INTRO TO RD & SQL PROGRAMMING REPORT

ON

COMIC STORE DATABASE PROJECT

UCD School Of Computer Science
University College Dublin
COMP40725-INTRO TO RD & SQL PROGRAMMING REPORT

Submitted by:
Rishabh Mer, 22200186

Submitted to:
Professor Tony Veale

TABLE OF CONTENTS

LIST OF FIGURES

INTRODUCTION

Comic books have been a popular form of entertainment for many decades. From superheroes to science fiction to fantasy, there is a comic book series for almost every interest. Comic book stores have played a significant role in the distribution and collection of comic books, providing a central location for fans to purchase, trade, and discuss their favourite series [1].

In this report we have developed a comic store database. In this project we explore the inner workings of comic book store including database structure used to manage comic books, customers data and its orders.

The data involved in the project includes information about comic book series, individual comic books, customers, order and writers. The database will be designed to handle a significant amount of data, including information on hundreds of comic book series, thousands of individual comic books, and thousands of customers.

The comic store's success depends on its ability to manage its operations effectively. To support this goal, a well-designed database will be implemented to serve as the foundation of the store's data management system. By enabling tracking of inventory levels, monitoring of customer purchases, and generating sales reports, the database will help the store to optimise its performance. Centralising data storage and management through the database will streamline operations and support efficient management of all aspects of the store's business.

Figure1. Comic Cover Art

DATABASE PLAN: A SCHEMATIC VIEW

Class Diagram:

