An OLAP Dashboard to Generate Reports for IMDb Using Optimized Queries

Krischelle Lourdes Cadao[1], Annika Ayesha A. Capada [2], Lauren Antoinette M. Garcia[3] and Shanna Raven H. Tan[4]
Software Technology Department, College of Computer Studies, De La Salle University, Manila, Philippines
[1]krischelle_cadao@dlsu.edu.ph, [2]annika_capada@dlsu.edu.ph, [3]lauren_garcia@dlsu.edu.ph, [4]shanna_tan@dlsu.edu.ph

ABSTRACT
Data visualizations and analysis of an IMDb dataset were produced in order to generate knowledge that can guide moviegoers and movie creators in making informed decisions. The IMDb dataset is a relational database containing data about movies, genres, directors, and actors. As the project entails working with large datasets to generate analytical reports, the aim of this project is to identify ways to optimize the queries in order to generate reports using visualization tools. Using ETL scripts, the IMDb IJS dataset was copied and stored into the local machine. Afterwards, a denormalized database using the star schema served as the data warehouse for OLAP queries. An interactive dashboard OLAP application was developed to display the visualizations of the four queries generated on the data which include roll-up, drill-down, slice, and dice operations. Experiments were then conducted involving normalized databases, denormalized databases (star schema data warehouse), and indexes in order to optimize the queries particularly in terms of executing speed. To characterize the behavior of the queries and investigate the limitations and efficiency of each experiment, these were also examined and compared with each other. Based on the experiments, the results show that a star schema can speed up execution time and is designed for querying large-scale datasets. Furthermore, indexes can also enhance the performance of the query depending on the report–hence, this project illustrates the advantages of commonly used optimization techniques on real-world data, and presents insights in both the movie industry and database design and administration.

Keywords
Data Warehouse, ETL, OLAP, Query Processing, Query Optimization.

1. Introduction
For the longest time, IMDb has been one of the most convenient ways to get information on a movie, whether it be a quick search for ratings, checking the casting for certain roles, or watching trailers for upcoming movies. As such, IMDb contains a suitable dataset for creating an OLAP application.

The dataset used was a relational database consisting of seven tables, the first four containing information on actors, directors, movies, and roles, and the latter three tables acting as links between movies, directors, and their respective genres. An accurate visualization can be seen in Figure 1. Apache NiFi was used for ETL pipeline in order to transform the data into the desired star schema. For the OLAP application, Tableau was utilized in creating visualizations for the needed data. Users who are interested in seeing movie statistics can utilize the application to view yearly trends.

2. Data Warehouse
This section discusses the IMDb dataset used for the OLAP application, an explanation of the dimensional model of the data warehouse, and the issues encountered and how these were addressed.

2.1 IMDb Dataset
The IMDb dataset used for the project, referred to as IMDb IJS, is collected by Janez Kranjc and the Jožef Stefan Institute (IMDb Dataset, 2014) [7]. It is a relational database that includes details on movies and their corresponding genres, actors (with their roles), and directors. The MariaDB server, an open-source clone of MySQL, serves as its host.

Figure 1 displays the entity-relationship diagram (ERD) for this dataset. Description of the tables are listed in Table 1.

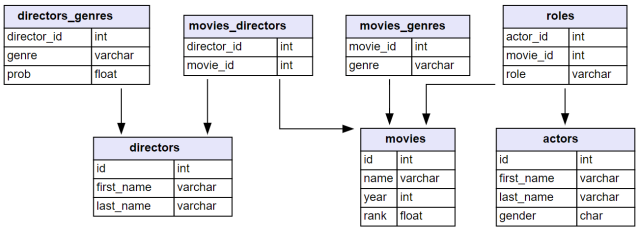

Figure 1. ERD of IMDb IJS [1]

Table 1. Tables of IMDb IJS

Name	Description
`movies`	Movies
`actors`	Actors
`directors`	Directors
`roles`	Mapping between the movies and actors
`movies_directors`	Mapping between movies and directors
`directors_genres`	Mapping between directors and genres
`movies_genres`	Mapping between movies and genres

2.2 Dimensional Model

The dimensional model for the data warehouse corresponds to a star schema, composed of one (1) fact table and three (3) dimension tables, namely "director", "role", and "actors". The fact table of the model is composed of the "movie_id", "director_id", "actor_id", and "genre". With all facts as the primary keys. The settlement on which schema is to be used is justified by its significant use cases in the OLAP application. One major advantage of the star schema is that it is denormalized (Praise for A Developer's Guide to Data Modeling for SQL Server, n.d.) [10] as it is said that it can accommodate a large number of read operations compared to insertions and updates seen under an OLAP environment. Comparing it towards the snowflake schema, data normalization under star schemas operates around multiple redundant tables, having to slow down join functions.

Christopher (Adamson, 2012) [1] suggests that a star schema design is optimized for queries with great volumes of data. Additionally, address a wide range of questions useful to the author's OLAP use cases under Section 4.

The star schema is shown under Figure 2 and furthermore explained in Section 2.2.1

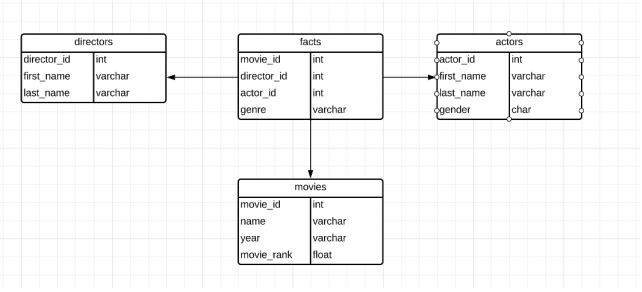

Figure 2. ERD of the Data Warehouse (Star Schema)

2.2.1 Fact and Dimension Tables

The fact table of the model is composed of the "movie_id", "director_id", "actor_id", and "genre"., connected to the three (3) dimensional tables which are "director", "movie", and "actors" tables. The level of detail represented by the fact table is significant with regards to the schema design process. As told by Christopher (Adamson, 2012) [1], it guarantees that all facts will be recorded at the same level of detail and is declared dimensionally. The grain of the fact table row is set at the lowest possible level of detail to capture data at an atomic level.

