REEL DATA

FILM MEETS FIGURES

IMDb

Cua, Lander Peter E.
Gaba, Jacob Bryan B.
Gonzales, Mark Edward M.
Lee, Hylene Jules G.

01 **DATABASE SCHEMA**

Sources, schema, and denormalization strategy

02 **ETL PIPELINE**

Pipeline processes and utilized strategies

03 **OPTIMIZATION AND RESULTS**

MySQL scripts, findings, and query execution plan

04 **TESTING**

Metadata, content, and preprocessing validation

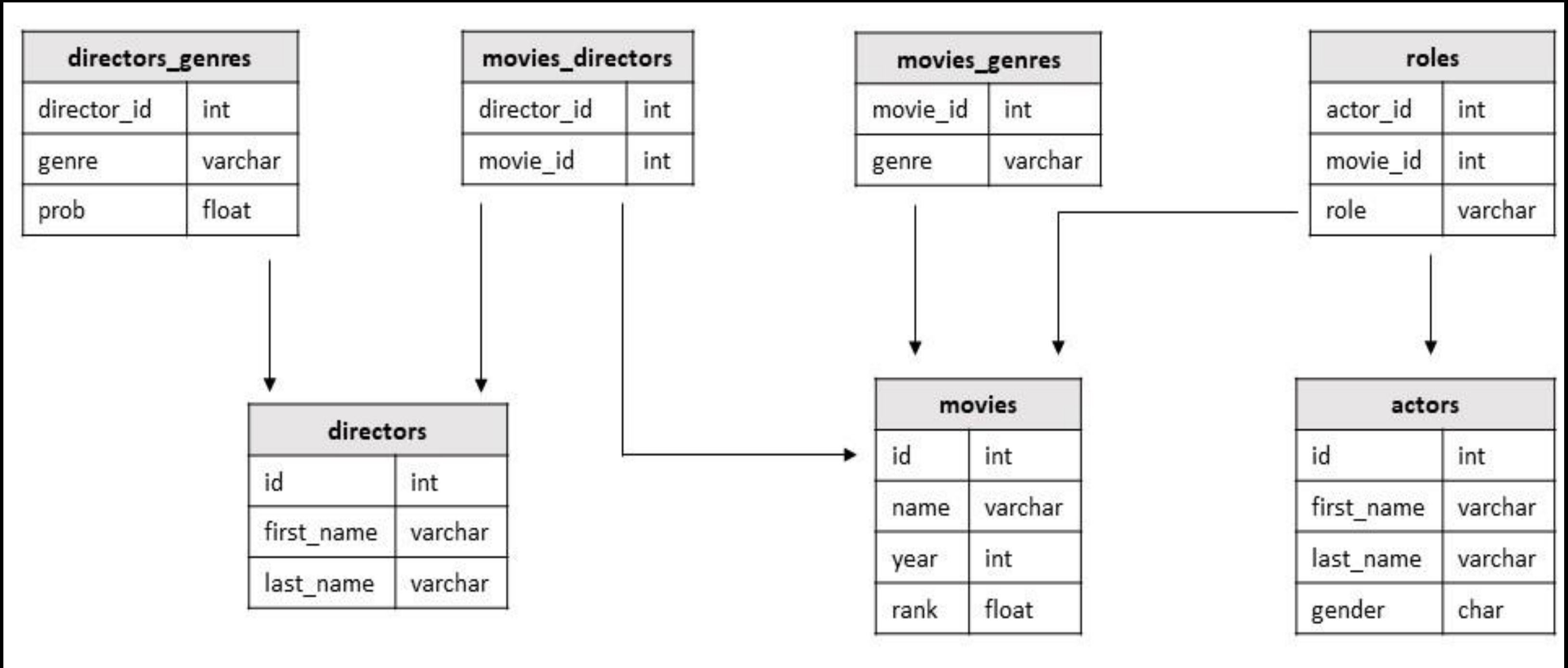
05 **CONCLUSION AND INSIGHTS**

Summary of findings and analyses

06 **DASHBOARD DEMO**

Data visualization via Tableau

Database Sources IMDb IJS



directors_genres	
director_id	int
genre	varchar
prob	float

movies_directors	
director_id	int
movie_id	int

movies_genres	
movie_id	int
genre	varchar

roles	
actor_id	int
movie_id	int
role	varchar

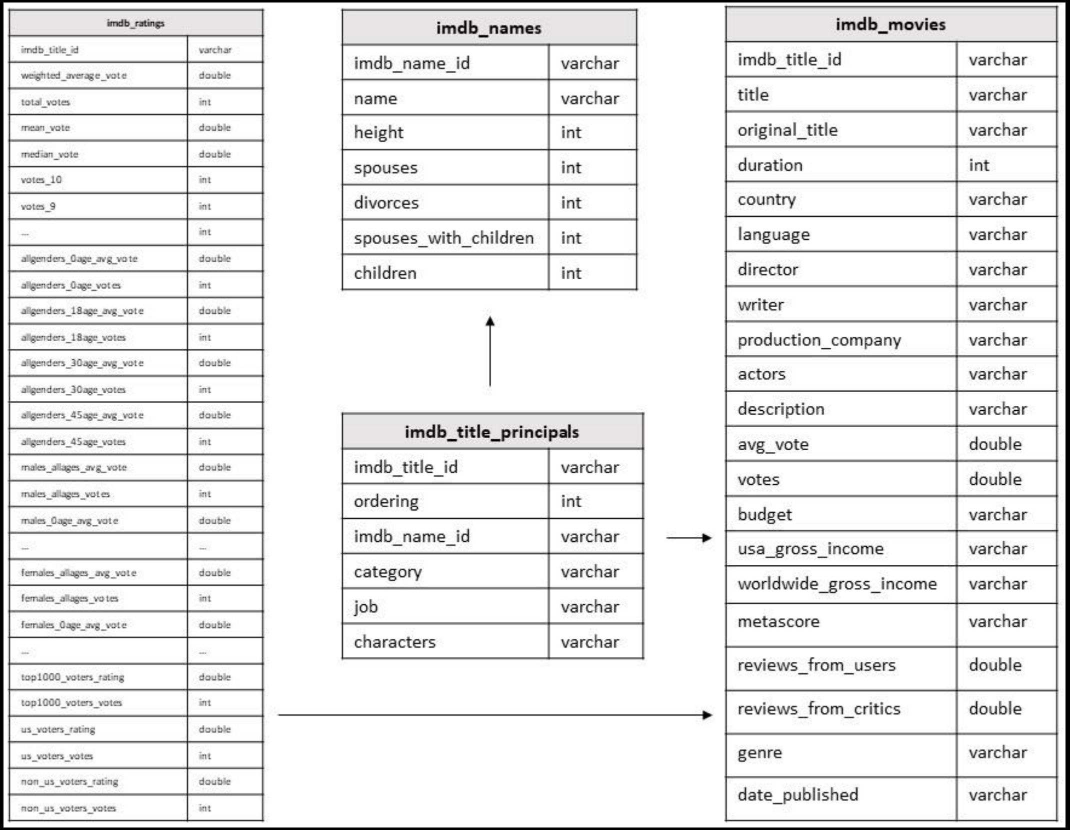
directors	
id	int
first_name	varchar
last_name	varchar

movies	
id	int
name	varchar
year	int
rank	float

actors	
id	int
first_name	varchar
last_name	varchar
gender	char

DATABASE SCHEMA

Database Sources IMDb Supplementary



imdb_ratings	
imdb_title_id	varchar
weighted_average_vote	double
total_votes	int
mean_vote	double
median_vote	double
votes_10	int
votes_9	int
...	int
allgenders_0age_avg_vote	double
allgenders_0age_votes	int
allgenders_18age_avg_vote	double
allgenders_18age_votes	int
allgenders_30age_avg_vote	double
allgenders_30age_votes	int
allgenders_45age_avg_vote	double
allgenders_45age_votes	int
males_allages_avg_vote	double
males_allages_votes	int
males_0age_avg_vote	double
...	...
females_allages_avg_vote	double
females_allages_votes	int
females_0age_avg_vote	double
...	...
top1000_voters_rating	double
top1000_voters_votes	int
us_voters_rating	double
us_voters_votes	int
non_us_voters_rating	double
non_us_voters_votes	int

imdb_names	
imdb_name_id	varchar
name	varchar
height	int
spouses	int
divorces	int
spouses_with_children	int
children	int

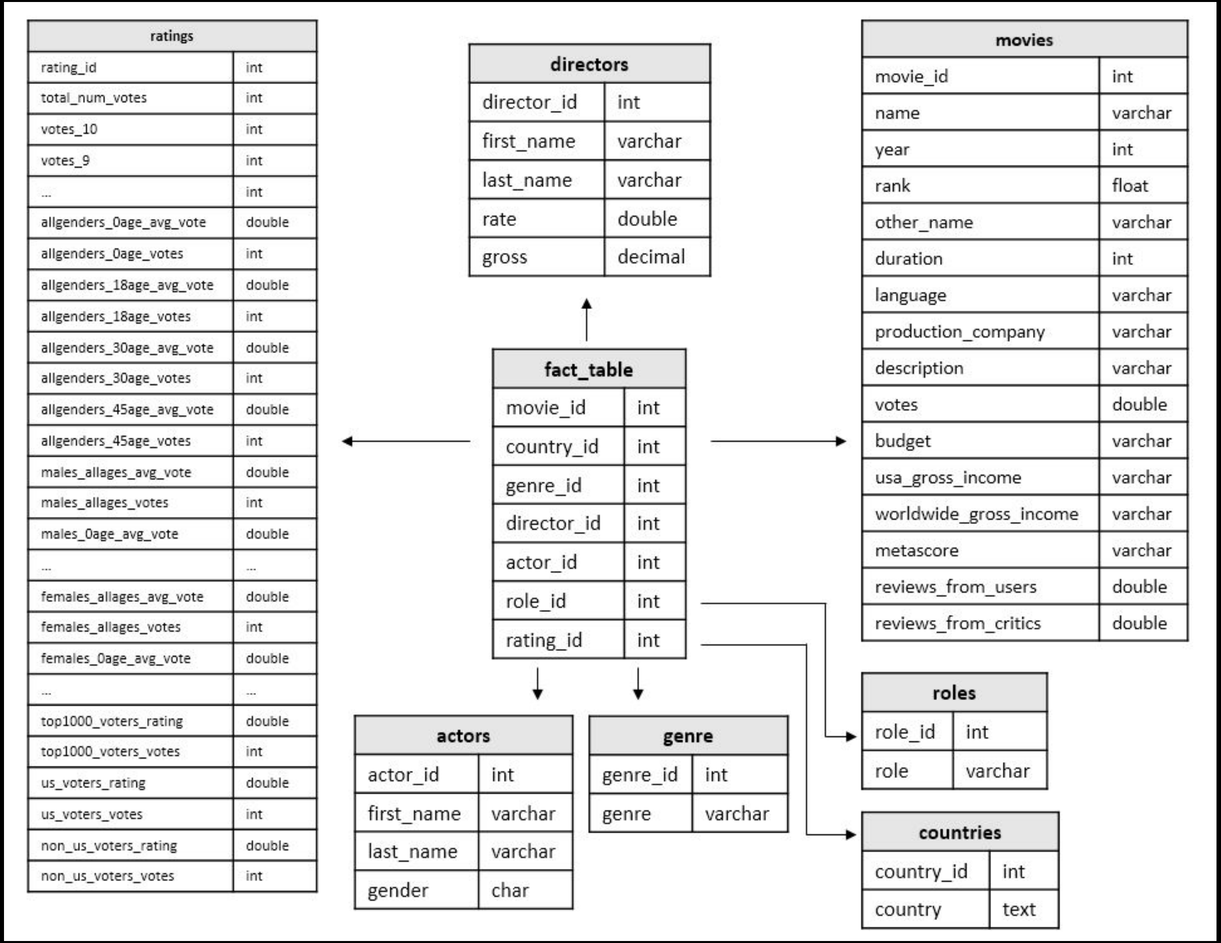
imdb_title_principals	
imdb_title_id	varchar
ordering	int
imdb_name_id	varchar
category	varchar
job	varchar
characters	varchar

imdb_movies	
imdb_title_id	varchar
title	varchar
original_title	varchar
duration	int
country	varchar
language	varchar
director	varchar
writer	varchar
production_company	varchar
actors	varchar
description	varchar
avg_vote	double
votes	double
budget	varchar
usa_gross_income	varchar
worldwide_gross_income	varchar
metascore	varchar
reviews_from_users	double
reviews_from_critics	double
genre	varchar
date_published	varchar

DATABASE SCHEMA

DATA WAREHOUSE

- ✔ 1 fact table
- ✔ 7 dimension tables
- ✔ Star schema



ratings	
rating_id	int
total_num_votes	int
votes_10	int
votes_9	int
...	int
allgenders_0age_avg_vote	double
allgenders_0age_votes	int
allgenders_18age_avg_vote	double
allgenders_18age_votes	int
allgenders_30age_avg_vote	double
allgenders_30age_votes	int
allgenders_45age_avg_vote	double
allgenders_45age_votes	int
males_allages_avg_vote	double
males_allages_votes	int
males_0age_avg_vote	double
...	...
females_allages_avg_vote	double
females_allages_votes	int
females_0age_avg_vote	double
...	...
top1000_voters_rating	double
top1000_voters_votes	int
us_voters_rating	double
us_voters_votes	int
non_us_voters_rating	double
non_us_voters_votes	int

directors	
director_id	int
first_name	varchar
last_name	varchar
rate	double
gross	decimal

fact_table	
movie_id	int
country_id	int
genre_id	int
director_id	int
actor_id	int
role_id	int
rating_id	int

movies	
movie_id	int
name	varchar
year	int
rank	float
other_name	varchar
duration	int
language	varchar
production_company	varchar
description	varchar
votes	double
budget	varchar
usa_gross_income	varchar
worldwide_gross_income	varchar
metascore	varchar
reviews_from_users	double
reviews_from_critics	double

actors	
actor_id	int
first_name	varchar
last_name	varchar
gender	char

genre	
genre_id	int
genre	varchar

roles	
role_id	int
role	varchar

countries	
country_id	int
country	text

DATABASE SCHEMA

STAR SCHEMA
JOINING TABLES

$$\Theta\left(\prod_{i=1}^{m} r_i\right)$$

$$\Theta\left(\prod_{i=1}^{p} s_i\right)$$

Normalized

$$O((\max r_i)^m)$$

Denormalized

$$O((\max s_i)^p)$$

$$\boxed{m > p}$$

Denormalization Strategies (Han, Kamber & Pei, 2012)