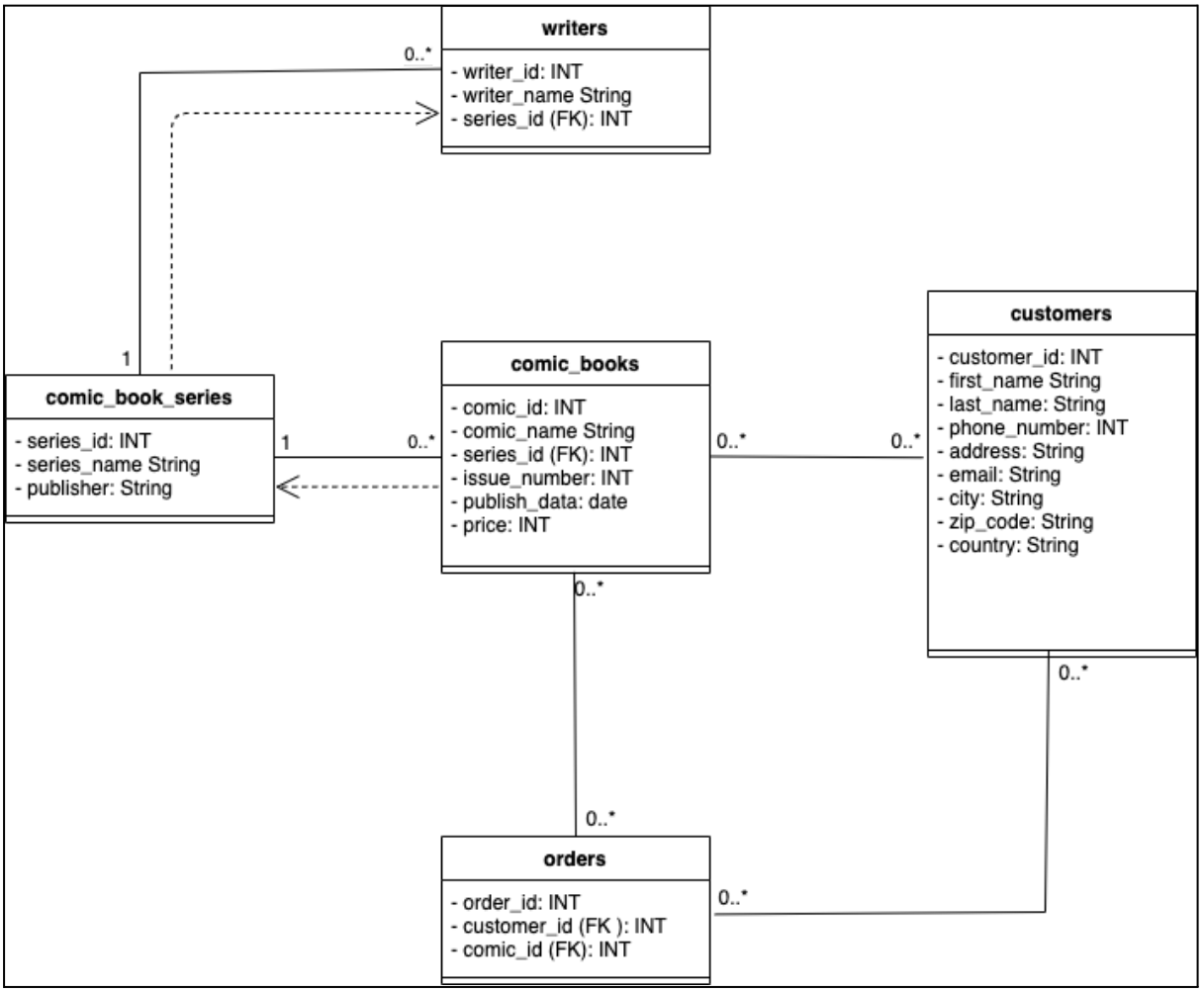


Figure 2: Class Diagram for Comic Store Database

The class diagram represents the object-oriented design of the comic store's database (Figure 2). It provides the visual representation of classes involved in the system and their relationships. Also class diagram serves as a blueprint for the database, for providing a clear understanding of the structure and relationships of the data entities involved [2].

The classes in the above diagram include:

- Comic book series
- Comic books
- Writers
- Customers
- Orders

The relationships between the classes are also represented, such as:

- One - to - many relationships between comic book series and comic books, which indicates comic series can have multiple comic books.
- Many - to - many relationships between comic books and customers which represents the fact that a customer can purchase multiple comic books as well as a comic book can be purchased by multiple customers.
- One - to - Many relationship between comic book series and writers, which indicates that a comic series can have multiple writers
- Many - to - Many relationships between comic books and orders, which indicates that many comic books can be purchased by customers as well as many number of customers purchase comic books
- Many - to - Many relationships between orders and customers, which represents customer can make multiple orders and an order an have multiple customers

ER Diagram:

An Entity-Relationship diagram is a graphical representation of the entities and relationships in a database system. It is used to model the real world entities and their relationship in a database. It consists of entities, attributes and relationships between entities [3].

The ER diagram for the comic store database (Figure 3) includes entities such as comic book series, comic books, writers, customers and orders with its key attributes such as comic_id, series_id, writers_id, customer_id and order_id.

These key attributes are used to uniquely identify each entity. The relationships between these entities are represented using foreign keys.

For example, the Orders entity has foreign keys for customer_id and comic_id, which represent the relationship between Customers and Orders, and between Comic Books and Orders, respectively. Similarly, the Comic Books entity has a foreign key for series_id, which represents the relationship between Comic Book Series and Comic Books.