The star schema contains dimension tables for groups of attributes describing the director, movie, and actor. To maximize the load and success of the data warehouse, it is therefore important to ensure denormalization by collapsing many-to-many relationships based on volatility and low cardinality. The hierarchical relationship of the schema is as follows from highest to lowest priority in this order: genre, year, movie, director, actor.

The four tables under the data warehouse are described below in Table 2.

Table 2. Tables of the Data Warehouse

Name	Description
Facts	The fact table containing all relations between the dimension tables
- movie_id	ID of a movie under the movies table
- director_id	ID of director under the director table
- actor_id	ID of an actor under the actors table
- genre	Name of the genre
director	The dimension table that contains film directors under the database
- director_id	ID of a director
- first_name	First name of a director
- last_name	Last name of a director
movie	Dimension table that contains all movies under the database
- movie_id	ID of a movie
- year	Year the movie was released
- name	Name of a movie
- rank	Total score of a movie among all users
actors	Dimension table that contains all actors under the database
- actor_id	ID of an actor
- first_name	First name of an actor
- last_name	Last name of an actor
- gender	Gender of an actor

2.3 Issues Encountered and Considerations

The main issue that affected the database design was the size of the dataset. The largest table in the dataset is returned by the actors table, returning 817,718 rows; hence, the OLAP operations were first developed, evaluated and finalized before the schema was created. Given the time and resources, the *(i)* selection of schema, and *(ii)* selection of tables and attributes were taken into consideration.

Initially, the snowflake schema was used in designing the database because it uses less space and reduces data redundancy. However, after designing a draft schema, it was discovered that it would not produce the desired reports. Therefore, designing a star schema would be the most appropriate in accommodating high volumes of data and taking less time for the execution of queries (Smallcombe, 2023) [13].

For the selection of tables and columns, it was primarily based on their relevance to the OLAP application and the attributes

applicable to the reports to be generated. All of the tables in the IMDb IJS were included in the schema except for the `directors_genres` since the data in this table such as the probability of a director directing a movie for a specific genre will not be used and would already be analytical in nature. Moreover, attributes not directly used in the queries were also not included.

3. ETL Script

This section discusses the process for extracting, transforming, and loading (ETL) the data into the data warehouse. Apache NiFi was utilized to create the ETL pipeline.

3.1 Creating Local Copies

Due to slow speeds in accessing the source database, local copies first needed to be made from the source database. Local copies were also made as writing is not allowed on the original database.

To create local copies, the Data Export wizard from the Navigation area of MySQL Workbench was used to create an SQL file containing all tables and records from the source database. After, the local instance connection was opened and the generated SQL script was run to load the data.

3.2 Transferring to Data Warehouse

This section discusses the ETL script for transferring the data source into the data warehouse following the star schema. Before the ETL was executed, an empty schema was already created in MySQL following the dimensional model described in Section 2. The ETL pipeline is shown below in Figure 3.

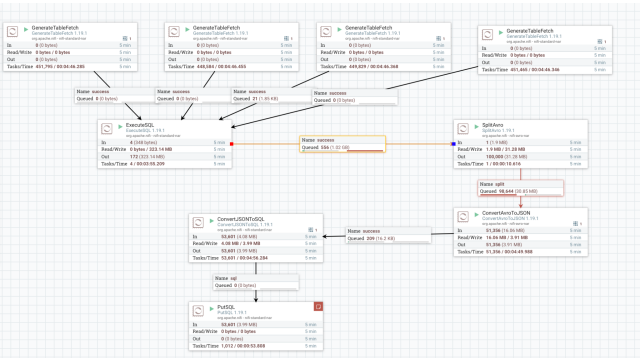

Figure 3. ETL Pipeline for Transferring to Data Warehouse

3.2.1 Extract

To create the fact table, it was important to know which tables from the original database would be joined. Knowing the number of tables to be joined determines the number of GenerateTableFetch processors to be used in the pipeline. For the fact table, four of such processors were used. Each processor would then fetch records from the indicated table it is configured to.

The four GenerateTableFetch processors were then all connected to one ExecuteSQL processor. The ExecuteSQL processor was configured to normalize the table/column names as a fact table was being produced. Listing #1 shows the SQL query placed inside the ExecuteSQL processor that is part of the ETL pipeline for the facts table. INNER JOIN operations were used so that the data extracted had matching values in all mentioned tables. In addition, only the movies with rankings were extracted so that the overall process of ETL would be faster and the data would be more relevant.

Listing 1. Extraction Query for `facts` Table

```
SELECT m.id as movie_id,d.director_id,
r.actor_id, g.genre
FROM movies m INNER JOIN movies_directors d
ON m.id = d.movie_id
INNER JOIN roles r ON m.id = r.movie_id
INNER JOIN movies_genres g ON m.id =
g.movie_id
WHERE m.rank IS NOT NULL;
```

For the dimensions of the star schema, join operations were no longer utilized. Thus, only one GenerateTableFetch processor was used when extracting data for the dimensions. The processors were also configured to a false value in normalizing the table/column names as dimension tables were being handled.

Listing 2-5 are simple SQL select queries for the directors, actors, and movies dimensions in the star schema.

