Steam Games Query Processing: A Technical Report

Jon Piolo C. Jacinto[1], Marion Jose S. Manipol[2], and Carl Gabriel C. Yap[3]

De La Salle University Manila

[1]jon_jacinto@dlsu.edu.ph, [2]marion_manipol@dlsu.edu.ph, [3]carl_gabriel_yap@dlsu.edu.ph

ABSTRACT

This report details the application of Online Analytical Processing (OLAP) to an instructor-provided dataset, intending to teach students about key concepts in data warehousing and query optimization. First, we design a star schema (Appendix A) to organize the data for efficient querying. Next, we develop an OLAP application that generates analytical reports based on multidimensional queries. After running several queries, we evaluate their performance and identify areas for improvement. Finally, we propose optimization strategies to enhance query efficiency, such as indexing and partitioning. This project provides hands-on experience in building data warehouses, working with OLAP tools, and optimizing queries, helping students gain skills they can apply to real-world data analysis tasks.

Keywords

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

1. Introduction

Online Analytical Processing (OLAP) plays a pivotal role in enabling the analysis of large datasets, particularly in scenarios where multidimensional queries and trend analyses are required. Leveraging OLAP with a robust data warehousing infrastructure allows for the efficient organization, storage, and retrieval of data for complex business intelligence tasks. The process begins with Extract, Transform, Load (ETL) procedures, where raw data from various sources is consolidated, transformed into a standardized format, and stored within a data warehouse. This structured data can be utilized for comprehensive analyses, providing valuable insights into underlying trends and patterns.

In this study, we explore the application of OLAP to the Steam Games DB, a dataset comprising game metrics, user scores, and categories for over 97,000 titles. This dataset offers a rich source of information on game performance, user engagement, and industry trends. By applying OLAP methodologies to this dataset, we aim to uncover actionable insights to inform recommendation systems, market analyses, and predictive modeling within the gaming industry.

2. Data Warehouse

This section outlines the influential design choices and implementation of the star schema for the data warehouse. The design includes two regular dimensions, discussed in Section 2.1, followed by two special dimensions in Section 2.2. Lastly, Section 2.3 presents the structure and details of the central fact table.

2.1 Company and Game Data

In the gaming industry, it is essential to distinguish between the companies responsible for development and those handling publishing. Games from the same developer may be published by different entities, necessitating clear separation of these data points. For example, both Sekiro™: Shadows Die Twice - GOTY Edition and Elden Ring were developed by FromSoftware Inc., yet the former was published by Activision, while the latter was by Bandai Namco Entertainment. Similarly, a publisher may oversee a diverse portfolio of games developed by different studios. Electronic Arts (EA), for instance, publishes both Apex Legends™, developed by Respawn Entertainment, and The Sims™ 4, developed by Maxis. This differentiation highlights the need for precise data retrieval regarding development and publishing roles.

Table 1. Columns of Company dimension

Column Name	Data Type
companyID (PK)	INT
developer	VARCHAR(255)
publisher	VARCHAR(255)

In addition to separating development and publishing data, fundamental information such as a game's title, release date, and description is crucial for players. These details, along with filters for categories, genres, and platform-curated tags, enhance the user experience by helping gamers discover titles within their favorite niches or explore new genres. To optimize search performance, particularly for filtering by categories, genres, and tags, we employed FULLTEXT indices. This approach provides more efficient matching compared to wildcards in the LIKE operator, which results in full table scans [1].

GitHub repo: github.com/carl-yap/STADVDB-Group4MCO

Table 2. Columns of Game dimension

Column Name	Data Type
gameID (PK)	INT
name	VARCHAR(255)
aboutTheGame	VARCHAR(255)
releaseDate	DATE
websiteURL	VARCHAR(255)
supportURL	VARCHAR(255)
supportEmail	VARCHAR(255)
supportedLanguages	TEXT
fullAudioLanguages	TEXT
categories **	TEXT
genres **	TEXT
tags **	TEXT

** Written in SQL as 'CREATE FULLTEXT INDEX *column_name* ON dim_game(*column_name*)'

2.2 OS Support

The data source includes three mysterious columns: Windows, Mac, and Linux. Upon closer inspection, it becomes evident that these columns represent the compatibility status for each OS. This insight leads to the proposal of consolidating these columns into a set of eight unique IDs, corresponding to all possible binary combinations across the three platforms, streamlining references and queries related to operating system support.