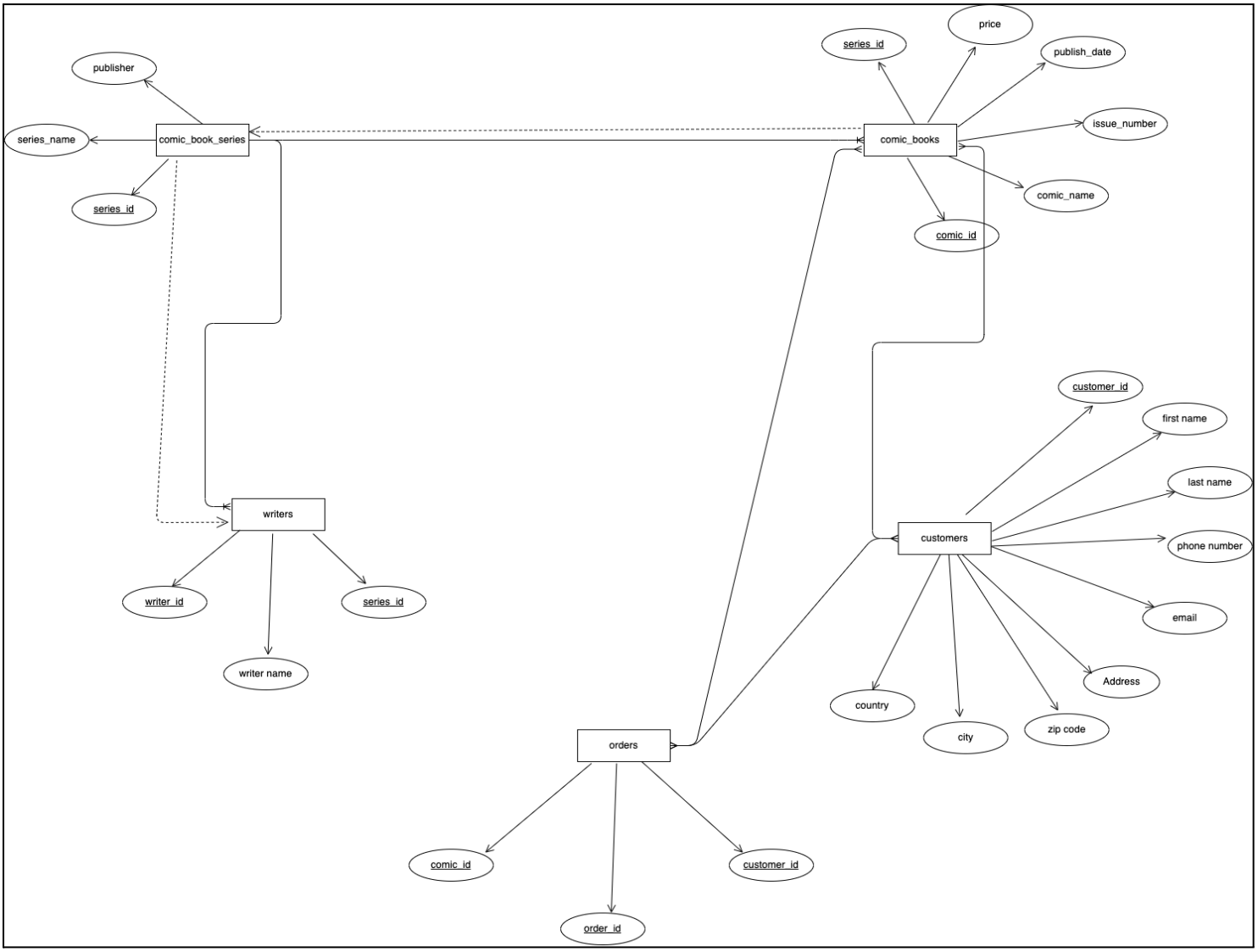


Figure 3: ER Diagram of Comic Store Database

DATABASE STRUCTURE: A NORMALISED VIEW

The comic store database consists of five tables: comic_book_series, comic_books, writers, customers, and orders.

Comic_book_series: The comic_book_series table holds information about comic book series which includes the series id, series name and publisher.

Comic_books: The comic_books table stores information about individual comic books, such as the comic ID, name, issue number, publish date, and price. The table also includes a foreign key that references the comic_book_series table to establish a relationship between the two tables.

Writers: It contains information about the writers of the comic book series. It includes a writer id, writer name, and series id as a foreign key, which links the writers to the specific comic book series they work on

Customers: This table holds information about the customers, including the customer id, first and last name, phone number, email, address, city, zip code, and country.

Orders: The orders table stores information about orders placed by customers. It includes the order id, customer id as a foreign key, comic id as a foreign key, and order date. The customer id and comic id are linked to the customers and comic_books tables, respectively.

All the above are in BCNF and 3 NF form by:

- The comic_book_series table has a single candidate key - series_id, which is also the primary key. There are no attributes that do not belong to any candidate key, so this table is in BCNF [4].
- The comic_books table has a primary key - comic_id and a foreign key series_id. All non-key attributes depend only on the whole primary key, so this table is in 3NF [4].
- For writers table is the same as the comic_books tables, all non-key attributes depend only on the whole primary key, so the table is in 3NF [4].
- The customer table is in BCNF form as it has a primary key - customer_id and no non-prime attributes.
- The orderS tables is in 3NF as it has two foreign keys - customer_id and comic_id and all non-key attributes depend only on the whole primary key [4].