COLLAPSING

Applied to **movies, actors**, and **directors**

DIRECTOR DENORMALIZATION

Normalized:
movies(movie_id, name, …)
directors(director_id, first_name, last_name)
movies_directors (director_id, movie_id)

Denormalized:
fact_table(movie_id, **directors_id**)
directors(director_id, first_name, last_name)

RATING DENORMALIZATION

Normalized:
movies(movie_id, name, …)
ijs2supplementary(movie_id, imdb_title_id)
imdb_ratings(imdb_title_id, votes_10,
votes_9, …)

Denormalized:
fact_table(movie_id, country_id, genre_id,
directors_id, actors_id, role_id, **rating_id**)
ratings(ratings_id, votes_10, votes_9, …)

Collapsing strategy with auto-incrementing surrogate keys applied to **ratings**

VERTICAL PARTITIONING

Applied to **roles**, **genres**, and **countries**

- Makes up for the loss of conceptual distinction
- Optimization:

  Separate long strings to another table to decrease number of pages

COUNTRY DENORMALIZATION

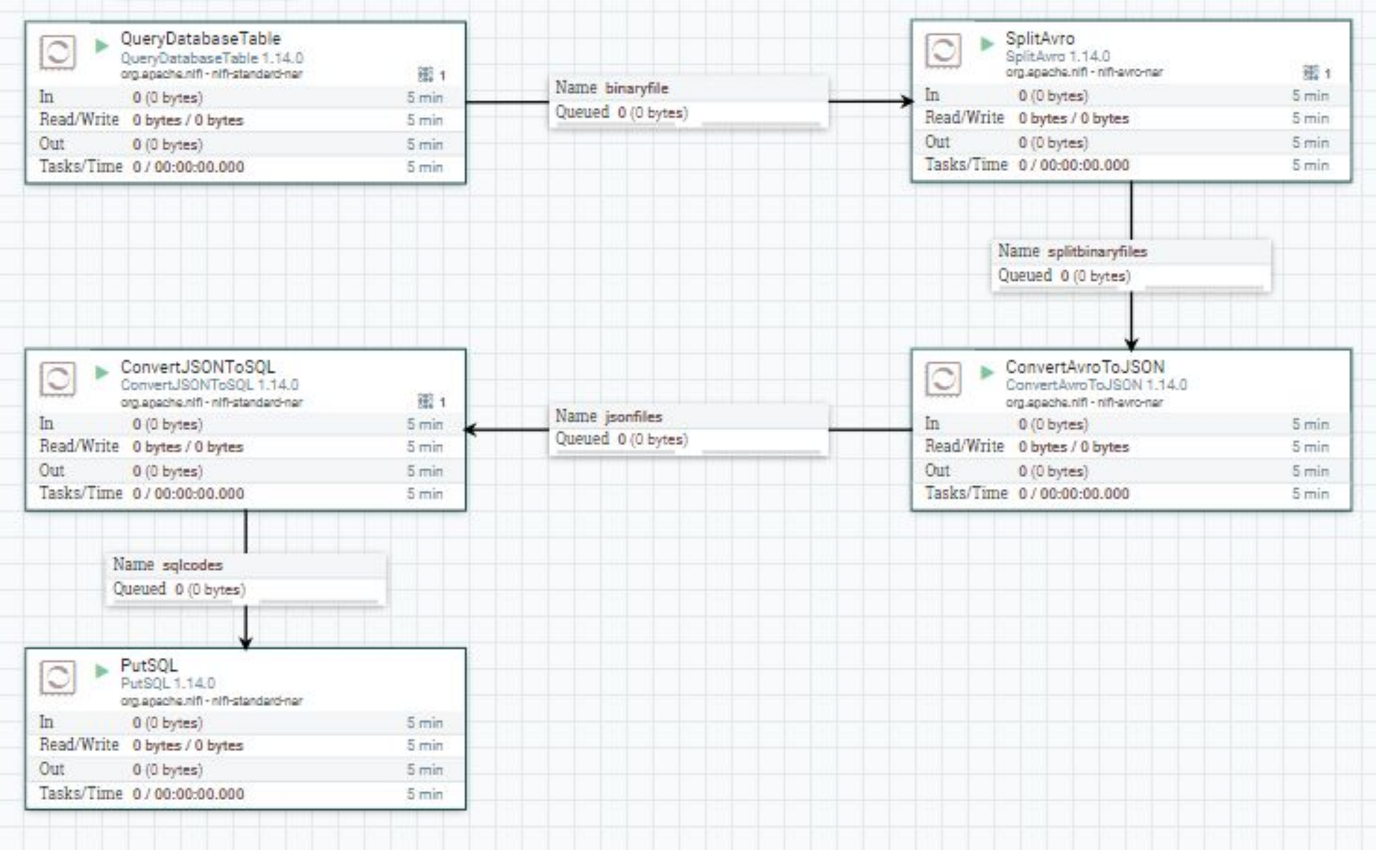
Normalized:
movies(movie_id, name, ..., country, ...)

Denormalized:
fact_table(movie_id, **country_id**, genre_id,
directors_id, actors_id, role_id)
movies(movie_id, name, ...)
country(country_id, country)

DATABASE SCHEMA

Creating Local Copies

IMDb IJS



QueryDatabaseTable
QueryDatabaseTable 1.14.0
org.apache.nifi - nifi-standard-nar
1

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Name binaryfile
Queued 0 (0 bytes)

SplitAvro
SplitAvro 1.14.0
org.apache.nifi - nifi-avro-nar
1

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Name splitbinaryfiles
Queued 0 (0 bytes)

ConvertAvroToJSON
ConvertAvroToJSON 1.14.0
org.apache.nifi - nifi-avro-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Name jsonfiles
Queued 0 (0 bytes)

ConvertJSONToSQL
ConvertJSONToSQL 1.14.0
org.apache.nifi - nifi-standard-nar
1

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Name sqlcodes
Queued 0 (0 bytes)

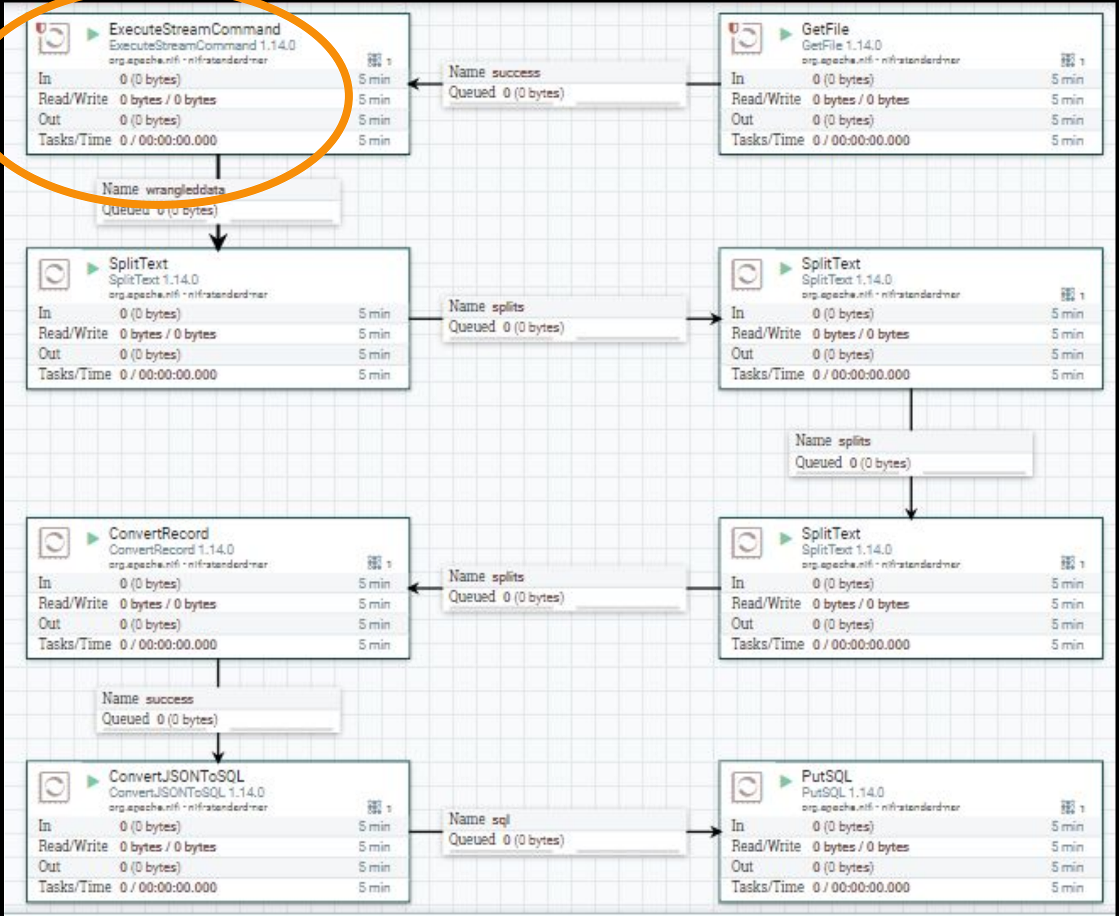
PutSQL
PutSQL 1.14.0
org.apache.nifi - nifi-standard-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

ETL PIPELINE

Creating Local Copies + Preliminary Data Wrangling

IMDb Supplementary



ExecuteStreamCommand
ExecuteStreamCommand 1.14.0
org.apache.nifi - nifi-standard-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Name success
Queued 0 (0 bytes)

GetFile
GetFile 1.14.0
org.apache.nifi - nifi-standard-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Name wrangleddata
Queued 0 (0 bytes)

SplitText
SplitText 1.14.0
org.apache.nifi - nifi-standard-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Name splits
Queued 0 (0 bytes)

SplitText
SplitText 1.14.0
org.apache.nifi - nifi-standard-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Name splits
Queued 0 (0 bytes)

ConvertRecord
ConvertRecord 1.14.0
org.apache.nifi - nifi-standard-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Name splits
Queued 0 (0 bytes)

SplitText
SplitText 1.14.0
org.apache.nifi - nifi-standard-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Name success
Queued 0 (0 bytes)

ConvertJSONToSQL
ConvertJSONToSQL 1.14.0
org.apache.nifi - nifi-standard-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Name sql
Queued 0 (0 bytes)

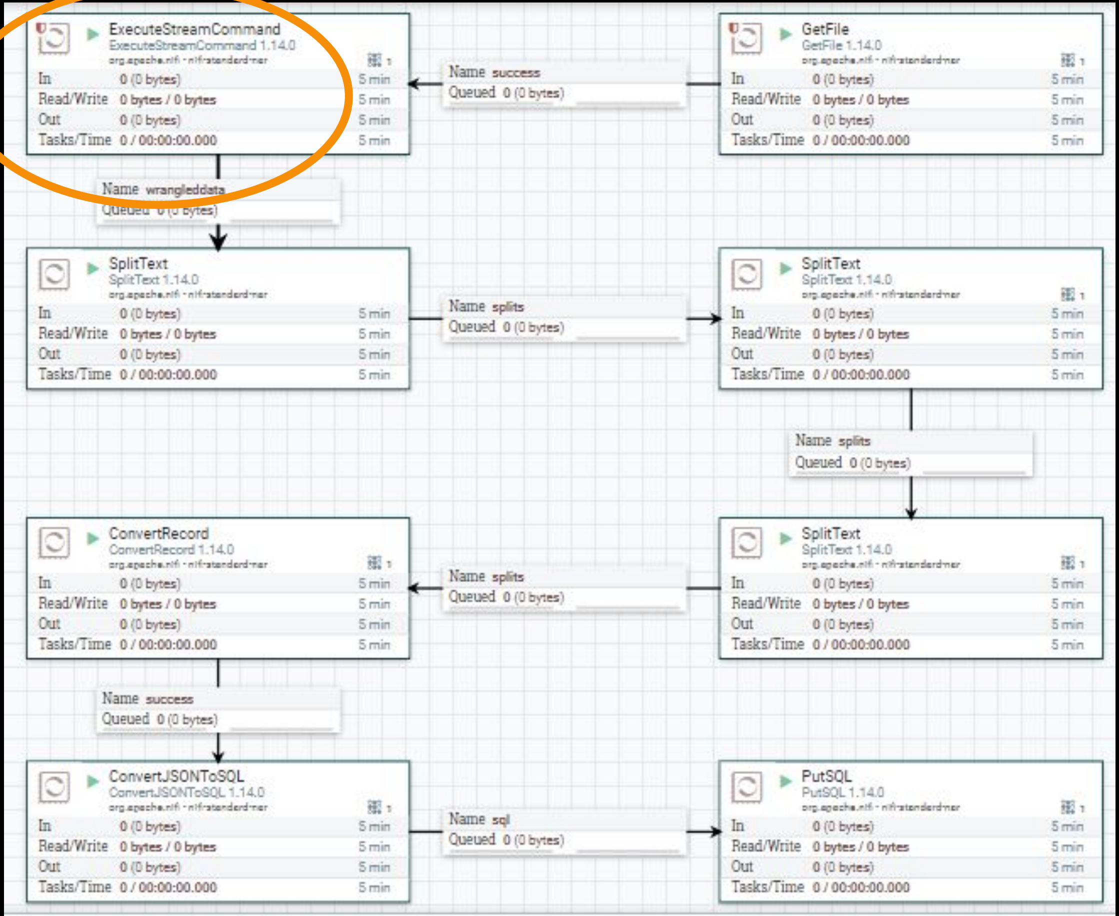
PutSQL
PutSQL 1.14.0
org.apache.nifi - nifi-standard-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