Table 3. Columns of OS dimension

Column Name	Data Type
osID (PK)	VARCHAR(3)
windowsSupport	TINYINT(1)
macSupport	TINYINT(1)
linuxSupport	TINYINT(1)

2.3 Game Metrics

This table serves as the fact table, distinct from the game dimension, as it focuses on significant metrics for stakeholders such as investors, game companies, and informed gamers. Rather than game details, it captures key data points like rankings, scores, and playtime statistics. These metrics can be cross-referenced with the associated dimensions to generate comprehensive analytical reports.

Table 4. Columns of fact table

Column Name	Data Type
gameID ***	INT
companyID ***	INT
osID ***	VARCHAR(3)
price	FLOAT
peakCCU	INTINT
achievementCount	INT
averagePlaytimeForever	FLOAT
medianPlaytimeForever	FLOAT
estimatedOwners	VARCHAR(255)
dlcCount	INT
metacriticScore	INT
userScore	FLOAT
positive	INT
negative	INT
scoreRank	INT
recommendations	INT

*** Written in SQL as 'FOREIGN KEY (*fk_column*) REFERENCES *dim_table_nam*e(*fk_column*)'

3. ETL Script

This section presents pseudocode for the Extract, Transform, and Load (ETL) processes used to populate the database tables described in the previous section. As the name suggests, these processes extract raw data from the source, transform it to ensure consistency and integrity, and load it into the appropriate tables. They ensure that essential information, such as game details, development and publishing relationships, and platform compatibility, is accurately represented and efficiently stored for querying and analysis.

Listing 1. Creating dim_game

```
Create game dimension table. It has columns with generic
information about the game. Set the genre, categories,
and tags to fulltext indexes.

Create a temporary table with the same columns.
Populate it with the CSV. Insert the content from the
temporary table into the dimension table. Format
releaseDate into date while doing so.

Drop the temporary table.
```

Listing 2. Creating dim_company

Create a company dimension table. It stores developers and publishers.

Create a temporary table with the same columns. Populate it with the CSV. Insert the content from the temporary table into the dimension table. Exclude existing entries in the dimension table while populating.

Drop the temporary table.

Listing 3. Creating dim_os

Create OS dimension table. It stores the supported OS games. Populate it with all possible combinations of data.

Listing 4. Creating fact_gamemetrics

Create the fact table. It stores the metrics of all games. It references the game, company, and OS dimensions.

Populate the foreign key referencing the game dimension based on the game's ID.

Populate the foreign key referencing the company dimension based on the developers and publishers of the game.

Populate the foreign key referencing the OS based on the game's supported OS.

4. OLAP Application

For this application, we developed a Python class, SteamDB, which leverages the SQLAlchemy library to interface with a MySQL database. This class includes methods for creating tables and inserting transformed data into the database. A key method is provided to execute SQL queries stored in string variables, enabling flexible query execution. The application is implemented as a Jupyter Notebook, which guides users through the underlying processes and offers instructions on crafting custom queries. Both the notebook and accompanying source code, along with the relevant data, are available in the associated GitHub repository.

Listing 5. OLAP 1 - Revenue Generated by Company

```
SELECT c.developer, c.publisher,
       SUM(f.price * f.estimatedOwners) AS
       estimatedRevenue
FROM fact_GameMetrics f
JOIN dim_game g
       ON f.gameID = g.gameID
JOIN dim_company c
       ON f.companyID = c.companyID
GROUP BY c.developer, c.publisher
ORDER BY estimatedRevenue DESC
```

Listing 6. OLAP 2 - Revenue by OS Compatibility

```
SELECT os.osID,
       SUM(f.price * f.estimatedOwners)
       AS totalRevenue,
       AVG(f.averagePlaytimeForever)
       AS avgPlaytime
FROM fact_GameMetrics f
JOIN dim_OS os
       ON f.osID = os.osID
GROUP BY os.osID
ORDER BY totalRevenue DESC
```

Listed above are some queries generating game revenue-focused reports.