DATABASE VIEWS

SQL views are similar to virtual tables, and they have rows and columns same as real tables in SQL. Views are created by selecting fields from one or more tables in the database, can contain all rows or specific rows based on certain conditions [5]

➢ Creating view for total amount spent on comic books by each customer

Query:

```
CREATE VIEW customer_purchases AS
SELECT
    customers.customer_id,
    CONCAT(customers.first_name, ' ',  customers.last_name) AS customer_name,
    FORMAT(SUM(comic_books.price), 2) AS total_spent
FROM
    customers
    JOIN orders ON customers.customer_id = orders.customer_id
    JOIN comic_books ON orders.comic_id = comic_books.comic_id
GROUP BY
    customers.customer_id;
```

Figure 4: view query 1

This view is created to display the total amount money spent by each customer in comic store. It is achieved by joining three tables customers, orders and comic_books.
Firstly selecting the customer_id, first_name and last_name from customer table. Concatenating the customer's name by using CONCAT and summing up the prices of all the comics books that each customer has purchased using SUM() function. The view then groups the results by customer_id using the GROUP BY clause.

Output:

```
mysql> select * from customer_purchases;
+-------------+-------------------+-------------+
| customer_id | customer_name     | total_spent |
+-------------+-------------------+-------------+
|           1 | Rishabh Mer       | 74.84       |
|           2 | Albert Ress       | 23.45       |
|           3 | Thanos Titan      | 41.91       |
|           4 | Jess Nolan        | 49.38       |
|           5 | Alex Wright       | 91.82       |
|           6 | Tony Stark        | 4.99        |
|          12 | Tyrion Lanister   | 8.48        |
|          10 | Parthiv Prabhakar | 4.99        |
|          13 | Tywin Lanister    | 11.97       |
|          20 | Joey Tribbiani    | 28.94       |
|          14 | Shey Ryan         | 4.99        |
|           9 | Peter Parker      | 13.47       |
|          17 | Natasha Romanoff  | 5.49        |
|           7 | Steve Rogers      | 8.98        |
|          18 | Nick Fury         | 8.98        |
|          19 | Ross Geller       | 8.48        |
|          11 | Robert Wales      | 1.99        |
|          15 | Brie Larson       | 7.96        |
+-------------+-------------------+-------------+
18 rows in set (0.00 sec)
```

Figure 5: view output 1

➢ Creating view for total comic book written by each writer

Query:

```sql
-- Creating view for total comic book written by each writer
CREATE VIEW comic_count_by_writers AS
SELECT
    writers.writer_name,
    COUNT(comic_id) AS comic_count
FROM
    writers
    JOIN comic_book_series ON writers.series_id = comic_book_series.series_id
    JOIN comic_books ON comic_book_series.series_id = comic_books.series_id
GROUP BY
    writers.writer_id;
```

Figure 6: view query 2

This view creates a tables that shows the count of comics associated with each writes in the databases. It does this by joining the writers table with the comic_book_series and comic_books tables using their respective foreign keys.
It then groups the results by writer and counts the number of comics associated with each writer.

Output:

```
mysql> select * from comic_count_by_writers;;
+--------------------+-------------+
| writer_name        | comic_count |
+--------------------+-------------+
| Jason Aaron        |          10 |
| Gerry Duggan       |           6 |
| Ta-Nehisi Coates   |           5 |
| Murewa Ayodele     |           3 |
| Jeff Lemire        |           7 |
| Benjamin Percy     |           6 |
| Donny Cates        |           7 |
| Donny Cates        |           5 |
| Kieron Gillen      |          12 |
| Ryan North         |           7 |
| Alan Moore         |           1 |
| Christopher Priest |           2 |
| Mark Waid          |          10 |
| Joshua Williamson  |          11 |
| Scott Snyder       |           6 |
| Scott Snyder       |           7 |
| Warren Ellis       |          10 |
| Tim Sheridan       |           4 |
+--------------------+-------------+
18 rows in set (0.00 sec)
```

Figure 7: view output 2

➢ Creating view for customer buying comic books which are published on same date

Query:

```
CREATE VIEW same_day_purchase AS
SELECT
    CONCAT(customers.first_name, ' ',  customers.last_name) AS customer_name,
    CONCAT(comic_books.comic_name) AS comic_books,
    orders.order_date,
    comic_books.publish_date
FROM
    customers
    JOIN orders ON customers.customer_id = orders.customer_id
    JOIN comic_books ON orders.comic_id = comic_books.comic_id
WHERE
    orders.order_date = comic_books.publish_date;
```

Figure 8: view query 3

This view same_day_purchase is created to provide the information about customers who purchased a comic book on the same day it was published. The data is retrieved by joining the customers, orders , and comic_books tables and filtering the results to include only the orders where the order date is equal to the publish date of the comic book. This view can be used to analyze customer behavior and sales trends for newly released comic books.