Listing 2. Extraction Query for `directors` Table

```
SELECT id, first_name, last_name
FROM directors;
```

Listing 4. Extraction Query for `actors` Table

```
SELECT id, first_name, last_name, gender
FROM actors;
```

Listing 5. Extraction Query for `movies` Table

```
SELECT id, name, year, rank
FROM movies;
```

3.2.2 Transform

After executing the necessary SQL select queries, the SplitAvro processor was used to split the table into multiple and smaller binary files with a configured Output Size of one. Moreover, the table cache was increased to 1000 to improve performance. Then, the ConvertAvroToJSON processor was used to convert the AVRO files into JSON format. Lastly, the ConvertJSONToSQL processor was used to establish a JDBC connection with the data warehouse and convert the received JSON flowfiles into an INSERT SQL statement.

3.2.3 Load

Finally, the last processor used in the ETL pipeline was the PutSQL processor to load the data into the data warehouse. In this processor, a JDBC connection pool to the data warehouse was enabled and an INSERT SQL statement was used to load the data.

3.3 Issues Encountered

During the ETL process, some issues were encountered due to the large amount of data being handled. These issues were mostly experienced when creating the fact table for the data warehouse as this had the most records of data.

3.3.1 Error on Heap Space

One of the main issues encountered was the error on heap space memory when generating the fact table. This error occurred when the initial pipeline did not include any GenerateFetchTable processors. This is because the pipeline was handling large volumes of data which caused the computer system to quickly run out of memory. Additionally, the use of multiple JOIN operations contributes to the slow retrieval of data.

Initially, multiple SplitText processors were used to resolve the error on heap space memory by splitting the data into batches of 10000 and 1000 to handle the data in portions rather than handling all at once. Though the error on heap space memory was resolved, the speed of the pipeline was very slow. It was inferred that the slowness was still due to the multiple JOIN operations performed in the ExecuteSQL processor.

GenerateFetchTables resolved this issue because it allows the pipeline to first extract data from the necessary tables for the fact table. The processor allows configurations that specify which columns of a table are to be fetched and the partition size can be modified to partition the data in batches. In addition, incremental fetching was achieved by setting the Maximum-Value Columns to the primary key of each table. Configuring this setting is used to imply that there's a hierarchical structure to the columns and the processor can retrieve only rows that have been updated since the last retrieval (Apache Software Foundation, n.d) [2].

3.3.2 Bottlenecks due to Duplicates

The ETL process was performed multiple times in attempts to fully create the fact table. However, a frequent problem occurred whenever the pipeline was close to completing the ETL process. When about 2000 to 3000 records of data were left to load onto the data warehouse, bottlenecks in the queues would occur due to duplicate FlowFiles. The pipeline would take over 12 hours to search through all the loaded data to determine whether the record being handled can be inserted into the fact table.

In this case, it was never discovered as to how long it would actually take for NiFi to break out of those bottlenecks. A solution was not found to overcome this issue. Running the ExecuteSQL processor once ([NIFI-3422] Run Once Scheduling - ASF JIRA, 2017) [9] was thought to be a solution as this option is said to be useful for debugging flows or repeated FlowFiles by Nazario (2021). However, no improvements were noticed.

Thus, the fact table in the generated data warehouse is not complete and will cause slight data ambiguity in the reports. To be more specific, 2346210 out of 2348417 were successfully loaded into the data warehouse leaving 2207 files not included. With this, 99.91% of the data has been successfully loaded into the data warehouse.

As a suggestion, different processors like DetectDuplicate may be studied so that they may be properly implemented and resolve the problem.

3.4 Data Warehouse Volume

As mentioned in the previous section, there is data ambiguity due to the unresolved issue of bottlenecks due to duplicate files during the ETL process. However, due to time constraints, the incomplete data in the fact table was settled and used for the reports. Listed in Table 3 are the number of rows of tables in the data warehouse.

Table 3. Tables of Data Warehouse

Name	Number of Rows
facts	2346210
actors	817718
directors	817718
movies	817718

4. OLAP Application

This section discusses the specific reports that were generated as part of the OLAP application, alongside the decisions made based on the analytical tasks highlighted in the project.

The application is a dashboard using a visualization software called Tableau which allows users to easily identify trends and offer reports on the IMDB dataset about the film industry. Specifically, it delivers information on the number of movies made per year, the number of female and male actors for each genre, and popular directors among others. The purpose of the application is to provide users, depending on their role, a better understanding of the movie industry and may influence their decision in going forward with the film industry. The application can be applied to different types of users such as moviegoers, directors, and actors. To support the decision-making processes, the application's visualizations are to be used. Screenshots of the interactive visualizations and their interpretations are provided in Section 5 of this report.

4.1 Generated Reports

This section presents the OLAP reports that were generated using the OLAP operations roll-up, drill-down, slice, and dice. Sample of these visualizations (created using the software Tableau) are seen in Figures 4 to 7.

4.1.1 For each genre, what is the total number of movies per year?

This query displays the number of movies made for each genre (e.g. action, adventure, comedy, and so on) in each year. The roll-up operation was used for this query in order to generate the aggregate values for the data's general categories (i.e. year).

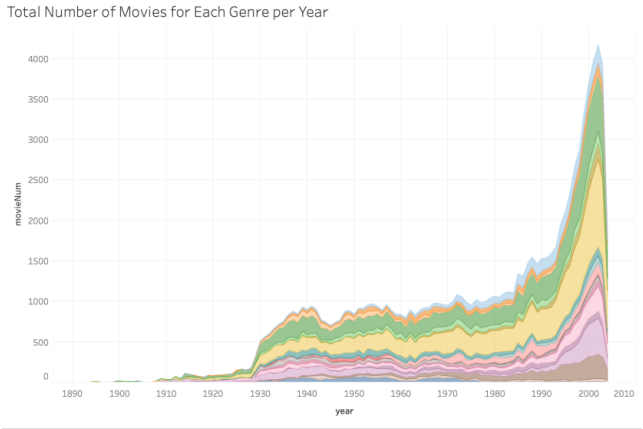

Figure 4. Tableau Dashboard for Query 4.1.1

4.1.2 For each year, what are the total number of male and female actors per genre?

The query shows the total number of male and female actors performed in all movie genres per decade. Additionally, the query displays detailed information of the data on the count of male or female per year in a given genre and an insightful interpretation of which gender of the actors can be seen under a specific genre.

