid, by, dy, ot, ti, ty, dc, r, sy, ey, did, by2, dy2, cid, pub, co, rd, sc.
Actor(aid, ‘Al Pacino’, by, dy) ⋀ HasActedIn(aid, mid)
⋀ Movie(mid, ot, ti, ty, dc, r, sy, ey) ⋀ HasDirected(did, mid)
⋀ Director(did, ‘Cristopher Nolan’, by2, dy2)
⋀ CriticReview(cid, pub, name, co, rd, sc, mid)}

Query 4

Name of directors and movie titles of oscar-winning movies in the DIRECTING category, produced by Warner Bros, with at least one user rating of 0.5



```sql
select distinct d."name", m.title
from movie m, oscaraward o, studiohasproduced s, userrating u, hasdirected h, director d
where m.movie_id = o.movie_id
     and m.movie_id = u.movie_id
     and s.movie_id = m.movie_id
     and h.movie_id = m.movie_id
     and h.director_id = d.director_id
     and u.rating = 0.5
     and s.studio_name = 'Warner Bros. Pictures'
     and o.category = 'DIRECTING'
```



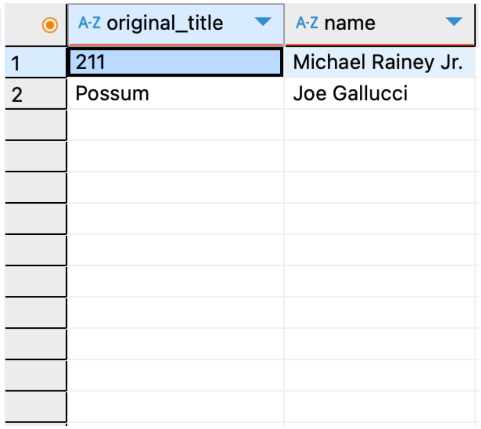
	A-Z name	A-Z title
1	Alfonso Cuarón	Gravity

q = {(name, title) |
∃ did, by, dy, mid, ot, ty, dc, r, sy, ey, yc, aw, rid, rating.
Director(did, name, by, dy) ⋀ HasDirected(did, mid)
⋀ Movie(mid, ot, title, ty, dc, r, sy, ey)
⋀ StudioHasProduced(‘Warner Bros. Pictures’, mid)
⋀ OscarAward(yc, ‘DIRECTING’, aw, mid)
⋀ UserRating(rid, 0,5, mid)}

Query 5

Title of horror movies and name of actors participating, born in the year 2000, with a critic review on rotten tomato, and a tension score of 0

```
select distinct m.original_title, a."name"
from movie m, genre g, isofgenre i, criticreview c, hasactedin h, actor a, energyscore e
where m.movie_id = i.movie_id
     and g."name" = i.genre_name
     and h.movie_id = m.movie_id
     and h.actor_id = a.actor_id
     and c.movie_id = m.movie_id
     and e.movie_id = m.movie_id
     and a.birth_year = 2000
     and g."name" = 'Horror'
     and e.tension = 0
```



	original_title	name
1	211	Michael Rainey Jr.
2	Possum	Joe Gallucci

q = {(title, name) |

∃ aid, by, dy, mid, ot, ty, dc, rt, sy, ey, cid, pub, cname, co, rd, sc, e, rh.

Actor(aid, name, by, dy) Ʌ HasActedIn(aid, mid)

Ʌ Movie(mid, ot, title, ty, dc, rt, sy, ey)

Ʌ IsOfGenre(‘Horror’, mid)

Ʌ CriticReview(cid, pub, cname, co, rd, sc, mid)

Ʌ EnergyScore(h, 0, e, rh, mid)}

…Thanks!

References

- Theoretical notions: LSDM course taught by prof. Antonella Poggi
- Github: Github repository
- Datasets: see README file in repository
- Guide link to install Pentaho Kettle on Apple Silicon