ETL PIPELINE

Creating Local Copies + Preliminary Data Wrangling

IMDb Supplementary



ExecuteStreamCommand
ExecuteStreamCommand 1.14.0
org.apache.nifi - nifi-standard-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min

Name success
Queued 0 (0 bytes)

GetFile
GetFile 1.14.0
org.apache.nifi - nifi-standard-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min

Name wrangleddata
Queued 0 (0 bytes)

SplitText
SplitText 1.14.0
org.apache.nifi - nifi-standard-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min

Name splits
Queued 0 (0 bytes)

SplitText
SplitText 1.14.0
org.apache.nifi - nifi-standard-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min

Name splits
Queued 0 (0 bytes)

ConvertRecord
ConvertRecord 1.14.0
org.apache.nifi - nifi-standard-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min

Name splits
Queued 0 (0 bytes)

SplitText
SplitText 1.14.0
org.apache.nifi - nifi-standard-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min

Name success
Queued 0 (0 bytes)

ConvertJSONToSQL
ConvertJSONToSQL 1.14.0
org.apache.nifi - nifi-standard-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min

Name sql
Queued 0 (0 bytes)

PutSQL
PutSQL 1.14.0
org.apache.nifi - nifi-standard-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min

DATE STANDARDIZATION

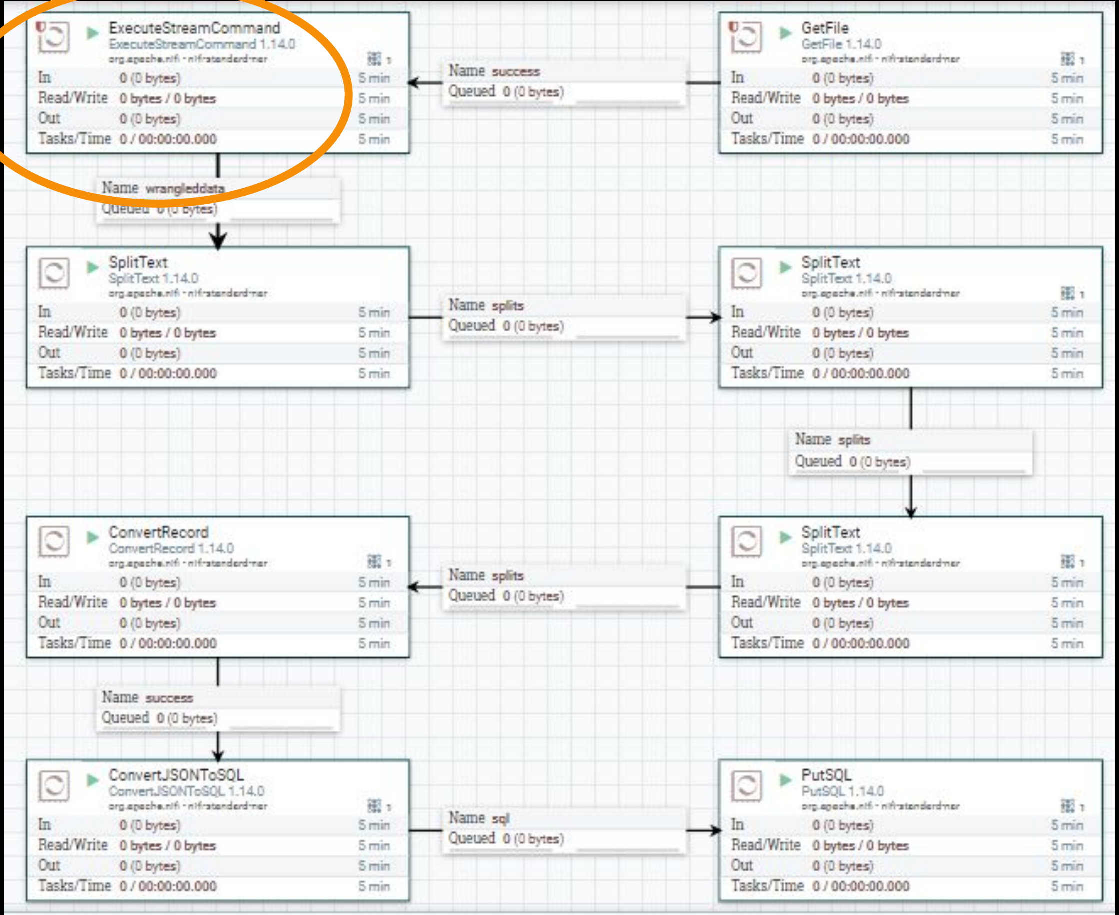
Non-standardized:
10-09-1894
26/12/1906
1913

Standardized:
1894-09-10
1906-12-26
1913-01-01

ETL PIPELINE

Creating Local Copies + Preliminary Data Wrangling

IMDb Supplementary



ExecuteStreamCommand
ExecuteStreamCommand 1.14.0
org.apache.nifi - nifi-standard-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Name success
Queued 0 (0 bytes)

GetFile
GetFile 1.14.0
org.apache.nifi - nifi-standard-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Name wrangleddata
Queued 0 (0 bytes)

SplitText
SplitText 1.14.0
org.apache.nifi - nifi-standard-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Name splits
Queued 0 (0 bytes)

SplitText
SplitText 1.14.0
org.apache.nifi - nifi-standard-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Name splits
Queued 0 (0 bytes)

ConvertRecord
ConvertRecord 1.14.0
org.apache.nifi - nifi-standard-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Name splits
Queued 0 (0 bytes)

SplitText
SplitText 1.14.0
org.apache.nifi - nifi-standard-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Name success
Queued 0 (0 bytes)

ConvertJSONToSQL
ConvertJSONToSQL 1.14.0
org.apache.nifi - nifi-standard-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Name sql
Queued 0 (0 bytes)

PutSQL
PutSQL 1.14.0
org.apache.nifi - nifi-standard-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

DATE STANDARDIZATION

Non-standardized:
10-09-1894
26/12/1906
1913

Standardized:
1894-09-10
1906-12-26
1913-01-01

COUNTRY STANDARDIZATION

Non-standardized:
France, Germany, Japan

Exploded into separate rows:
France
Germany
Japan

ETL PIPELINE

Standardizing Movie Titles

Godfather, The (1972) → The Godfather

Standardizing Movie Titles

Godfather, The (1972) → The Godfather

```sql
UPDATE movies SET `name` =
RTRIM(REVERSE(SUBSTRING(REVERSE(`name`),
LOCATE(" ",REVERSE(`name`)))))
WHERE `name` IN (
   SELECT `name`
   FROM (
      SELECT `name`
      FROM movies
         WHERE name LIKE "%(%)"
   ) AS tmp
);

UPDATE movies SET `name` =
RTRIM(REVERSE(SUBSTRING(REVERSE(`name`),
LOCATE(" ",REVERSE(`name`)))))
WHERE `name` IN (
   SELECT `name`
   FROM (
      SELECT `name`
      FROM movies
         WHERE name LIKE "%, The"
   ) AS tmp
);
```

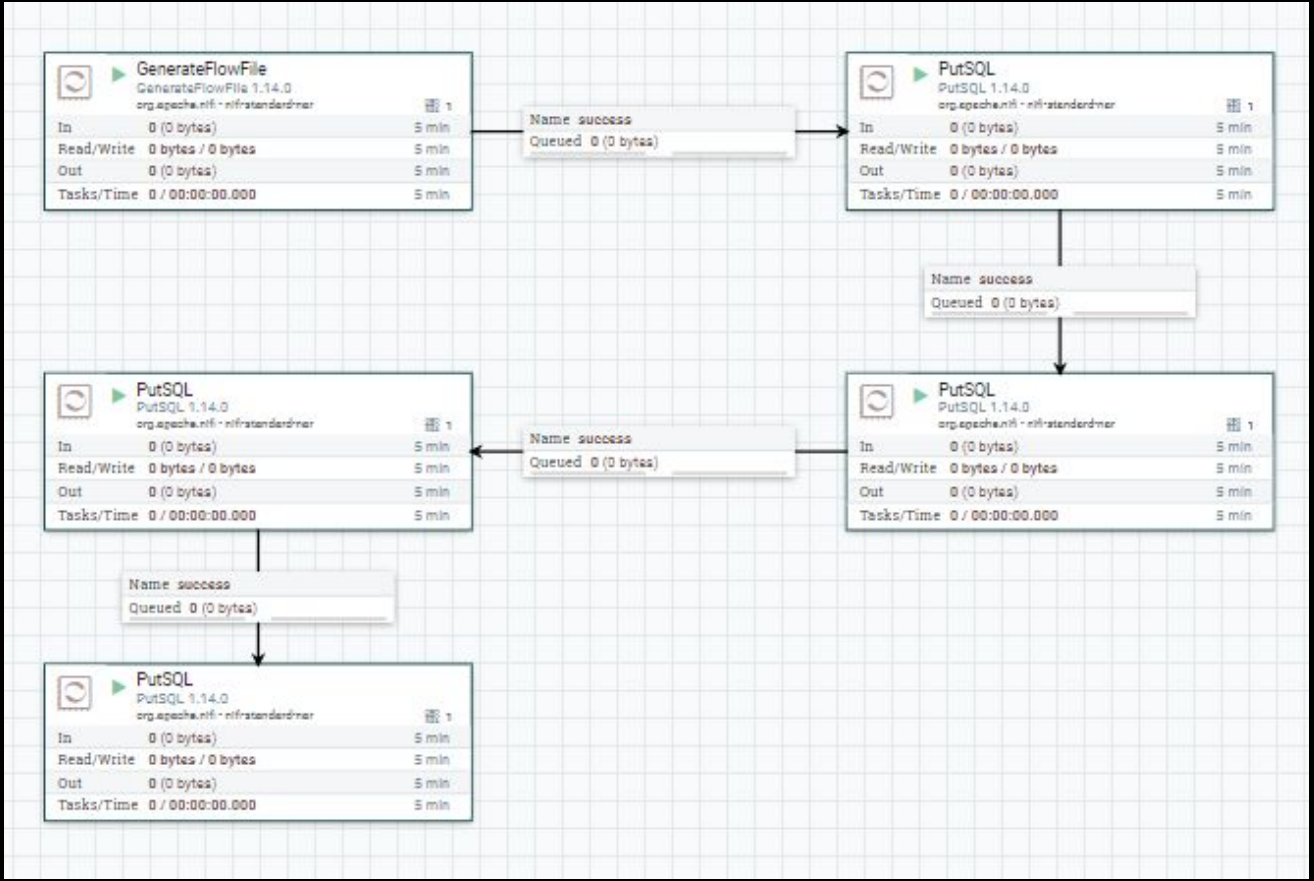
```sql
UPDATE movies SET `name` = CONCAT("The ",
   `name`)
WHERE `name` IN (
   SELECT `name`
   FROM (
        SELECT `name`
        FROM movies
        WHERE name LIKE "%,"
   ) AS tmp
);


UPDATE movies SET `name` = SUBSTRING(`name`,
    1, CHAR_LENGTH(`name`) - 1)
WHERE `name` IN (
    SELECT `name`
   FROM (
      SELECT `name`
      FROM movies
      WHERE name LIKE "%,"
   ) AS tmp
);
```

ETL PIPELINE

Standardizing Movie Titles

Godfather, The (1972) → The Godfather



GenerateFlowFile
GenerateFlowFile 1.14.0
org.apache.nifi - nifi-standard-nar 1

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Name success
Queued 0 (0 bytes)

PutSQL
PutSQL 1.14.0
org.apache.nifi - nifi-standard-nar 1

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Name success
Queued 0 (0 bytes)

PutSQL
PutSQL 1.14.0
org.apache.nifi - nifi-standard-nar 1

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Name success
Queued 0 (0 bytes)

PutSQL
PutSQL 1.14.0
org.apache.nifi - nifi-standard-nar 1

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

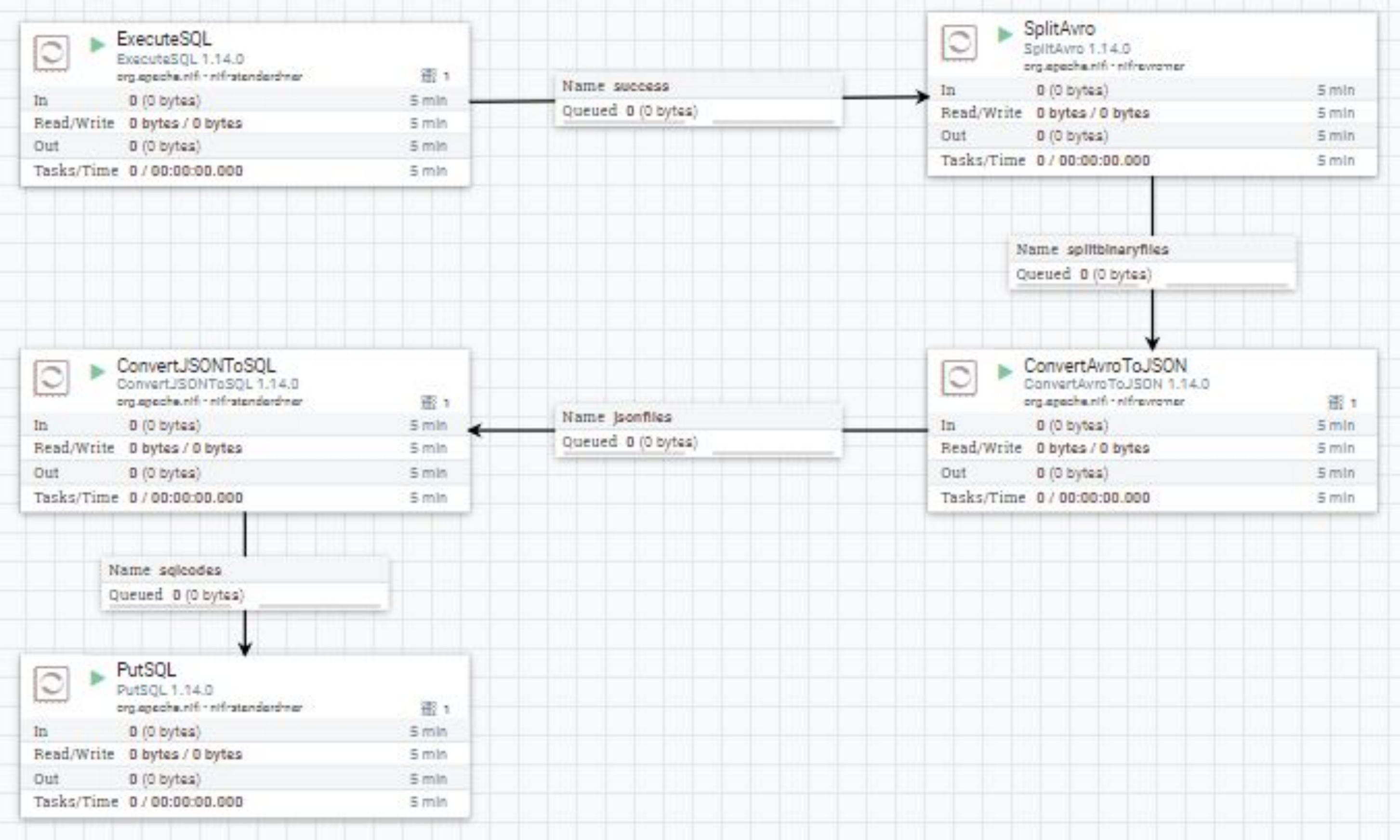
Name success
Queued 0 (0 bytes)