With the use of the drill-down operation, it is possible to see the breakdown of results after the decade level. While for the drill-up operation–view from the bottom up, it aggregates the number of the actor's gender in a year and per decade.

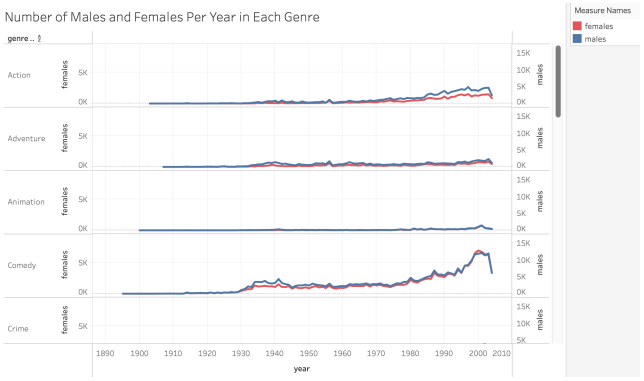

Figure 5. Tableau Dashboard for Query 4.1.2

4.1.3 Who are the top directors that made the most movies for the comedy genre?

This query shows the names of the top directors with the highest number of movies made for the comedy genre in decreasing order. This query makes use of the slice operation where the genre is limited to comedy only.

Originally, the initial report would limit the directors to the top 10. However, because of the inconsistency in results between the SQL queries and the visualization shown in Tableau when changing genres, it was decided that filtering the top 10 in Tableau would not be implemented. This is so that users of the OLAP application can interact with it more and look at the directors for different genres based on the amount of movies made.

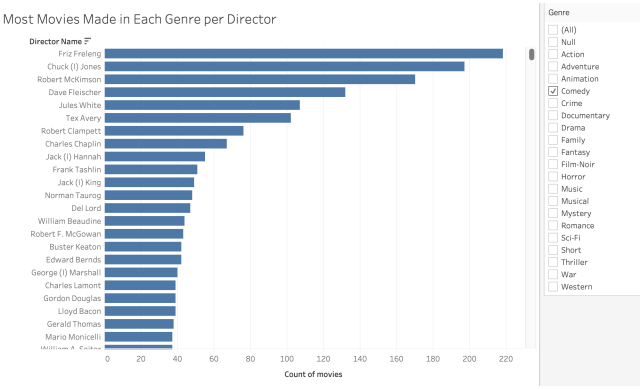

Figure 6. Tableau Dashboard for Query 4.1.3

4.1.4 How many action movies were made from 1990 to 2000?

This query shows the number of action movies made in each year from the year 1990 to 2000. The dice operation was utilized for this query in order to isolate the genre to action and movies made in from 1990 to 2000.

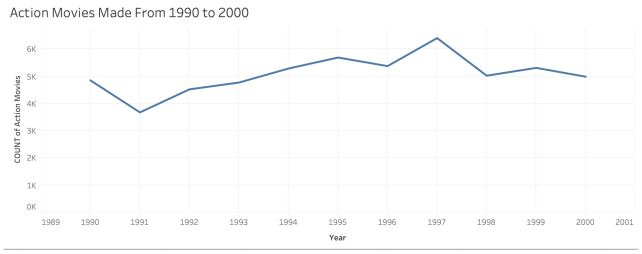

Figure 7. Tableau Dashboard for Query 4.1.4

5. Query Processing Optimization

Several optimizations were carried out, that were mentioned in the previous sections, to enhance the performance of the data warehouse operations. The size of cache tables, as well other system variables relevant to the MySQL server were increased during the ETL process to quicken the execution of join operations (Section 3.2).

The following experiments were used in evaluating the optimization techniques and the queries performance:

- Executing the query on the normalized database, which was produced by creating a local copy of the dataset (Section 3.1). The indexes used in the original dataset were also used.
- Running the query on the denormalized dataset which is the star schema based data warehouse without indexes.
- Running the query on the denormalized dataset, the star schema based data warehouse with indexes.

Each experiment was run for 5 times wherein the average was taken and placed as the result for the execution time. It is important to note that there is slight ambiguity in the results of the optimized queries due to the incomplete loading of records from the source database to the data warehouse.

5.1 For each genre, what is the total number of movies per year?

By determining the number of movies made for each genre, it would be possible to distinguish the genre that was "trending" for a given year. Moreover, this may also reveal a genre's popularity and the preferences of audiences in specific years. In the case of directors, they may plan what upcoming movie genres to make next based on a genres popularity or an entirely different genre that would be suitable for users.

5.1.1 Output

The output resulted in 1887 rows and 3 columns. This can be seen below in Figure 8

year	genre	movieNum
NULL	NULL	55790
1892	NULL	1
1892	Documentary	1
1892	Short	1
1893	NULL	1
1893	Short	1
1894	NULL	3
1894	Documentary	2
1894	Short	3
1895	NULL	5
1895	Comedy	1
1895	Documentary	3
1895	Short	5
1896	NULL	4
1896	Documentary	1
1896	Horror	1
1896	Short	4

Figure 8. Output for Query 5.1

5.1.2 Optimization and Statistics

Generating the OLAP report for Query 5.1 using a normalized database (Listing 6) took an average of 11.537 seconds.