5. Query Processing and Optimization

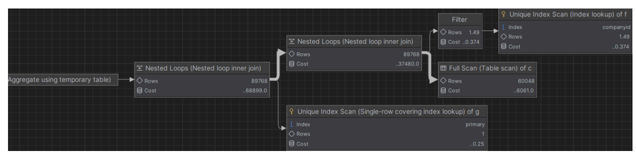

Figure 1. Revenue Generated by Developers and Publishers

The figure above illustrates the execution plan for the first query in the notebook, which involves two nested loops due to using two joins. However, there is potential for optimization. By aggregating the revenue before performing the joins, the query reduces the number of required joins and decreases the volume of rows processed in subsequent operations. This optimization not only simplifies the execution plan but can also lead to improved performance by minimizing unnecessary data handling.

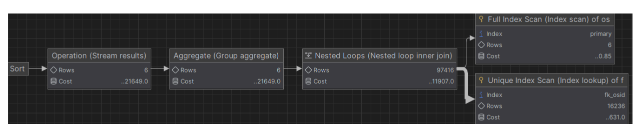

Figure 2. Query for Revenue Generated Based on Supported OS

The optimization involved preprocessing the data before querying. We used binary encoding and pre-populated the dataset with all possible OS support combinations, reducing complexity and improving query efficiency.

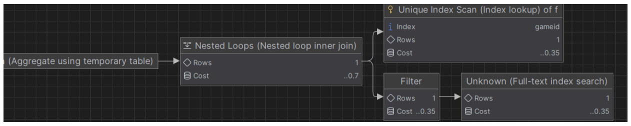

Figure 3. Query for Revenue Generated by Genre by Year With Fulltext Indexing

In our next query, we employed full-text indexing to address how genres were formatted in the CSV file. Since genres were listed collectively rather than individually, full-text indexing was particularly suited to this scenario, as it is designed to handle cases where entire words are combined into a single text entry. This approach enabled us to query the genre data despite its non-standard format.

6. Results and Analysis

This section discusses query execution times, identifies the causes of delays, and compares improvements achieved through optimization.

Table 5. Execution of OLAP 1

Before optimization (in ms)	**After optimization** (in ms)
633	236
657	248
627	243

Table 6. Execution of OLAP 2

Using LIKE (in ms)	**Using MATCH** (in ms)
281	274
277	273
266	253

From the tables above, it is evident that the first query shows a significant increase in speed post-optimization, achieving nearly 200% improvement in execution time. For the second query, using MATCH instead of LIKE appears to offer a slight improvement in execution time.

In the first OLAP application, aggregating annual revenue before performing the JOIN operation contributed to faster processing, as the JOIN operation was applied to fewer rows. For the second query, the improvement could be attributed to full-text indexing, which is generally more effective for text fields concatenated into a single, continuous entity, similar to the structure of the CSV storage.

7. Conclusion

In conclusion, our work with manipulating raw data and populating the data warehouse highlighted the critical importance of understanding different data types and selecting appropriate handling strategies. Initially, binary encoding was considered for genres, tags, and categories, but this approach would have resulted in numerous tables with only boolean values. Instead, we chose to store these text-heavy entries directly and leveraged full-text indexing to improve readability and searchability, while restricting binary encoding to OS support for efficiency. Additionally, we observed that the most intuitive queries are not always the most efficient. For instance, a query summarizing game company revenue required two joins, which led to nested loops and slower performance. Pre-aggregating the revenue data would have reduced the number of joins and improved the overall query performance by minimizing row processing.

8. References

[1] Microsoft Learn. (n.d). *Full-text search overview*. https://learn.microsoft.com/en-us/sql/relational-databases/search/full-text-search?view=sql-server-ver16

A. OLAP Query Visualization

The following visualizations were generated using IPython and pandas in a Jupyter notebook.

developer	publisher	estimatedRevenue
Game Science	Game Science	2,999,500,083
Amazon Games	Amazon Games	1,999,500,083
Valve	Valve	1,364,569,971
FromSoftware Inc.	FromSoftware Inc.,Bandai Namco Entertainment	1,199,800,033
CD PROJEKT RED	CD PROJEKT RED	1,010,294,533

Results of OLAP 1 in table format.

osID	totalRevenue	avgPlaytime
100	43,008,091,229	82.694550
111	8,997,528,875	142.050445
110	4,755,858,975	121.180269
101	903,791,498	74.268617
010	1,918,099	1553.960000
001	599,799	4.833333
011	249,499	341.000000

Results of OLAP 2 in table format.