PutSQL
PutSQL 1.14.0
org.apache.nifi - nifi-standard-nar 1

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

ETL PIPELINE

Transferring to the Data Warehouse



```
ExecuteSQL
ExecuteSQL 1.14.0
org.apache.nifi - nifi-standard-nar                1
In            0 (0 bytes)                          5 min
Read/Write    0 bytes / 0 bytes                    5 min
Out           0 (0 bytes)                          5 min
Tasks/Time    0 / 00:00:00.000                     5 min
```

```
Name  success
Queued  0 (0 bytes)
```

```
SplitAvro
SplitAvro 1.14.0
org.apache.nifi - nifi-avro-nar
In            0 (0 bytes)                          5 min
Read/Write    0 bytes / 0 bytes                    5 min
Out           0 (0 bytes)                          5 min
Tasks/Time    0 / 00:00:00.000                     5 min
```

```
Name  splitbinaryfiles
Queued  0 (0 bytes)
```

```
ConvertAvroToJSON
ConvertAvroToJSON 1.14.0
org.apache.nifi - nifi-avro-nar                    1
In            0 (0 bytes)                          5 min
Read/Write    0 bytes / 0 bytes                    5 min
Out           0 (0 bytes)                          5 min
Tasks/Time    0 / 00:00:00.000                     5 min
```

```
Name  jsonfiles
Queued  0 (0 bytes)
```

```
ConvertJSONToSQL
ConvertJSONToSQL 1.14.0
org.apache.nifi - nifi-standard-nar                1
In            0 (0 bytes)                          5 min
Read/Write    0 bytes / 0 bytes                    5 min
Out           0 (0 bytes)                          5 min
Tasks/Time    0 / 00:00:00.000                     5 min
```

```
Name  sqlcodes
Queued  0 (0 bytes)
```

```
PutSQL
PutSQL 1.14.0
org.apache.nifi - nifi-standard-nar                1
In            0 (0 bytes)                          5 min
Read/Write    0 bytes / 0 bytes                    5 min
Out           0 (0 bytes)                          5 min
Tasks/Time    0 / 00:00:00.000                     5 min
```

ETL PIPELINE

Constructing the Fact Table

```
SELECT movie_id, country_id, genre_id,
  director_id, actor_id, role_id, rating_id
FROM movies
LEFT JOIN (movies_directors
JOIN directors
  ON directors.director_id =
    movies_directors.director_id)
  ON movies.movie_id =
    movies_directors.movie_id
LEFT JOIN (movies_actors
JOIN actors
  ON actors.actor_id =
    movies_actors.actor_id)
  ON movies.movie_id =
    movies_actors.movie_id
```

```
LEFT JOIN genres
  ON movies.movie_id =
    movies_genres.movie_id
LEFT JOIN roles
  ON movies.movie_id = roles.movie_id
  AND actors.actor_id = roles.actor_id
LEFT JOIN (ijs2supplementary
JOIN imdb_movies
  ON ijs2supplementary.imdb_title_id =
    imdb_movies.imdb_title_id
JOIN imdb_ratings
  ON ijs2supplementary.imdb_title_id =
    imdb_ratings.imdb_title_id)
  ON movies.movie_id =
    ijs2supplementary.movie_id;
```

ETL PIPELINE

Data Warehouse Size

Name	Rows
fact_table	6,617,850
actors	815,842
countries	184
directors	85,731
genres	21
movies	381,762
ratings	35,428
roles	2,513,767

ETL PIPELINE

MACHINE SPECIFICATIONS

- ✔ **Processor:** AMD Ryzen 5 5600x 6-Core Processor
- ✔ **Processor Base Frequency:** 3.70 GHz
- ✔ **Memory:** 16GB DDR4 2667MHz
- ✔ **Disk:** 1TB Hard Disk Drive

EXPERIMENTS

First Version	normalized schema
Second Version	without indexes
Third Version	with indexes on attributes involved in join operations
Fourth Version	with composite indexes on attributes involved in join, group by, and order by
Fifth Version	revised, more optimized, equivalent queries (if possible)

OPTIMIZATION AND RESULTS

QUERY PROCESSING

Q1. For each genre, what is the total number of movie votes per year and decade?

```
SELECT CONCAT(FLOOR(m.year/10) * 10, 's') AS
   decade_released, m.year, g.genre,
   (SUM(r.votes_10) + SUM(r.votes_9) +
   SUM(r.votes_8) + SUM(r.votes_7) +
   SUM(r.votes_6) + SUM(r.votes_5) +
   SUM(r.votes_4) + SUM(r.votes_3) +
   SUM(r.votes_2) + SUM(r.votes_1)) AS
   total_num_votes,
   SUM(r.votes_10) AS votes_10, SUM(r.votes_9)
   AS votes_9, SUM(r.votes_8) AS votes_8,
   SUM(r.votes_7) AS votes_7, SUM(r.votes_6) AS
   votes_6, SUM(r.votes_5) AS votes_5,
   SUM(r.votes_4) AS votes_4, SUM(r.votes_3) AS
   votes_3, SUM(r.votes_2) AS votes_2,
   SUM(r.votes_1) AS votes_1
FROM movies m
JOIN genres g ON g.movie_id = m.movie_id
JOIN ijs2supplementary i ON i.movie_id =
   m.movie_id
JOIN imdb_ratings r ON r.imdb_title_id =
   i.imdb_title_id
GROUP BY decade_released, m.year, g.genre
   WITH ROLLUP
ORDER BY decade_released, m.year, g.genre;
```

NORMALIZED SCHEMA

```
SELECT CONCAT(FLOOR(m.year/10) * 10, 's') AS
   decade_released, m.year, g.genre,
   SUM(r.total_num_votes) AS total_votes,
   SUM(r.votes_10) AS votes_10, SUM(r.votes_9)
   AS votes_9, SUM(r.votes_8) AS votes_8,
   SUM(r.votes_7) AS votes_7, SUM(r.votes_6) AS
   votes_6, SUM(r.votes_5) AS votes_5,
   SUM(r.votes_4) AS votes_4, SUM(r.votes_3) AS
   votes_3, SUM(r.votes_2) AS votes_2,
   SUM(r.votes_1) AS votes_1
FROM fact_table f
JOIN movies m ON m.movie_id = f.movie_id
JOIN ratings r ON r.rating_id = f.rating_id
JOIN genres g ON g.genre_id = f.genre_id
GROUP BY decade_released, m.year, g.genre
   WITH ROLLUP
ORDER BY decade_released, m.year, g.genre;
```

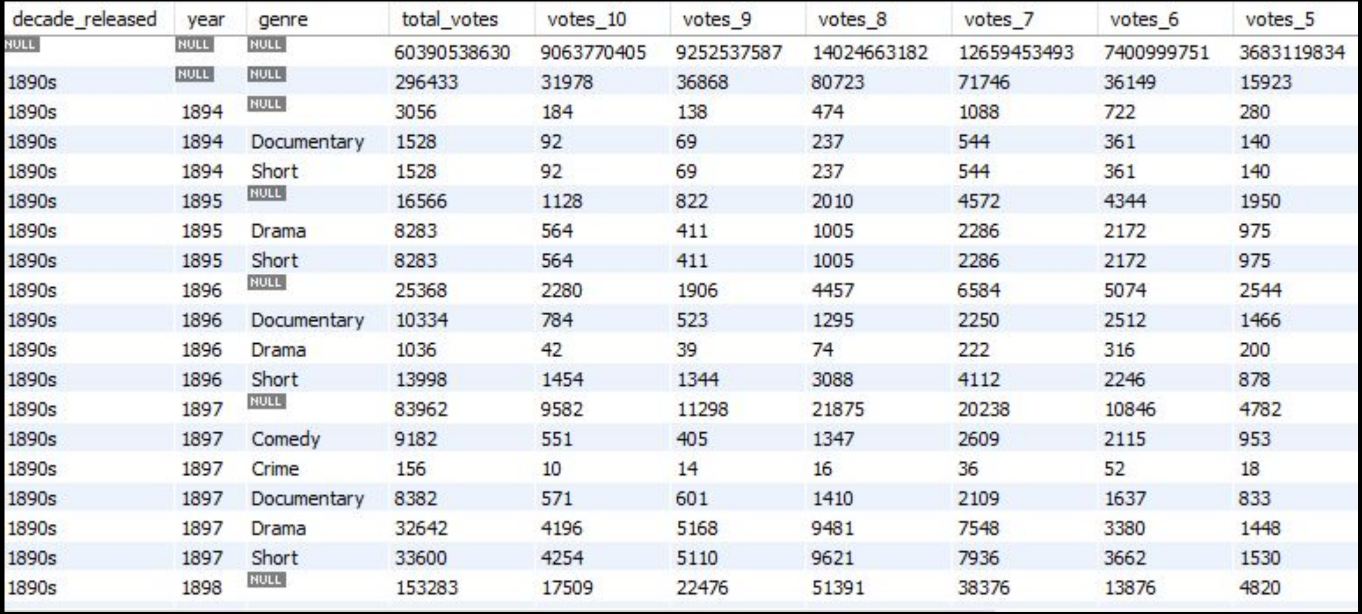
DENORMALIZED SCHEMA

OPTIMIZATION AND RESULTS

QUERY PROCESSING

Q1. For each genre, what is the total number of movie votes per year and decade?

Output



decade_released	year	genre	total_votes	votes_10	votes_9	votes_8	votes_7	votes_6	votes_5
NULL	NULL	NULL	60390538630	9063770405	9252537587	14024663182	12659453493	7400999751	3683119834
1890s	NULL	NULL	296433	31978	36868	80723	71746	36149	15923
1890s	1894	NULL	3056	184	138	474	1088	722	280
1890s	1894	Documentary	1528	92	69	237	544	361	140
1890s	1894	Short	1528	92	69	237	544	361	140
1890s	1895	NULL	16566	1128	822	2010	4572	4344	1950
1890s	1895	Drama	8283	564	411	1005	2286	2172	975
1890s	1895	Short	8283	564	411	1005	2286	2172	975
1890s	1896	NULL	25368	2280	1906	4457	6584	5074	2544
1890s	1896	Documentary	10334	784	523	1295	2250	2512	1466
1890s	1896	Drama	1036	42	39	74	222	316	200
1890s	1896	Short	13998	1454	1344	3088	4112	2246	878
1890s	1897	NULL	83962	9582	11298	21875	20238	10846	4782
1890s	1897	Comedy	9182	551	405	1347	2609	2115	953
1890s	1897	Crime	156	10	14	16	36	52	18
1890s	1897	Documentary	8382	571	601	1410	2109	1637	833
1890s	1897	Drama	32642	4196	5168	9481	7548	3380	1448
1890s	1897	Short	33600	4254	5110	9621	7936	3662	1530
1890s	1898	NULL	153283	17509	22476	51391	38376	13876	4820

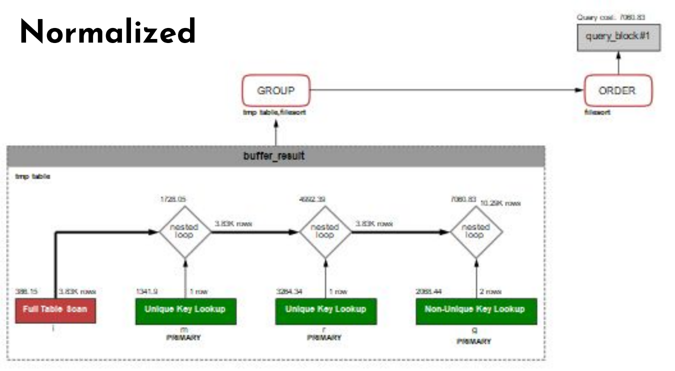
Rows: 1992
Columns: 14

OPTIMIZATION AND RESULTS

Q1. For each genre, what is the total number of movie votes per year and decade?