Listing 6. SQL Statement for Query 5.1 (Normalized)

```
SELECT m.year, mg.genre, COUNT(DISTINCT id)
AS movieNum
FROM movies m
JOIN movies_genres mg ON mg.movie_id=m.id
JOIN movies_directors md ON
md.movie_id=m.id
JOIN roles r ON r.movie_id=m.id
WHERE m.rank IS NOT NULL
GROUP BY m.year, mg.genre WITH ROLLUP
ORDER BY m.year, mg.genre;
```

Table #4. Average Duration for Query 5.1 (Normalized)

	Duration (seconds)
1	12.641
2	11.906
3	11.203
4	11.500
5	10.437
Average	11.537

Based on Listing #6, in order to determine the number of movies made per year, the COUNT() and GROUP BY function were used to count and group the movies based on their corresponding years and genres. The data was then sequenced according to their year and genre. The JOIN function was used to link the tables and because the mapping table has the lowest cardinality, the optimizer does a full table scan.

Afterwards the joined table is buffered into a temporary table before being grouped and sorted which is necessary in roll-up as it involves aggregating values (MySQL :: MySQL 5.6 Reference Manual :: 8.2.1.14 GROUP BY Optimization, 2023) [8].

On the other hand, generating the OLAP report for Query 5.1 using a denormalized database without indexes (Listing #7) took an average of 6.344 seconds, showing an improvement.

Listing #7. SQL Statement for Query 5.1 (Denormalized)

```
SELECT m.year, f.genre, COUNT(DISTINCT
m.movie_id) AS movieNum
FROM facts f
JOIN movies m ON m.movie_id=f.movie_id
WHERE m.rank IS NOT NULL
GROUP BY m.year, f.genre WITH ROLLUP
ORDER BY m.year, f.genre;
```

Table #5. Average Duration for Query 5.1 (Denormalized)

	Duration (seconds)
1	5.875
2	6.578
3	6.219
4	6.297
5	6.750
Average	6.344

Based on Table #5, using the denormalized table sped up the execution time because it reduces the number of JOIN's in the query, hence, boosts performance (Hampel, M. , 2021) [6]. Creating an index would not be suitable for this report because the query would still have to process all the rows.

5.2 For each year, what are the total number of male and female actors per genre?

Identifying the total number of male and female actors sorted out on the basis of the movie genre they took part in determines the popularity of a genre towards the audience. Through this data, filmmakers would indicate a trendline where movie genres have preferences of having either male or female characters or actors more often than in previous years. Further samples from (Blanchard et.al, 1986) [4], reported that female actors are preferred under horror movie genres that typically contain frightening and violent scenes.

5.2.1 Output

Based on the Figure #9, the query returns 1887 rows and 4 columns, each pertaining to the year of the genre and the count of male and female actors of each genre.

year	genre	male_count	female_count
2000	Action	3731	1256
2000	Adventure	1994	771
2000	Animation	1060	501
2000	Comedy	11832	6988
2000	Crime	3516	1334
2000	Documentary	1565	620
2000	Drama	16542	9075
2000	Family	1488	885
2000	Fantasy	1040	608
2000	Horror	1906	1060
2000	Music	686	287
2000	Musical	513	327
2000	Mystery	2382	1422
2000	Romance	3766	2607
2000	Sci-Fi	1152	490
2000	Short	3682	2210
2000	Thriller	5168	2373
2000	War	545	177
2000	Western	392	131

Figure 9. Output for Query 5.2

5.2.2 Optimization and Statistics

Generating an OLAP report using a normalized database (Listing #8) took 20.68 seconds.

Listing #8. SQL Statement for Query 5.1 (Normalized)

```
SELECT DISTINCT m.year, mg.genre, SUM(case
when gender='M' then 1 else 0 end) AS
male_count, SUM(case when gender='F' then 1
else 0 end) as female_count
FROM movies m
JOIN movies_genres mg ON m.id = mg.movie_id
JOIN roles r ON m.id = r.movie_id
JOIN actors a ON r.actor_id = a.id
JOIN movies_directors md ON m.id =
md.movie_id
WHERE m.rank IS NOT NULL
GROUP BY m.year, mg.genre WITH ROLLUP
ORDER BY m.year, mg.genre;
```

Table #6. Average Duration for Query 5.2 (Normalized)

	Duration (seconds)
1	20.516
2	22.531
3	20.922
4	19.797
5	19.625
Average	20.6782

Using the MySQL Application, the drill-down operation is not a built-in function. Hence, as seen under Listing #8, the ROLL UP function was used. Since the roll-up and 'roll-down' or drill down differs by the direction of the granularity. Increased granular level of detail or decreased granular level of detail that provides insights on the totality of the data, that is for the description of the two functions.

From the Listing #8, the SUM() function was used to provide the distinct total of each parameter or group under male and female. While the GROUP BY function groups the genders onto its year and specific genre with respect to the ROLL UP function and its intended aggregating values. The JOIN functions used were to connect each table based on the lowest cardinality from genres, roles, and to actors.

Generating an OLAP report using a denormalized database (Listing 9) took 13.26 seconds.

Listing 9. SQL Statement for Query 5.1 (Denormalized)

```
SELECT m.year, f.genre, SUM(case when
gender='M' then 1 else 0 end) AS males,
SUM(case when gender='F' then 1 else 0 end)
AS females
FROM facts f
JOIN movies m ON m.movie_id = f.movie_id
JOIN actors a ON a.actor_id = f.actor_id
GROUP BY  m.year, f.genre WITH ROLLUP
ORDER BY m.year, f.genre;
```

While for the denormalized query under Listing #9, evident differences can be seen such as the use of JOIN operations for movies and actors tables as it requires only the tables needed to perform the use case. This query results in a faster duration (As seen under Table #7) based on the reduction in the number of JOIN functions or joined tables. The normalized four join operations into only two results in a decrease of time complexity by two orders of magnitude.