Output:

```
mysql> select * from same_day_purchase;
+---------------+-------------------+------------+--------------+
| customer_name | comic_books       | order_date | publish_date |
+---------------+-------------------+------------+--------------+
| Thanos Titan  | Infinity Wars #1  | 2018-08-01 | 2018-08-01   |
| Thanos Titan  | Infinity Wars #2  | 2018-08-25 | 2018-08-25   |
| Thanos Titan  | Infinity Wars #3  | 2018-09-09 | 2018-09-09   |
| Thanos Titan  | Infinity Wars #4  | 2018-09-30 | 2018-09-30   |
| Thanos Titan  | Infinity Wars #5  | 2018-10-14 | 2018-10-14   |
| Thanos Titan  | Infinity Wars #6  | 2018-10-31 | 2018-10-31   |
| Nick Fury     | Flash #50         | 2019-01-25 | 2019-01-25   |
| Joey Tribbiani | Captain America #4 | 2018-08-30 | 2018-08-30   |
| Joey Tribbiani | Captain America #5 | 2018-09-10 | 2018-09-10   |
| Joey Tribbiani | I Am Iron Man #1  | 2023-03-01 | 2023-03-01   |
+---------------+-------------------+------------+--------------+
10 rows in set (0.01 sec)
```

Figure 9: view output 3

➢ Creating view for total comic sell by publisher

Query:

```
-- comic sell by publisher
CREATE VIEW comic_book_sold_publisher AS
SELECT
    comic_book_series.publishier,
    COUNT(orders.comic_id) AS total_comic_sold
FROM
    comic_book_series
    JOIN comic_books ON comic_book_series.series_id = comic_books.series_id
    JOIN orders ON comic_books.comic_id = orders.comic_id
GROUP BY
    comic_book_series.publishier;
```

Figure 10: view query 4

This view retrieves the total numbers of comic books sold by publisher. It joins the comic_book_series table with the comic_books and order tables to retrieve the publisher information and sales data.
Then by using GROUP BY clause calculating the total number of comics sold by using COUNT function

Output:

```
mysql> select * from comic_book_sold_publisher;
+------------+------------------+
| publishier | total_comic_sold |
+------------+------------------+
| MARVEL     |               72 |
| DC COMICS  |               17 |
+------------+------------------+
2 rows in set (0.00 sec)
```

Figure 11: view output 4

PROCEDURAL ELEMENTS

PL/SQL is a combination of SQL and procedural programming language features, allowing for the declaration of variables, procedures, functions, and triggers, and is processed by the PL/SQL engine in Oracle [6].

➢ Procedure for generating revenue

Query:

```
-- Procedure for generating revenue
CREATE PROCEDURE `calculate_total_revenue`()
BEGIN
    DECLARE total_revenue FLOAT(10,2);
    SELECT SUM(comic_books.price) INTO total_revenue FROM comic_books
    JOIN orders ON comic_books.comic_id = orders.comic_id;
    SELECT CONCAT('Total revenue generated by selling the comics: $', total_revenue) AS 'Total Revenue';
END
```

Figure 12: PL/SQL query 1

The SQL code create a stored procedure that calculates the total revenue Generated from selling comics.

First we declare a local variable name total_revenue of type float. Then executes a select statement that calculate the sum of price column in the comic books table which is joined with orders tables on comic_id column.

Calculated total is then stored in the total_revenue variable.

Output:

```
mysql> CALL calculate_total_revenue();
+-----------------------------------------------------+
| Total Revenue                                       |
+-----------------------------------------------------+
| Total revenue generated by selling the comics: $401.11 |
+-----------------------------------------------------+
1 row in set (0.01 sec)

Query OK, 0 rows affected (0.01 sec)
```

Figure 13: PL/SQL output 1

➢ Procedure query for generating top-selling comic book series

Query:

```
-- Procedure query for genreating top-selling comic book series
CREATE PROCEDURE `get_top_selling_series`(IN num_results INT)
BEGIN
    SELECT
        comic_book_series.series_name,
        COUNT(orders.comic_id) AS total_sold
    FROM
        comic_book_series
        JOIN comic_books ON comic_book_series.series_id = comic_books.series_id
        JOIN orders ON comic_books.comic_id = orders.comic_id
    GROUP BY
        comic_book_series.series_id
    ORDER BY
        total_sold DESC
    LIMIT num_results;
END
```

Figure 14: PL/SQL query 2

In this procedure, it takes an input parameters num_results. The procedure selects the top num_results comic book series based on the total number of comics sold for each series. The procedure uses joins between the comic_book_series, comic_books and orders tables to gather the necessary information. The results is then grouped by series_id column.