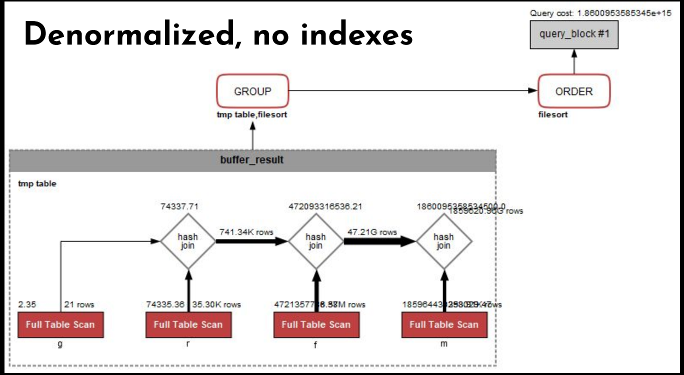
Query Execution Plans

Normalized



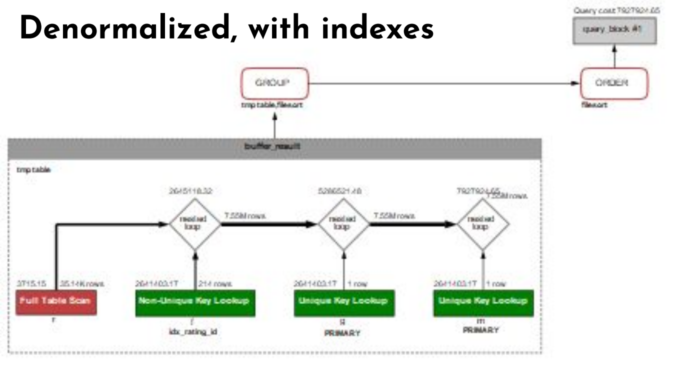
Execution Time: 23.400 seconds

Denormalized, no indexes



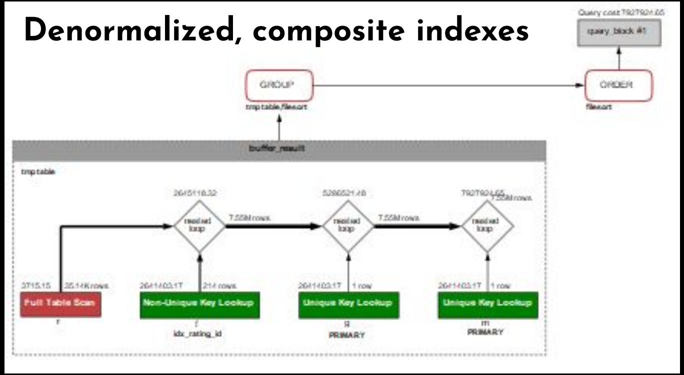
Execution Time: 9.344 seconds

Denormalized, with indexes



Execution Time: 15.453 seconds

Denormalized, composite indexes



Execution Time: 16.516 seconds

OPTIMIZATION AND RESULTS ▲

QUERY PROCESSING

Q2. For each year of each decade, what are the total number of male and female votes per genre?

```
SELECT CONCAT(FLOOR(m.year/10) * 10, 's') AS
   decade_released, m.year, g.genre,
   (SUM(r.votes_10) + SUM(r.votes_9) +
   SUM(r.votes_8) +
   SUM(r.votes_7) + SUM(r.votes_6) +
   SUM(r.votes_5) +
   SUM(r.votes_4) + SUM(r.votes_3) +
   SUM(r.votes_2) +
   SUM(r.votes_1)) AS total_num_votes,
   SUM(r.males_allages_votes) AS votes_male,
   SUM(r.females_allages_votes) AS votes_female
FROM movies m
JOIN genres g ON g.movie_id = m.movie_id
JOIN ijs2supplementary i ON i.movie_id =
   m.movie_id
JOIN imdb_ratings r ON r.imdb_title_id =
   i.imdb_title_id
GROUP BY decade_released, m.year, g.genre
   WITH ROLLUP
ORDER BY decade_released, m.year, g.genre;
```

NORMALIZED SCHEMA

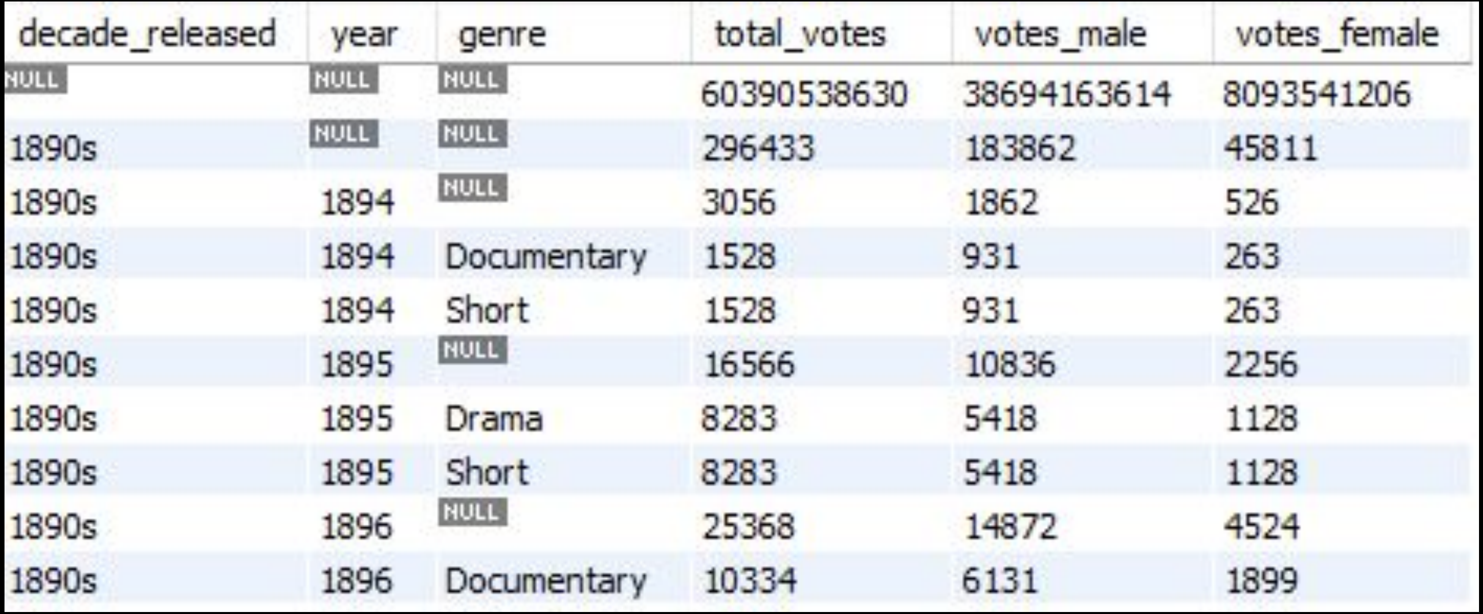
```
SELECT CONCAT(FLOOR(m.year/10) * 10, 's')
   AS decade_released, m.year, g.genre,
   SUM(r.total_num_votes) AS total_votes,
   SUM(r.males_allages_votes) AS votes_male,
   SUM(r.females_allages_votes) AS
   votes_female
FROM fact_table f
JOIN movies m ON m.movie_id = f.movie_id
JOIN genres g ON g.genre_id = f.genre_id
JOIN ratings r ON r.rating_id = f.rating_id
GROUP BY decade_released, m.year, g.genre
   WITH ROLLUP
ORDER BY decade_released, m.year, g.genre;
```

DENORMALIZED SCHEMA

QUERY PROCESSING

Q2. For each year of each decade, what are the total number of male and female votes per genre?

Output



decade_released	year	genre	total_votes	votes_male	votes_female
NULL	NULL	NULL	60390538630	38694163614	8093541206
1890s	NULL	NULL	296433	183862	45811
1890s	1894	NULL	3056	1862	526
1890s	1894	Documentary	1528	931	263
1890s	1894	Short	1528	931	263
1890s	1895	NULL	16566	10836	2256
1890s	1895	Drama	8283	5418	1128
1890s	1895	Short	8283	5418	1128
1890s	1896	NULL	25368	14872	4524
1890s	1896	Documentary	10334	6131	1899

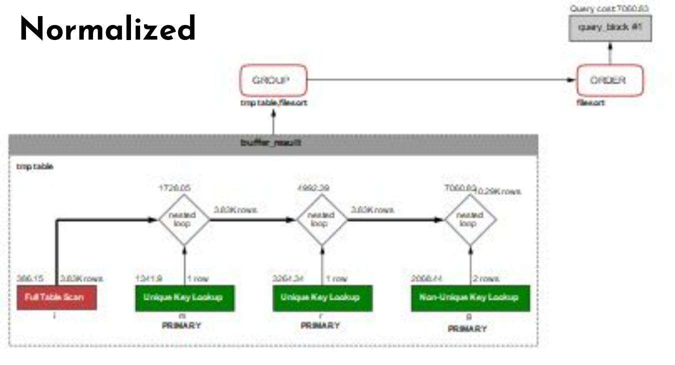
Rows: 1992
Columns: 6

OPTIMIZATION AND RESULTS

Q2. For each year of each decade, what are the total number of male and female votes per genre?

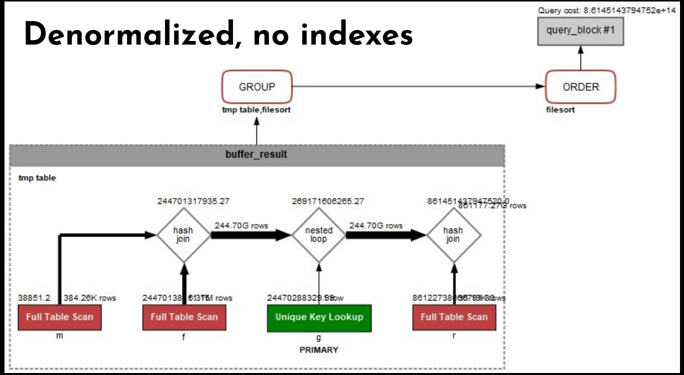
Query Execution Plans

Normalized



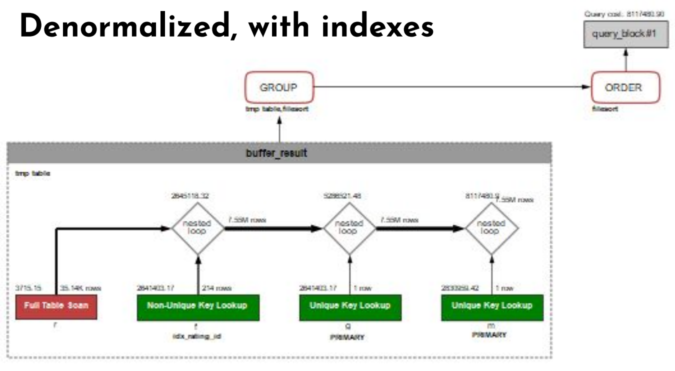
Execution Time: 12.5 seconds

Denormalized, no indexes



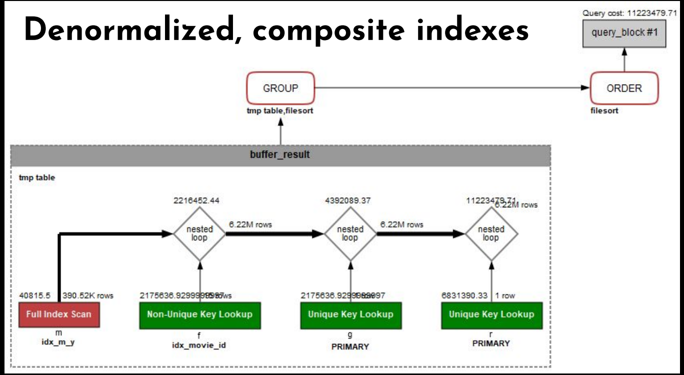
Execution Time: 9.359 seconds

Denormalized, with indexes



Execution Time: 12.656 seconds

Denormalized, composite indexes



Execution Time: 13.375 seconds

OPTIMIZATION AND RESULTS ▲

QUERY PROCESSING

Q3. Who are the top 15 actors with the most number of movie appearances for the action genre?

```
SELECT name, Num_roles
FROM (
   SELECT a.name, COUNT(m.movie_id)
      AS Num_roles,
      RANK() OVER
         (PARTITION BY g.genre
          ORDER BY COUNT(m.movie_id) DESC)
      AS Num_roles_rank
   FROM movies m
   JOIN genres g ON g.movie_id = m.movie_id
   JOIN roles i ON i.movie_id = m.movie_id
   JOIN actors a ON a.actor_id = i.actor_id
   WHERE g.genre = "Action"
   GROUP BY g.genre, a.name
) AS Role_rank
WHERE Num_roles_rank <= 15;
```

NORMALIZED SCHEMA

```
SELECT name, Num_roles
FROM (
   SELECT CONCAT(a.last_name, ", ",
       a.first_name) AS name,  COUNT(m.movie_id)
       AS Num_roles,
       RANK() OVER
          (PARTITION BY g.genre
           ORDER BY COUNT(m.movie_id) DESC)
       AS Num_roles_rank
    FROM fact_table f
    JOIN genres g ON f.genre_id = g.genre_id
    JOIN actors a ON f.actor_id = a.actor_id
    JOIN movies m ON f.movie_id = m.movie_id
    WHERE g.genre = "Action"
    GROUP BY g.genre, a.last_name, a.first_name
) AS Role_rank
WHERE Num_roles_rank <= 15;
```

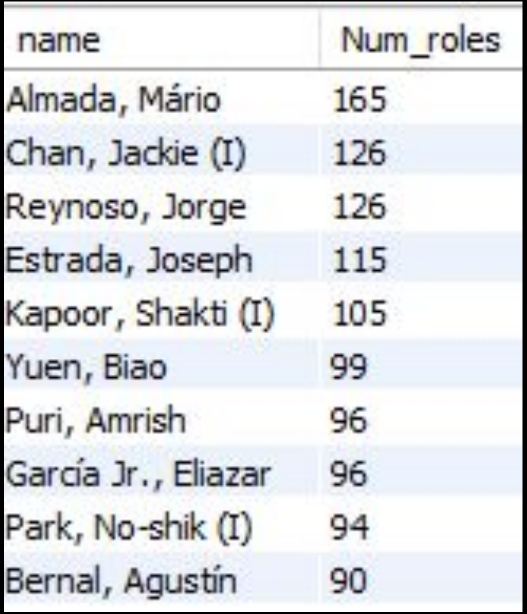
DENORMALIZED SCHEMA

OPTIMIZATION AND RESULTS

QUERY PROCESSING

Q3. Who are the top 15 actors with the most number of movie appearances for the action genre?