Table #7. Average Duration for Query 5.2 (Denormalized)

	Duration (seconds)
1	17.454
2	16.842
3	11.341
4	10.293
5	10.389
Average	13.2638

5.3 Who are the top directors that made the most movies for the comedy genre?

Directors are the creative leaders of the film and make artistic choices that affect the special effects, filming locations, camera angles, and actor performances to shape the look of a movie (Chen, 2021) [5]. By seeing the top directors under the movie genre, comedy, moviegoers would be able to identify which comedy movies are worth watching based on the director and how they direct their films.

5.3.1 Output

The output resulted in 10 rows and 2 columns. This can be seen below in Figure #10

director_name	movies_made
Friz Freleng	218
Chuck (I) Jones	197
Robert McKimson	170
Dave Fleischer	133
Jules White	107
Tex Avery	102
Robert Clampett	76
Charles Chaplin	67
Jack (I) Hannah	55
Frank Tashlin	51

Figure 10. Output for Query 5.3

5.3.2 Optimization and Statistics

Generating the OLAP report for Query 5.3 using a normalized database (Listing 10) took an average of 9.066 seconds as seen in Table #8.

Listing 10. SQL Statement for Query 5.3 (Normalized)

```
SELECT director_name, movies_made
FROM (SELECT DISTINCT CONCAT(first_name,
      ' ', last_name) AS director_name,
      COUNT(DISTINCT m.id) AS movies_made,
```

```
      RANK() OVER (PARTITION BY g.genre
      ORDER BY COUNT(DISTINCT m.id) DESC)
      AS num_movies_rank
      FROM movies_directors md JOIN movies
            m ON m.id = md.movie_id
      JOIN directors d ON md.director_id =
            d.id
      JOIN movies_genres g ON  m.id =
            g.movie_id
      JOIN roles r ON r.movie_id = m.id
      WHERE m.rank IS NOT NULL AND g.genre
       = "Comedy"
      GROUP BY director_name) AS m
WHERE num_movies_rank <= 10;
```

Table #8. Average Duration for Query 5.3 (Normalized)

	Duration (seconds)
1	9.075
2	9.346
3	9.552
4	8.494
5	8.866
Average	9.066

As seen on Listing #10, the indexes from multiple tables resulted in the need to use a nested-loop join algorithm. A full table scan on movies_directors with unique key lookups on movies_genres and movies is performed in the inner subquery before the outer query is executed to do a full table scan. The inner subquery makes use of a PARTITION BY expression to divide the query's results based on the genre and in this case, it is the comedy genre. The RANK() function is used to initially sort the number of movies made by each director in descending order. Outside of the subquery, the WHERE clause is used to limit the ranked number of movies per director to the top 10.

Generating the OLAP report using the denormalized data warehouse (Listing #11) without indexes decreased the execution time to an average of 1.9122 seconds.

Listing #11. SQL Statement for Query 5.3 (Denormalized)

```
SELECT director_name, movies_made
FROM (SELECT DISTINCT CONCAT(first_name, '
      ', last_name) AS director_name,
      COUNT(DISTINCT movie_id) AS
           movies_made,
      RANK() OVER (PARTITION BY genre
      ORDER BY COUNT(DISTINCT movie_id)
      DESC) AS num_movies_rank
      FROM facts f JOIN directors d ON
           f.director_id = d.director_id
      WHERE genre = "Comedy"
      GROUP BY director_name) AS m
WHERE num_movies_rank <= 10;
```

Table #9. Average Duration for Query 5.3 (Denormalized)

	Duration (seconds)
1	2.640
2	2.319
3	1.548
4	1.546
5	1.508
Average	1.9122

For a more direct comparison, a similar query to the query shown in Listing #10 for the denormalized data was used as shown in Listing #11. The only difference is that only one JOIN operation was utilized in the query as the design of the data warehouse reduces the need to perform multiple JOINs. Because of this the performance is seen to be noticeably faster.

Generating the OLAP report using the denormalized data warehouse (Listing #12) with indexes brought down the execution time to a final average of 1.7624 seconds.

Table #10. Average Duration for Query 5.3 (Denormalized with index on directors table)

	Duration (seconds)
1	1.887
2	1.5
3	1.494
4	1.496
5	2.435
Average	1.7624

To further optimize the query denormalized data, indexes were implemented as they are known to produce faster queries by creating pointers to data stored within a database (Barnhill, 2021) [3]. A composite index was first made for the director's name as this result requires the combination of two columns from the directors table. With this index alone, the performance improved very slightly as they use b-tree structures to store pointers to the name columns (Santanu, 2023) [11].

Additionally, the performance was increased with another index which was created in the fact table for the movie_id column. The movie_id column is used to count the movies that have been made by a director. A value under the movie_id column can get repeated

multiple times in the fact table because of its combination with the attributes `genre` and `actor_id`. This index helps make those rows more special.

Table #11. Average Duration for Query 5.3 (Denormalized with index on directors table and index on fact table)

	Duration (seconds)
1	1.547
2	1.572
3	1.388
4	1.399
5	1.387
Average	1.4586

5.4 How many action movies were made from 1990 to 2000?

The number of action movies made every year between 1990 to 2000 can be a useful statistic for analyzing the popularity of action movies during the 90's. Moreover, through this query, it can be seen whether the rise in box office revenue for action films in recent years is the result of a technological shift, an improvement in the quality of action movies, or simply an increase in the quantity of action movies.