Output:

```
mysql> CALL get_top_selling_series(10);
+------------------------+------------+
| series_name            | total_sold |
+------------------------+------------+
| Avengers-(2018)        |         26 |
| Infinity Wars          |         12 |
| Captain America-(2018) |          7 |
| The Authority-(1999)   |          7 |
| Fantastic Four-(2022)  |          6 |
| King in Black          |          5 |
| Eternals-(2021)        |          5 |
| I Am Iron Man-(2023)   |          4 |
| Ghost Rider-(2022)     |          4 |
| Flash-(2019)           |          4 |
+------------------------+------------+
10 rows in set (0.00 sec)
```

Figure 15: PL/SQL output 2

➢ Procedure query for generating top-customer purchasing comic books

Query:

```
-- Procedure query for generating top-customer purchasing comic books
CREATE PROCEDURE `get_top_customer`()
BEGIN
    SELECT
        CONCAT(customers.first_name, ' ', customers.last_name) AS customer_name,
        COUNT(orders.order_id) AS total_orders
    FROM
        customers
        JOIN orders ON customers.customer_id = orders.customer_id
    GROUP BY
        customers.customer_id
    ORDER BY
        total_orders DESC
    LIMIT 1;
END;
```

Figure 16: PL/SQL query 3

The above code retrieves the top customer with the highest number of orders. It is performed using SELECT query on the customer and orders tables and uses COUNT function to count the number of orders for each customers and LIMIT the output to only one record.

Output:

```
mysql> CALL get_top_customer();;
+---------------+--------------+
| customer_name | total_orders |
+---------------+--------------+
| Alex Wright   |           18 |
+---------------+--------------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)
```

Figure 17: PL/SQL output 3

➢ Procedure query for generating total comic by writers

Query

```
-- Procedure query for generating total comic by writers
CREATE PROCEDURE `get_comic_count_by_writer`()
BEGIN
    SELECT
        writers.writer_name,
        COUNT(comic_books.comic_id) AS comic_count
    FROM
        writers
        JOIN comic_book_series ON writers.series_id = comic_book_series.series_id
        JOIN comic_books ON comic_book_series.series_id = comic_books.series_id
        JOIN orders ON comic_books.comic_id = orders.comic_id
    GROUP BY
        writers.writer_name;
END;
```

Figure 18: PL/SQL query 4

The above code returns a result set containing the number of comics written by each writer in the database. The procedure joins the writers, comic_book_series, comic_books, and orders tables, groups the result by the writer's name, and counts the number of comics for each writer.

Output:

```
mysql> CALL get_comic_count_by_writer();
+--------------------+-------------+
| writer_name        | comic_count |
+--------------------+-------------+
| Jason Aaron        |          26 |
| Gerry Duggan       |          12 |
| Ta-Nehisi Coates   |           7 |
| Murewa Ayodele     |           4 |
| Jeff Lemire        |           3 |
| Benjamin Percy     |           4 |
| Donny Cates        |           5 |
| Kieron Gillen      |           5 |
| Ryan North         |           6 |
| Christopher Priest |           1 |
| Mark Waid          |           1 |
| Joshua Williamson  |           4 |
| Scott Snyder       |           3 |
| Warren Ellis       |           7 |
| Tim Sheridan       |           1 |
+--------------------+-------------+
15 rows in set (0.01 sec)

Query OK, 0 rows affected (0.01 sec)
```

Figure 19: PL/SQL output 4

EXAMPLE QUERIES: YOUR DATABASE IN ACTION

➢ Get the top 10 best-selling comic books in terms of revenue generated

Query:

```sql
SELECT
    comic_books.comic_name, comic_books.price,
    COUNT(*) AS total_sales,
    SUM(comic_books.price) AS total_revenue
FROM comic_books
    JOIN orders ON comic_books.comic_id = orders.comic_id
GROUP BY comic_books.comic_id
ORDER BY total_revenue DESC
LIMIT 10;
```