Output



name	Num_roles
Almada, Mário	165
Chan, Jackie (I)	126
Reynoso, Jorge	126
Estrada, Joseph	115
Kapoor, Shakti (I)	105
Yuen, Biao	99
Puri, Amrish	96
García Jr., Eliazar	96
Park, No-shik (I)	94
Bernal, Agustín	90

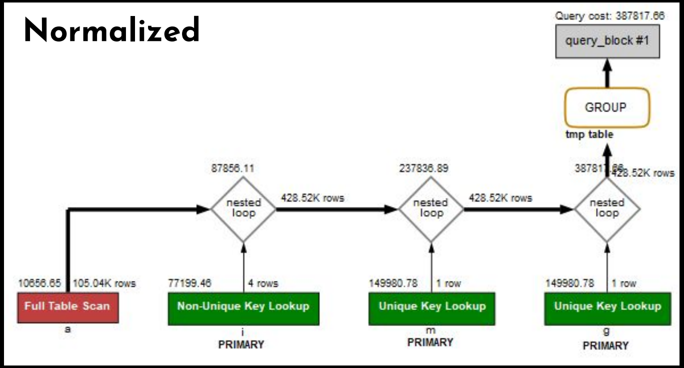
Rows: 15
Columns: 2

OPTIMIZATION AND RESULTS

Q3. Who are the top 15 actors with the most number of movie appearances for the action genre?

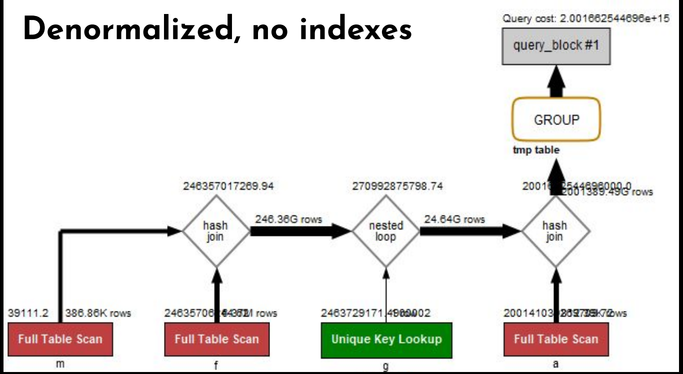
Query Execution Plans

Normalized



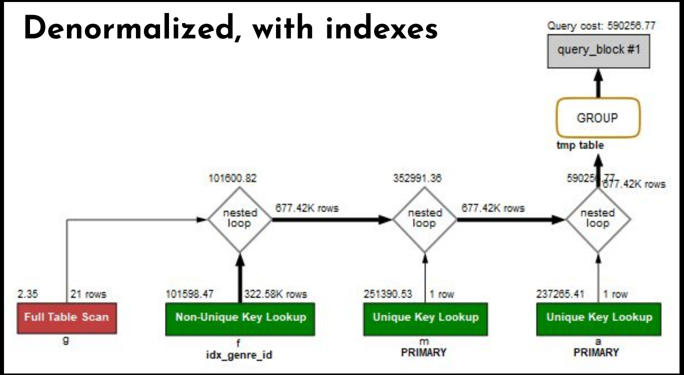
Execution Time: 18.752 seconds

Denormalized, no indexes



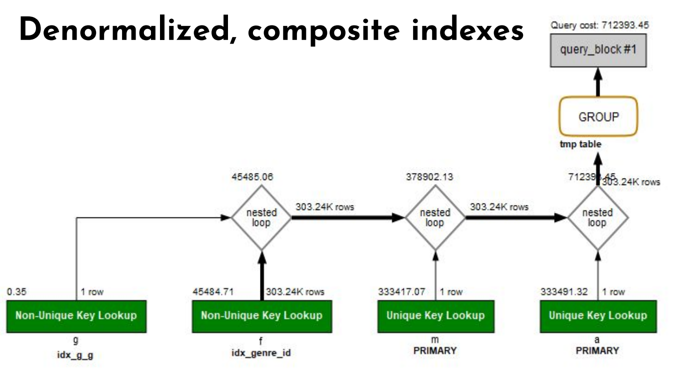
Execution Time: 11.860 seconds

Denormalized, with indexes



Execution Time: 2.547 seconds

Denormalized, composite indexes



Execution Time: 1.391 seconds

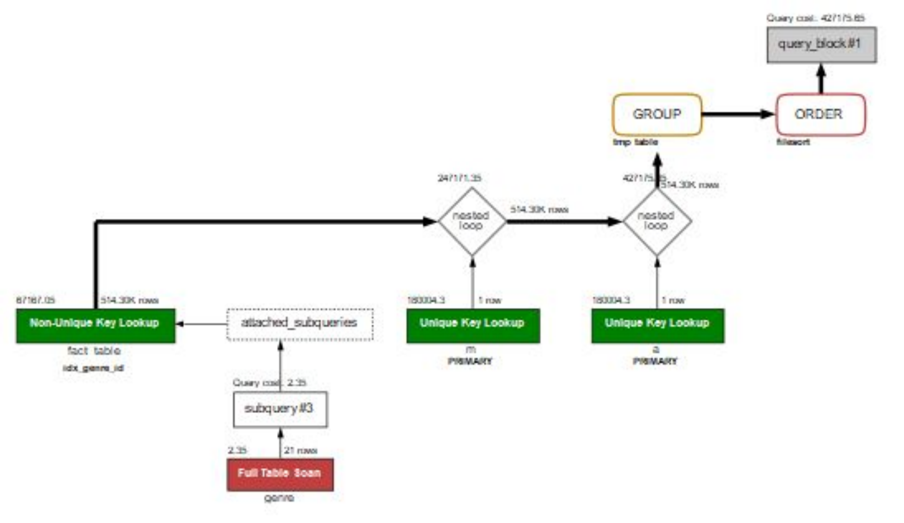
QUERY PROCESSING

Q3. Who are the top 15 actors with the most number of movie appearances for the action genre?

Optimized Query

```
SELECT CONCAT(a.last_name, ", ",
    a.first_name)
    AS name, COUNT(m.movie_id) AS
    Num_roles
FROM (
    SELECT *
    FROM fact_table
    WHERE genre_id = (
        SELECT genre_id
        FROM genres
        WHERE genre = "Action"))
    AS f
JOIN actors a ON f.actor_id = a.actor_id
JOIN movies m ON f.movie_id = m.movie_id
GROUP BY a.last_name, a.first_name
ORDER BY Num_roles DESC
LIMIT 15;
```

Query Execution Plan



Execution Time: 0.728 seconds

OPTIMIZATION AND RESULTS

QUERY PROCESSING

Q4. For each country, what are the top 10 movie genres?

```
SELECT *
FROM (
   SELECT c.country, g.genre,
       COUNT(m.movie_id)
       AS Num_movies,
       RANK() OVER
       (PARTITION BY c.country
       ORDER BY COUNT(m.movie_id) DESC)
       AS Num_movies_rank
   FROM movies m
   JOIN countries c ON c.movie_id = m.movie_id
   JOIN genres g ON g.movie_id = m.movie_id
   WHERE c.country  != ""
   GROUP BY c.country, g.genre
) AS Movies_rank
WHERE Num_movies_rank <= 10;
```

NORMALIZED SCHEMA

```
SELECT *
FROM (
   SELECT c.country, g.genre, COUNT(m.movie_id)
       AS Num_movies,
       RANK() OVER
       (PARTITION BY c.country
       ORDER BY COUNT(m.movie_id) DESC)
       AS Num_movies_rank
   FROM fact_table f
   JOIN countries c ON c.country_id = f.country_id
   JOIN genres g ON g.genre_id = f.genre_id
   JOIN movies m ON m.movie_id = f.movie_id
   WHERE c.country  != ""
   GROUP BY c.country, g.genre
) AS Movies_rank
WHERE Num_movies_rank <= 10;
```

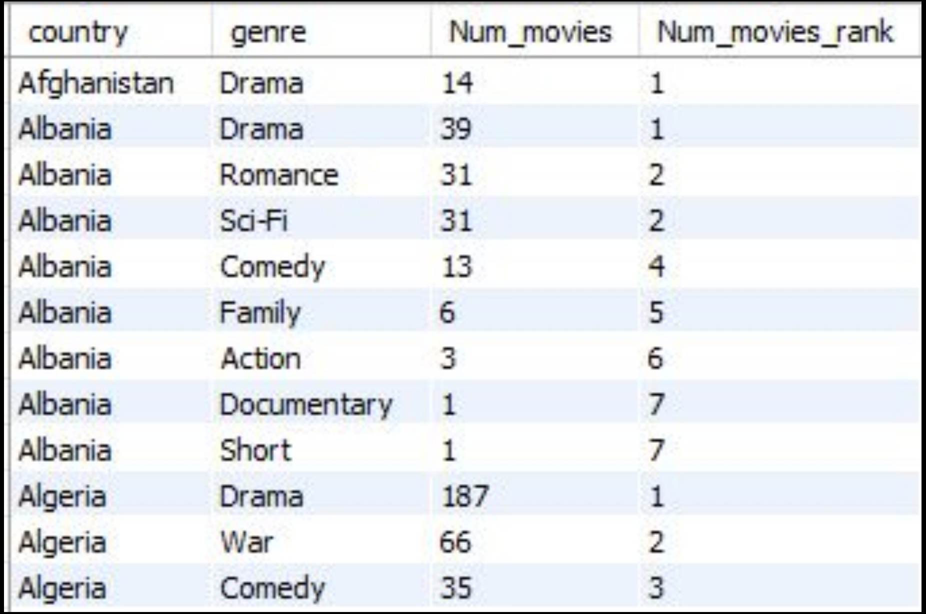
DENORMALIZED SCHEMA

OPTIMIZATION AND RESULTS

QUERY PROCESSING

Q4. For each country, what are the top 10 movie genres?

Output



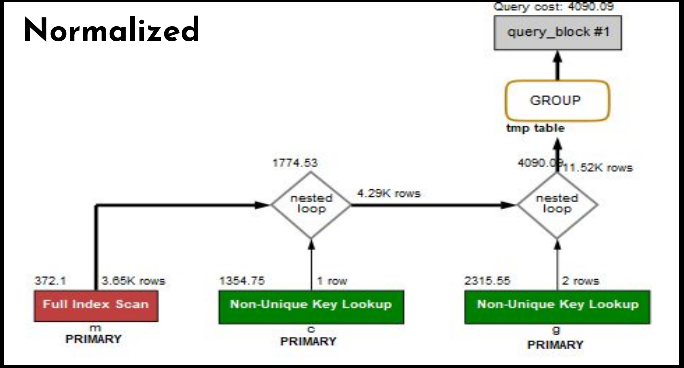
country	genre	Num_movies	Num_movies_rank
Afghanistan	Drama	14	1
Albania	Drama	39	1
Albania	Romance	31	2
Albania	Sci-Fi	31	2
Albania	Comedy	13	4
Albania	Family	6	5
Albania	Action	3	6
Albania	Documentary	1	7
Albania	Short	1	7
Algeria	Drama	187	1
Algeria	War	66	2
Algeria	Comedy	35	3

Rows: 1057
Columns: 4

Q4. For each country, what are the top 10 movie genres?

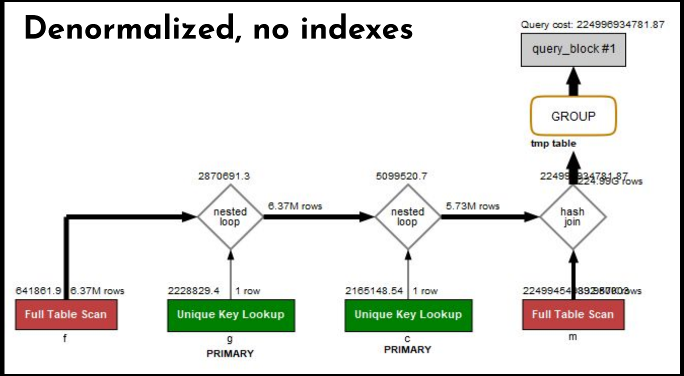
Query Execution Plans

Normalized



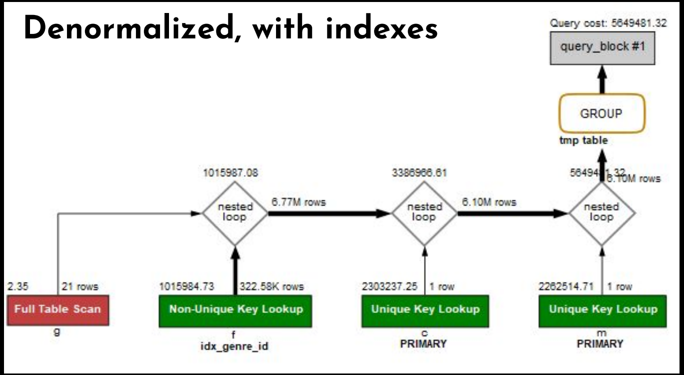
Execution Time: 25.682 seconds

Denormalized, no indexes



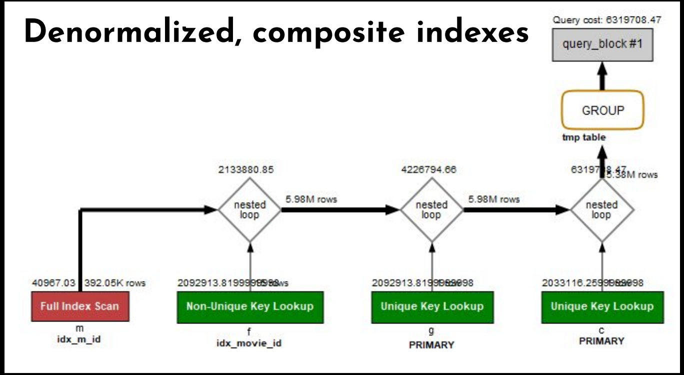
Execution Time: 21.129 seconds

Denormalized, with indexes



Execution Time: 14.884 seconds

Denormalized, composite indexes



Execution Time: 9.468 seconds

QUERY PROCESSING

Q5. How many horror movies are released in the USA for each year and decade?

```
SELECT CONCAT(FLOOR(year/10) * 10, 's')
   AS decade_released, m.year,
   COUNT(m.movie_id) AS
   num_movies
FROM fact_table f
JOIN genres g ON g.genre_id = f.genre_id
JOIN movies m ON m.movie_id = f.movie_id
JOIN countries c ON c.country_id =
   f.country_id
WHERE c.country = "USA"
   AND g.genre = "Horror"
GROUP BY decade_released, m.year
WITH ROLLUP;
```