5.4.1 Output

The output resulted in 11 rows and 2 columns. This can be seen below in Figure #11

year	total_movies
1990	4852
1991	3675
1992	4523
1993	4772
1994	5288
1995	5691
1996	5377
1997	6398
1998	5018
1999	5311
2000	4987

Figure 11. Output for Query 5.4

5.4.2 Optimizations and Statistics

Generating the OLAP report for Query 5.4 using a normalized database (Listing #13) took an average of 1.297 seconds as seen in Table #12.

Listing #13. SQL Statement for Query 5.4 (Normalized)

```
SELECT m.year, COUNT(m.name)AS total_movies
FROM movies m
JOIN movies_genres mg ON m.id = mg.movie_id
JOIN movies_directors md ON md.movie_id =
m.id
JOIN roles r ON r.movie_id = m.id
JOIN actors a ON a.id = r.actor_id
WHERE mg.genre = "Action" AND m.year
BETWEEN 1990 AND 2000 AND m.rank IS NOT
NULL
GROUP BY m.year
ORDER BY m.year;
```

Table #12. Average Duration for Query 5.4 (Normalized)

	Duration (seconds)
1	1.781
2	1.813
3	1.219
4	0.953
5	0.719
Average	1.297

As seen in Listing #13, COUNT () was used to group each movie together based on their year of release. Multiple joins were also used to ensure that each movie had existing genres, directors, roles, and actors. This resulted in the use of temporary tables so that a group by could be performed. Furthermore, due to a lack of indices, the filesort algorithm had to be used, resulting in a slower execution time.

Generating the OLAP report using the denormalized data warehouse (Listing #14) without indexes decreased the execution time to an average of 0.6314 seconds.

Listing #14. SQL Statement for Query 5.4 (Denormalized)

```
SELECT m.year, COUNT(m.name)AS total_movies
FROM movies m
JOIN facts f ON m.movie_id = f.movie_id
WHERE f.genre = "Action" AND m.year BETWEEN
1990 AND 2000 AND m.movie_rank IS NOT NULL
GROUP BY m.year
ORDER BY m.year;
```

Table #13. Average Duration for Query 5.4 (Denormalized)

	Duration (seconds)
1	1.617

2	0.375
3	0.389
4	0.394
5	0.382
Average	0.6314

In comparison, it can be seen in Table #13 that using a similar query on denormalized data produces a much shorter average time than on normalized data. This is due to the lack of JOINs needed, since the structure of the data warehouse used is designed to eliminate that issue.

Table #14. Average Duration for Query 5.4 (Denormalized with index on the fact table)

	Duration (seconds)
1	0.281
2	0.296
3	0.250
4	0.265
5	0.235
Average	0.2654

Finally, the use of indices on denormalized data in the same query shows a faster average speed than without, as seen in Table #14. An index was made on the genre column of the fact table, as the query requires that the genre of the selected data be equal to action.

6. Results and Analysis

The findings and analysis from different functional and performance tests executed to confirm the accuracy of the ETL script and OLAP queries, as well their execution time, are presented in this section.

6.1 Functional Testing

This section discusses the functional testing done to ensure the correctness of the ETL process and OLAP operations.

6.1.1 Correctness of the ETL Process

For the ETL process, the number of records returned in the fact table of the denormalized schema was compared to the number of records generated from the SQL query that created the fact table from the normalized dataset. Due to the previously mentioned issues encountered in the ETL process, not all records were loaded from the source dataset, thus causing data ambiguity. Additionally a manual check was performed on both results to make sure that the data was the same. More specifically, the denormalized data was checked to ensure that only movies with ranks were loaded into the data warehouse.

6.1.2 Correctness of the OLAP operations

For the OLAP operations, both normalized and denormalized data were put through the same operations in the OLAP application. Correctness was ensured by making sure that both created the same resulting graphs. The results were also compared to the results from running their queries in SQL Workbench in making sure that the number of records returned were the same. Random sample records were manually checked between the OLAP application and the SQL query results to ensure that both results were the same.

Due to data ambiguity from the issues experienced in the ETL process, some records were found to be different between the normalized and denormalized data, especially with roll-up and drill-down operations that look at all data for their reports. However, the differences between the two results overall were miniscule and did not seem to create a significant difference in results.

6.2 Performance Testing

The performance of the queries were determined by the duration (seconds) taken to execute them. As shown in Table #15, the queries were tested among three kinds of datasets: normalized data (the original dataset), denormalized data (the data warehouse), and denormalized data with optimization (the data warehouse with implemented indexes). Underlined values in the table highlight the faster durations per query.

Table #15. Performance Testing Results (in seconds)

Query	**Normalized**	**Denormalized**	**Denormalized with Indexes**
Roll-up	11.537	6.344	N/A
Drill Down	20.6782	13.2638	N/A
Slice	9.066	1.9122	1.4586
Dice	1.297	0.6314	0.2564

Based on the results, it can be observed that the denormalized data results in faster queries. The original dataset follows a snowflake schema. Because of that design, some of its biggest disadvantages are that it requires more joins compared to a star schema and that there's higher data redundancy due to normalization (Simplilearn, 2022) [12].

When it comes to optimization, it can be observed that the use of indexes is effective on the slice and dice OLAP operations. However, roll-up and drill-down are already optimized due to denormalization. Implementing indexes does not improve the performance because it would still do a full table scan.

6.3 Performance of the OLAP Application

To measure the performance of the OLAP application, the specification provided in the rubric was used wherein the application must load under 5 seconds.

Reports in Tableau can be made by first importing the data source and dragging the tables from the source into the workbook. Tableau's algorithm would automatically search for the foreign key if multiple tables were dragged and they would join them. After doing so, different types of data visualizations can be generated by dragging and dropping columns from imported tables into the workspace. However, doing this can be slow every time an attribute is dropped into the visualization as Tableau would take around 5 seconds to 30 seconds to adjust to the new changes.