Figure 20: Sample query 1

This SQL query joins the comic_books and orders tables, group the data by comic book name and calculate the total number of sales and revenue for each book

Output:

```
    -> LIMIT 10;
+--------------------+-------+-------------+--------------------+
| comic_name         | price | total_sales | total_revenue      |
+--------------------+-------+-------------+--------------------+
| Avenger #1         |  4.99 |          12 |  59.87999725341797 |
| I Am Iron Man #1   |  6.99 |           3 | 20.969999313354492 |
| Avenger #2         |  3.99 |           4 | 15.960000038146973 |
| Avenger #3         |  4.99 |           3 | 14.969999313354492 |
| Avenger #4         |  4.99 |           3 | 14.969999313354492 |
| Avenger #5         |  4.99 |           2 |  9.979999542236328 |
| Captain America #4 |  4.99 |           2 |  9.979999542236328 |
| Infinity Wars #3   |  4.99 |           2 |  9.979999542236328 |
| Captain America #5 |  4.99 |           2 |  9.979999542236328 |
| Infinity Wars #5   |  4.49 |           2 |  8.979999542236328 |
+--------------------+-------+-------------+--------------------+
10 rows in set (0.00 sec)
```

Figure 21: Sample output 1

➢ Get the total revenue generated from comic book sales by each month by year

Query:

```
SELECT
    DATE_FORMAT(orders.order_date, '%Y-%m') AS month,
    FORMAT(SUM(comic_books.price), 2) AS total_revenue
FROM comic_books
    JOIN orders ON comic_books.comic_id = orders.comic_id
GROUP BY DATE_FORMAT(orders.order_date, '%Y-%m')
ORDER BY month;
```

Figure 22: Sample query 2

This SQL statement retrieves the total revenue generated from sales of comic books for each month. It joins the comic_books and orders_tables groups the results by the month and calculate the sum of the price of comic books

Output:

month	total_revenue
2016-05	39.92
2017-07	1.99
2018-02	3.99
2018-04	3.99
2018-05	3.99
2018-08	13.47
2018-09	13.97
2018-10	8.48
2019-01	3.49
2019-02	4.99
2019-03	8.98
2019-05	4.99
2019-06	4.99
2020-07	4.99
2020-10	3.99
2020-11	6.98
2020-12	13.97
2021-01	37.92
2021-02	5.49
2021-03	8.98
2021-04	13.47
2021-05	5.49
2021-10	3.49
2022-01	1.99
2022-04	13.47
2022-05	8.48
2022-09	5.49
2022-10	50.89
2022-12	19.96
2023-01	6.98
2023-02	7.96
2023-03	21.96
2023-04	10.98
2023-05	30.94

34 rows in set (0.01 sec)

Figure 23: Sample output 2

➢ Get list of comic not purchased by customer

Query:

```sql
-- List of comic not purchased by customer
SELECT comic_books.comic_id, comic_books.comic_name
FROM comic_books
LEFT JOIN orders ON comic_books.comic_id = orders.comic_id
WHERE orders.order_id IS NULL;
```

Figure 24: Sample query 3

The above SQL query, returns comic which are not purchased by customers. Query selects the comic_id, comic_name from the comic books table. It uses LEFT JOIN to include all records from the comic books table and only those records from orders table that match the comic_id.
The WHERE clause filter out the records where order_id value is NULL indicating list of not purchased comic books

Output:

comic_id	comic_name
8	Avenger #8
9	Avenger #9
10	Avenger #10
24	I Am Iron Man #3
25	Thanos #1
29	Thanos #5
30	Thanos #6
31	Thanos #7
32	Ghost Rider #1
34	Ghost Rider #3
37	Ghost Rider #6
38	Hulk #1
39	Hulk #2
40	Hulk #3
41	Hulk #4
42	Hulk #5
43	Hulk #6
44	Hulk #7
53	Eternals #4
54	Eternals #5
56	Eternals #7

Figure 25: Sample output 3

➢ Get Customers who purchased whole comic book series

Query:

```
SELECT DISTINCT
    customers.customer_id,
    CONCAT(customers.first_name, ' ', customers.last_name) AS customer_name
FROM
    customers
    JOIN orders ON customers.customer_id = orders.customer_id
    JOIN comic_books ON orders.comic_id = comic_books.comic_id
    JOIN comic_book_series ON comic_books.series_id = comic_book_series.series_id
WHERE
    comic_book_series.series_name = 'Infinity Wars'
    AND customers.customer_id NOT IN (
        SELECT DISTINCT
            customers.customer_id
        FROM
```