NORMALIZED SCHEMA

```
SELECT CONCAT(FLOOR(year/10) * 10, 's')
   AS decade_released, m.year,
   COUNT(m.movie_id) AS
   num_movies
FROM fact_table f
JOIN genres g ON g.genre_id = f.genre_id
JOIN movies m ON m.movie_id = f.movie_id
JOIN countries c ON c.country_id =
   f.country_id
WHERE c.country = "USA"
   AND g.genre = "Horror"
GROUP BY decade_released, m.year
WITH ROLLUP;
```

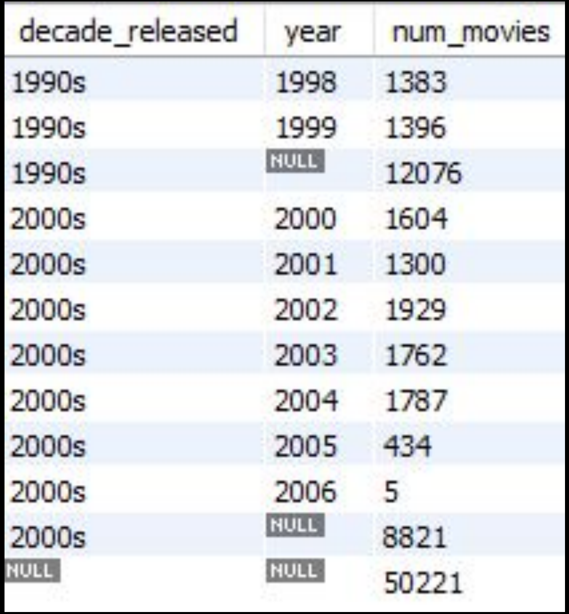
DENORMALIZED SCHEMA

OPTIMIZATION AND RESULTS

QUERY PROCESSING

Q5. How many horror movies are released in the USA for each year and decade?

Output

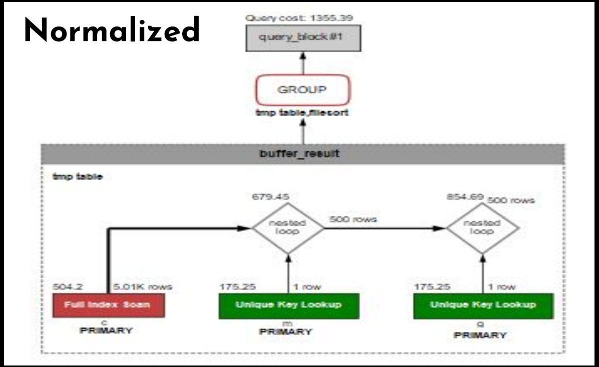


decade_released	year	num_movies
1990s	1998	1383
1990s	1999	1396
1990s	NULL	12076
2000s	2000	1604
2000s	2001	1300
2000s	2002	1929
2000s	2003	1762
2000s	2004	1787
2000s	2005	434
2000s	2006	5
2000s	NULL	8821
NULL	NULL	50221

Rows: 99
Columns: 3

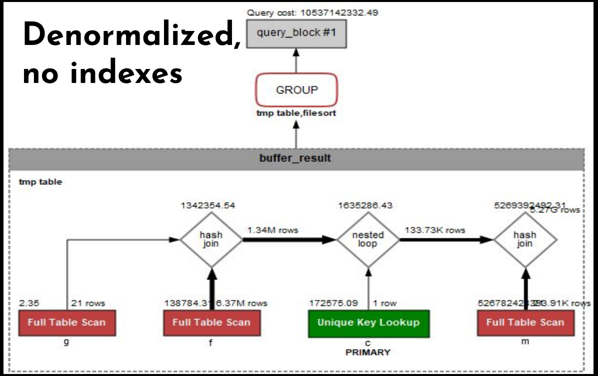
Q5. How many horror movies are released in the USA for each year and decade?

Query Execution Plans



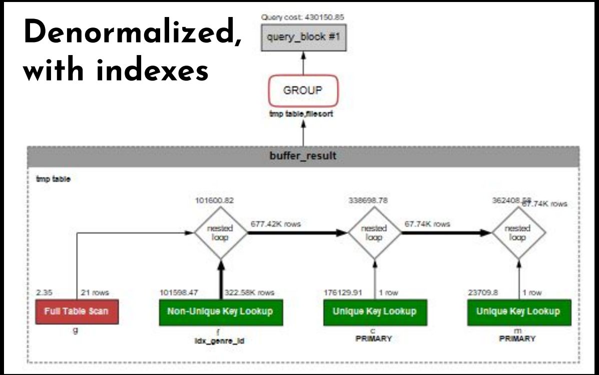
Normalized

Execution Time: 31.154 seconds



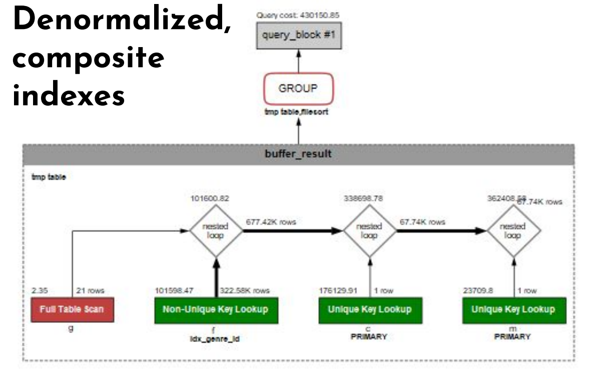
Denormalized, no indexes

Execution Time: 2.860 seconds



Denormalized, with indexes

Execution Time: 0.562 seconds



Denormalized, composite indexes

Execution Time: 0.328 seconds

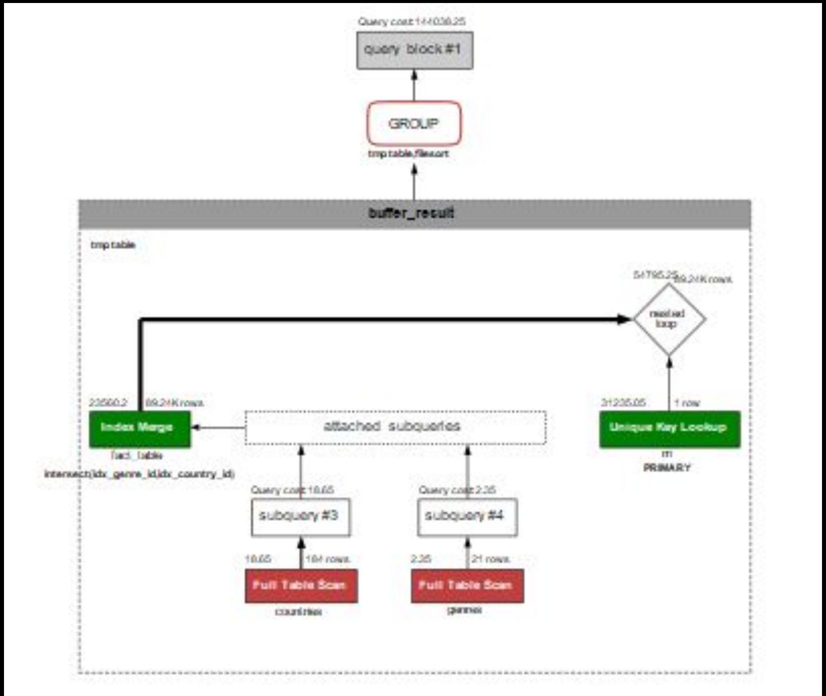
QUERY PROCESSING

Q5. How many horror movies are released in the USA for each year and decade?

Optimized Query

```
SELECT CONCAT(FLOOR(year/10) * 10, 's') AS
    decade_released, m.year, COUNT(m.movie_id)
    AS num_movies
FROM (
    SELECT *
    FROM fact_table
    WHERE country_id = (
        SELECT country_id
        FROM countries
        WHERE country = "USA"
    )
    AND genre_id = (
                SELECT genre_id
        FROM genres
        WHERE genre = "Horror"
    )) AS f
JOIN movies m ON m.movie_id = f.movie_id
GROUP BY decade_released, m.year
WITH ROLLUP;
```

Query Execution Plan



Execution Time: 0.152 seconds

OPTIMIZATION AND RESULTS

QUERY PROCESSING

Q6. What are the production companies that have released over 10 movies with a rating above the average rating and a number of votes above the average total number of votes?

```
SELECT production_company, COUNT(imdb_title_id) AS num_movies
FROM (
  SELECT i.production_company, i.imdb_title_id
  FROM imdb_movies i
  WHERE i.imdb_title_id IN (
    SELECT r.imdb_title_id
    FROM imdb_ratings r
    WHERE r.mean_vote > (
      SELECT AVG(r.mean_vote)
      FROM imdb_ratings r
    )
  )
  AND imdb_title_id IN (
    SELECT i.imdb_title_id
    FROM imdb_movies i
    WHERE i.votes > (
         SELECT AVG(i.votes)
       FROM imdb_movies i
    )
  )
) AS selected_movies
GROUP BY production_company
HAVING num_movies > 10
ORDER BY num_movies DESC;
```

NORMALIZED SCHEMA

```
SELECT production_company, COUNT(movie_id) AS  num_movies
FROM (
  SELECT m.movie_id, m.production_company,
     total_num_votes AS total
  FROM movies m
  JOIN fact_table ft ON ft.movie_id = m.movie_id
  JOIN ratings ra ON ra.rating_id = ft.rating_id
  WHERE m.production_company IS NOT NULL
  AND m.rank > (
    SELECT AVG(m.rank)
    FROM movies m
  )
  GROUP BY m.movie_id
  HAVING total > (
    SELECT AVG(total_num_votes)
    FROM (
      SELECT  name, total_num_votes
      FROM fact_table f
      JOIN movies m ON m.movie_id = f.movie_id
      JOIN ratings r ON r.rating_id = f.rating_id
      JOIN genres g  ON g.genre_id = f.genre_id
   ) as total_votes)
) AS selected_movies
GROUP BY production_company
HAVING num_movies > 10
ORDER BY num_movies DESC;
```

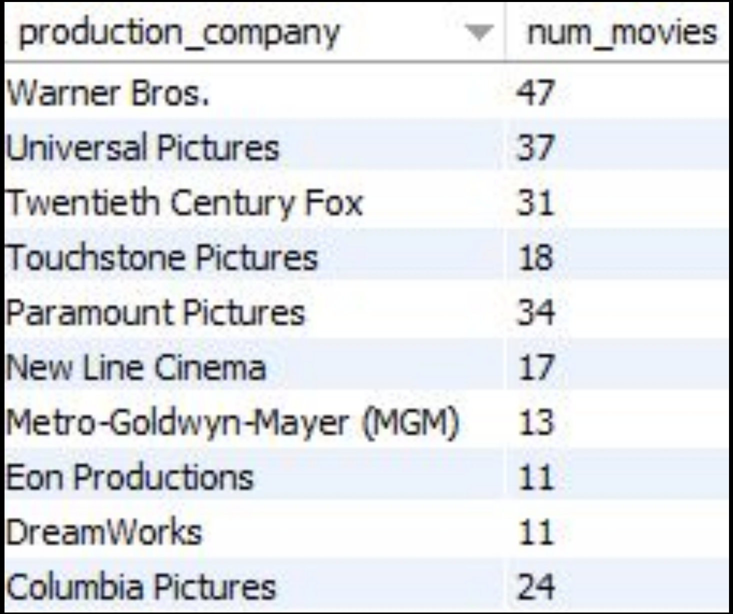
DENORMALIZED SCHEMA

OPTIMIZATION AND RESULTS

QUERY PROCESSING

Q6. What are the production companies that have released over 10 movies with a rating above the average rating and a number of votes above the average total number of votes?

Output



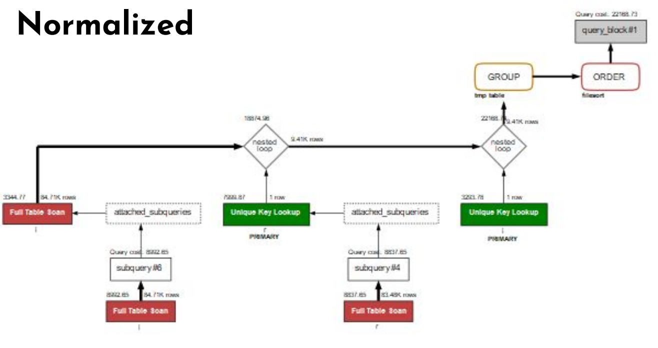
production_company	num_movies
Warner Bros.	47
Universal Pictures	37
Twentieth Century Fox	31
Touchstone Pictures	18
Paramount Pictures	34
New Line Cinema	17
Metro-Goldwyn-Mayer (MGM)	13
Eon Productions	11
DreamWorks	11
Columbia Pictures	24

Rows: 10
Columns: 2

OPTIMIZATION AND RESULTS

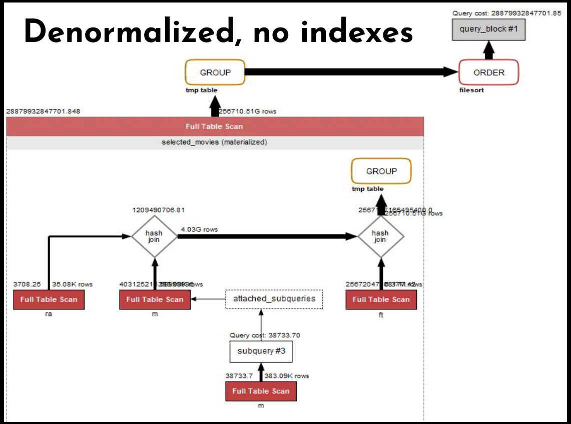
Q6. What are the production companies that have released over 10 movies with a rating above the average rating and a number of votes above the average total number of votes?