To generate reports faster, custom SQL queries, more specifically the queries from Section 5 were placed in the workbook to generate a variety of visualizations that would complement the reports.

After generating the desired visualizations and placing them into dashboards, it was important that the data was extracted and not connected to a live host. This is so that the source code can be extracted and the dashboard can be published to Tableau public.

To test the speed of the application, all members loaded the application on Tableau Desktop and the application was able to load in under 5 seconds on all devices. Additionally, the interactive parts of the dashboard were reported to be working without errors.

7. Conclusion

In performing the project, many learnings were made through the different tools that were used and realizations after exploring the data.

In designing the data warehouse schema, it was important to plan out the types of reports that would be presented. This is so that only the necessary data is shown and so that the speed of queries can be fast.

In creating the pipeline for ETL, the configuration of processors was essential so that memory would not be overloaded and so that the ETL process was fast. However, the experience of bottlenecks prevented the fact table in the data warehouse from being complete. Thus, data integrity was compromised and the reports through denormalized data cannot be completely compared to the normalized source dataset.

Through the use of Tableau in creating the OLAP Application, the different functionalities of the tool opened up discoveries to the many ways data can be visualized. Through joins, filters, parameters, and other methods in organizing data, an interactive application was created so that users can analyze the dataset and learn from the information it holds.

In query processing, the effects of querying normalized and denormalized data were presented per query. Overall, denormalized data queries faster than normalized data. In optimizing queries, different types of indexes can increase or decrease the overall performance. It is dependent on the query itself and what OLAP operations used.

Aside from the understandings on the technology the team used, the insights gathered on the film industry would provide beneficial information for the development of their future projects and improvement based on the data extracted through the years, genres, actors, and directors. Not only for people under the film industry who can partake in the benefits of this analysis but also for consumers or moviegoers. For example, present and future filmmakers can be aware of the gender equality among actors and directors who are provided with much more opportunities than the latter gender. Furthermore, consumers can do some research under the top directors of the comedy genre that shows how good the directors are on their artistic design. Which can be the metric of how worth it is to watch the director's films.

Working around the production side of films, directors, producers, and casting crew can look over cultural shifts to understand the increase of popularity of genres in a given decade or year. Such as the years between 1990 to 2000, where action movies were brought into attention that distinguished what audiences should look out for in upcoming seasons. Through these queries and insights, more users would be able to gather more information that was not seen on a granular level. This can serve to describe the past, identify the challenge of the present, and be able to predict the future of the film or media industry and to their audiences.

8. References

[1] Adamson, C. (2012). Mastering Data Warehouse Aggregates. John Wiley & Sons. Retrieved from https://www.wiley.com/en-us/Mastering+Data+Warehouse+Aggregates%3A+Solutions+for+Star+Schema+Performance-p-9781118429181

[2] Apache Software Foundation. (n.d). Retrieved from Apache.org website: https://nifi.apache.org/docs/nifi-docs/components/org.apache.nifi/nifi-standard-nar/1.6.0/org.apache.nifi.processors.standard.GenerateTableFetch/

[3] Barnhill, B. (2021). How to use Indexing to Improve Database Queries. Retrieved from The Data School website: https://dataschool.com/sql-optimization/how-indexing-works

[4] Blanchard D. C., Graczyk B., Blanchard R. J. (1986). Differential reactions of men and women to realism, physical damage, and emotionality in violent films. Aggress. Behav. 12 45–55. 10.1002/1098-2337198612:1

[5] Chen, J. (2021, March 14). What Does A Film Director Do: Everything You Need To Know - NFI. Retrieved from NFI website: https://www.nfi.edu/what-does-a-film-director-do/

[6] Hampel, M. (2021, April 9). What is database denormalization and how it can help you? | DS Stream. Retrieved from Data Analytics website: https://dsstream.com/what-is-database-denormalization/

[7] IMDb Dataset. (2014). from Cvut.cz https://relational.fit.cvut.cz/dataset/IMDb

[8] MySQL :: MySQL 5.6 Reference Manual :: 8.2.1.14 GROUP BY Optimization. (2023). Retrieved from Mysql.com website: https://dev.mysql.com/doc/refman/5.6/en/group-by-optimization.html

[9] [NIFI-3422] Run Once Scheduling - ASF JIRA. (2017, January 31). Retrieved from Apache.org website: https://issues.apache.org/jira/browse/NIFI-3422

[10] Praise for A Developer's Guide to Data Modeling for SQL Server. (n.d.). Retrieved from http://www.cherrycreekeducation.com/bbk/b/Addison_Wesley_A_Developers_Guide_to_Data_Modeling_for_SQL_Server_Jul_2008.pd

[11] Santanu, B. (2023, January 5). Composite Index in SQL - Scaler Topics. Retrieved from Scaler Topics website: https://www.scaler.com/topics/sql/composite-index-in-sql/

[12] Simplilearn. (2022, December 27). What is Snowflake Schema in the data warehouse model? Retrieved from

Simplilearn website: https://www.simplilearn.com/snowflake-schema-in-data-warehouse-model-article

[13] Smallcombe, M. (2023, February 15). Snowflake Schemas vs Star Schemas: What Are They and How Are They Different? Retrieved from Integrate.io website: https://www.integrate.io/blog/snowflake-schemas-vs-star-schemas-what-are-they-and-how-are-they-different/

9. Appendix

For reference to the OLAP application, this is a public URL to view the dashboard: https://public.tableau.com/app/profile/lauren.garcia/viz/STADVDBMCO1Group16OLAP/Dashboard12?publish=yes