Figure 26: Sample query 4.1

```
        FROM
            customers
            JOIN orders ON customers.customer_id = orders.customer_id
            JOIN comic_books ON orders.comic_id = comic_books.comic_id
            JOIN comic_book_series ON comic_books.series_id = comic_book_series.series_id
        WHERE
            comic_book_series.series_name = 'Infinity Wars'
        GROUP BY
            customers.customer_id
        HAVING
            COUNT(DISTINCT comic_books.comic_id) < (SELECT COUNT(*) FROM
            comic_books WHERE series_id = comic_book_series.series_id)
);
```

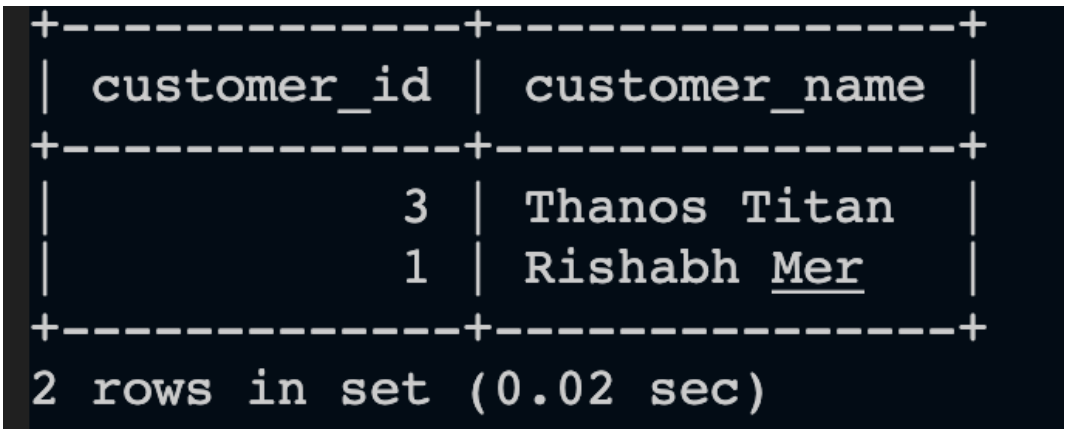
Figure 27: Sample query 4.2

This SQL query selects all customers who have purchased at least one comic book from the series Infinity Wars but have not purchased all the comic books from that series.

First we join the customer table with order table and comic table based on cusotmer_id and comic_id. Also joins the comic books table with the comic book series based on series_id.

Next, we filters the results to only get customers who have purchased at least one comic book from Inifinity wars series. Using WHERE clause that check if a customer has bought less than total number of comic books in the series.

Finally return the customer name.



```
+-------------+---------------+
| customer_id | customer_name |
+-------------+---------------+
|           3 | Thanos Titan  |
|           1 | Rishabh Mer   |
+-------------+---------------+
2 rows in set (0.02 sec)
```

Figure 28: Sample output 4

CONCLUSIONS

In conclusion, the Comic Store Database project has successfully addressed the requirements. The database design has a normalised structure that reduces the redundancy and ensures data consistency. The implementation of views and procedure queries enable efficient data retrieval and analysis, providing valuable insights on sales performance and customer behaviour.

This project serves as a solid foundation for building upon in the future. For example, adding additional tables like maintaining data of suppliers, pencilers, rating of comic books, Mint editions. Also, the database could be integrated with an e-commerce platform to enable online sales and a convenient customer experience.

Acknowledgements

I received guidance and feedback from my professor and classmates, but all the work presented in this report is my own. I have properly cited any referenced or quoted material to avoid plagiarism.

References

[1]. *https://en.wikipedia.org/wiki/Comics*
[2]. *https://www.javatpoint.com/uml-class-diagram*
[3]. *https://www.guru99.com/er-diagram-tutorial-dbms.html*
[4]. *https://www.guru99.com/database-normalization.html*
[5]. *https://www.geeksforgeeks.org/sql-views/*
[6]. *https://www.geeksforgeeks.org/plsql-introduction/*
[7]. *Alexander H. Harding. Comic Book Metadata and Database Design.*