Normalized



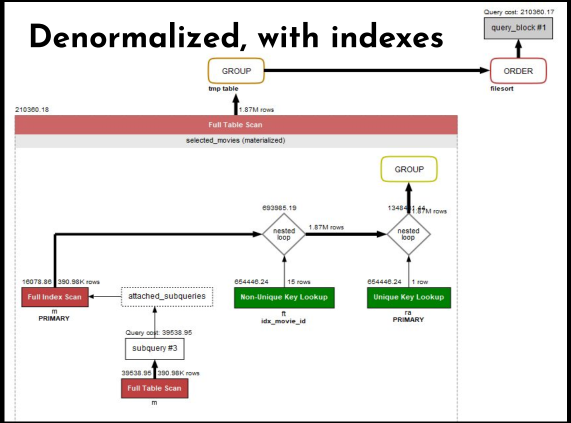
Execution Time: 10.311 seconds

Denormalized, no indexes



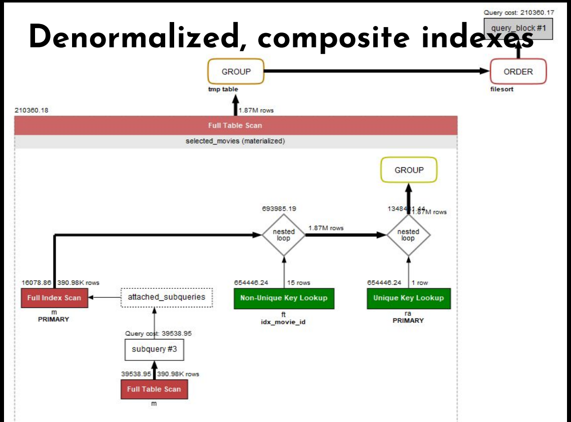
Execution Time: 12.468 seconds

Denormalized, with indexes



Execution Time: 10.531 seconds

Denormalized, composite indexes



Execution Time: 8.762 seconds

QUERY PROCESSING

Q7. For each of the top 6 directors with the highest gross earnings, what are their 15 highest-rated movies?

```
SELECT name, title, `rank`
FROM (
   SELECT m.title, topdir.name, topdir.gross,
      r.rank, ROW_NUMBER() OVER(PARTITION BY
      topdir.name ORDER BY `rank` DESC) AS
      director_movie_rank
   FROM (
      SELECT d.directorid, d.name, d.gross
      FROM directors d
      ORDER BY d.gross DESC
      LIMIT 6
   ) as topdir
   JOIN movies m
   JOIN ratings r ON r.movieid = m.movieid
   JOIN movies2directors md ON md.movieid =
      m.movieid AND md.directorid =
      topdir.directorid
   ORDER BY topdir.gross DESC, r.rank DESC
) AS topdirmovies
WHERE director_movie_rank <= 15;
```

NORMALIZED SCHEMA

```
SELECT full_name, gross, name, `rank`
FROM (
   SELECT full_name, m.name, gross, m.rank,
      ROW_NUMBER() OVER
      (PARTITION BY td.director_id ORDER BY
         `rank` DESC) AS Director_movie_rank
   FROM (
      SELECT d.director_id,
         CONCAT(CONCAT(d.last_name, ", "),
         d.first_name) AS full_name, d.gross
      FROM directors d
      ORDER BY d.gross DESC
      LIMIT 6
   ) as td
   JOIN fact_table f ON td.director_id =  f.director_id
   JOIN movies m ON m.movie_id = f.movie_id
   WHERE `rank` IS NOT NULL
   GROUP BY m.movie_id
   ORDER BY gross DESC, m.rank DESC
) AS top_director_movies
WHERE Director_movie_rank <= 15;
```

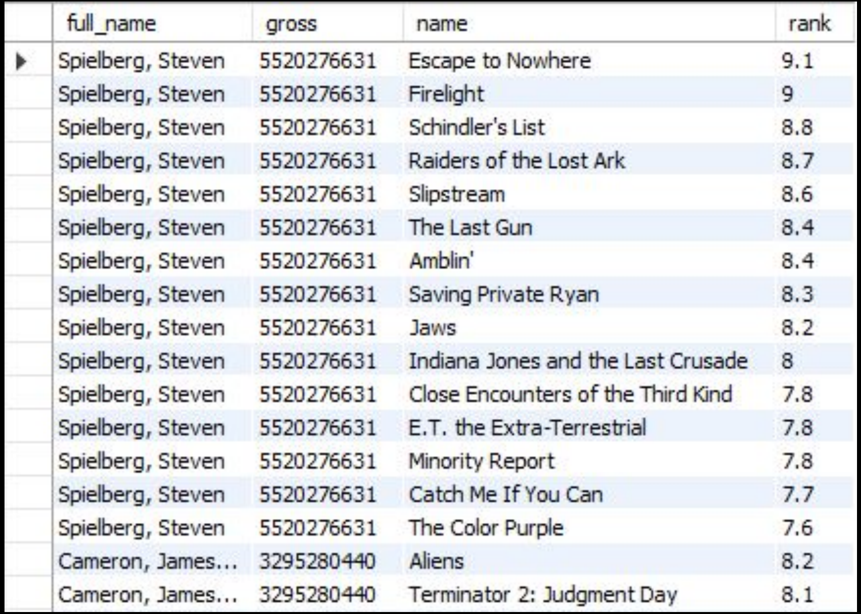
DENORMALIZED SCHEMA

OPTIMIZATION AND RESULTS

QUERY PROCESSING

Q7. For each of the top 6 directors with the highest gross earnings, what are their 15 highest-rated movies?

Output



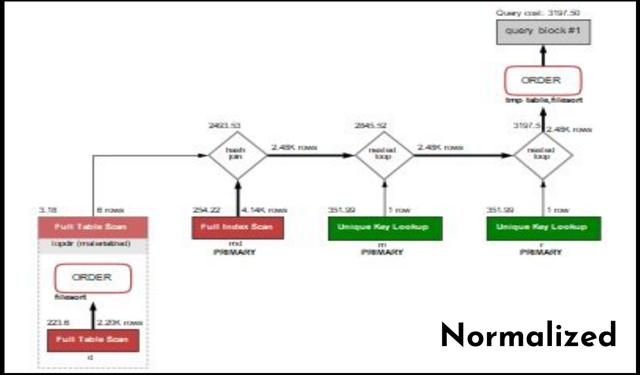
full_name	gross	name	rank
Spielberg, Steven	5520276631	Escape to Nowhere	9.1
Spielberg, Steven	5520276631	Firelight	9
Spielberg, Steven	5520276631	Schindler's List	8.8
Spielberg, Steven	5520276631	Raiders of the Lost Ark	8.7
Spielberg, Steven	5520276631	Slipstream	8.6
Spielberg, Steven	5520276631	The Last Gun	8.4
Spielberg, Steven	5520276631	Amblin'	8.4
Spielberg, Steven	5520276631	Saving Private Ryan	8.3
Spielberg, Steven	5520276631	Jaws	8.2
Spielberg, Steven	5520276631	Indiana Jones and the Last Crusade	8
Spielberg, Steven	5520276631	Close Encounters of the Third Kind	7.8
Spielberg, Steven	5520276631	E.T. the Extra-Terrestrial	7.8
Spielberg, Steven	5520276631	Minority Report	7.8
Spielberg, Steven	5520276631	Catch Me If You Can	7.7
Spielberg, Steven	5520276631	The Color Purple	7.6
Cameron, James...	3295280440	Aliens	8.2
Cameron, James...	3295280440	Terminator 2: Judgment Day	8.1

Rows: 76
Columns: 4

OPTIMIZATION AND RESULTS

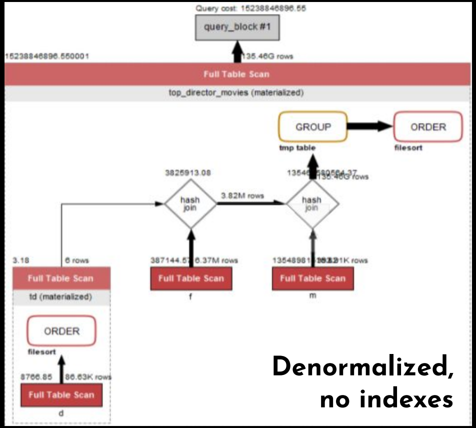
Q7. For each of the top 6 directors with the highest gross earnings, what are their 15 highest-rated movies?

Query Execution Plans



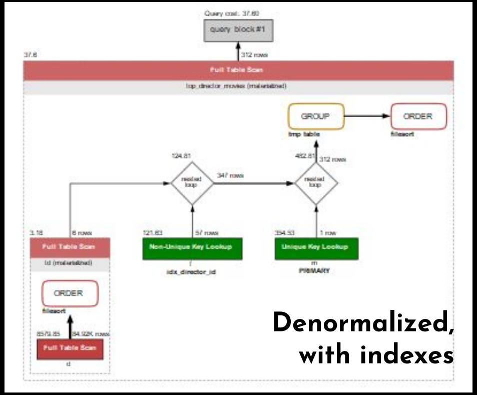
Normalized

Execution Time: 2.906 seconds



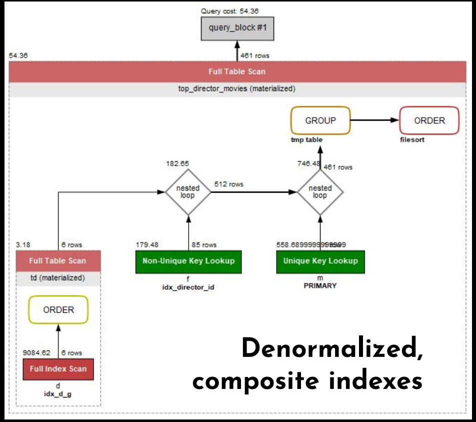
Denormalized, no indexes

Execution Time: 0.157 seconds



Denormalized, with indexes

Execution Time: 0.062 seconds



Denormalized, composite indexes

Execution Time: 0.032 seconds

OPTIMIZATION AND RESULTS

naïve nested-loop joins vs indexed nested- loop joins

B⁺-tree Lookup

$$\boxed{O(\log n)}$$

Time complexity

$$\boxed{O(n)}$$

Space complexity

Join algorithm costs

$$\boxed{|r| \times b_s + b_r}$$

naïve nested-loop join
block transfers

$$\boxed{|r| + b_r}$$

naïve nested-loop join
disk seeks

$$\boxed{b_r \times (T_{transfer} + T_{seek}) + |r| \times c}$$

indexed nested-loop join

DATABASE SCHEMA

Metadata Validation

✓ metadata of the local IMDb IJS database was compared to the remote copy

```
SELECT TABLE_NAME, COLUMN_NAME,
   ORDINAL_POSITION, IS_NULLABLE, COLUMN_TYPE,
   COLLATION_NAME
FROM INFORMATION_SCHEMA.COLUMNS
WHERE table_schema = 'imdb_ijs';
```

TESTING

Content Validation

- ✓ **CHECKSUM used for IMDb IJS**

```
CHECKSUM TABLE imdb_ijs.actors;
```

- ✓ **Boundary value analysis for IMDb Supplementary**

Maximum and minimum per column

- ✓ **Compare cardinality of table against CSV file**

Preprocessing Validation

✔ Movie Title Wrangling Validation (Comma)

```
SELECT COUNT(*) FROM imdb_star.movies
WHERE `name` LIKE "%,";
```

✔ Movie Title Wrangling Validation (Article)

```
SELECT COUNT(*) FROM imdb_star.movies
WHERE `name` LIKE "%, The";
```

✔ Genre Wrangling Validation

```
SELECT COUNT(*) FROM imdb_star.genres
WHERE genre LIKE "%,%";
```

The results of these SQL statements should be 0 since the data have already been preprocessed

TESTING ▲

Preprocessing Validation

✓ Production Company Wrangling Validation (Brackets)

```
SELECT COUNT(*) FROM imdb_star.movies
WHERE production_company LIKE "%[%]";
```

✓ Production Company Wrangling Validation (Parentheses)

```
SELECT COUNT(*) FROM imdb_star.movies
WHERE production_company LIKE "%- (%)";
```

The results of these SQL statements should be 0 since the data have already been preprocessed

TESTING

Performance Testing Results

Query	Normalized	Denormalized			
		No Indexes	With Indexes	With Composite Indexes	Revised Query
1	23.400	**9.344**	15.453	16.516	N/A
2	12.500	**9.359**	12.656	13.375	N/A
3	18.752	11.860	2.547	1.391	**0.728**
4	25.682	21.129	14.884	**9.468**	N/A
5	31.154	2.860	0.562	0.328	**0.152**
6	10.311	12.468	10.531	**8.762**	N/A
7	2.906	0.157	0.062	**0.032**	N/A

CONCLUSION AND INSIGHTS

CONCLUSION AND INSIGHTS

Deeper appreciation for the forms of data and issues arising from synthesizing data from different sources

Learning to integrate externals tools into the pipeline

Optimization strategies in query processing perform differently on a case-to-case basis

CONCLUSION AND INSIGHTS

Moviegoers can use the results to examine the history of genres and predict film quality

Producers and directors can find prolific actors for a genre or examine the past production company projects

The results allow for insights and meaningful interpretations of the film industry

Optimizing the Dashboard

- Using Tableau's relations instead of joins
- Using Tableau's sets

TABLEAU