Comic-Store--EXCELSIOR

REPORT

-VIJAYDITYA SARKER

TABLE OF CONTENTS

LIST OF FIGURES

INTRODUCTION

The comic database project aims to create a comprehensive database that catalogs information about comic books and graphic novels for an Excelsior company. The domain of the project encompasses the world of comic stores, like various genres, publishers, creators, characters, customers inventory, and sales.

The intended application for this database is to provide a centralized resource for comic book store Excelsior to run and manage their store efficiently and for their customers to have a hassle-free experience.

The nature and scale of the data are vast and diverse, as the project covers comics from different years, countries, and publishers. The data includes metadata such as title, author, publisher, publication date, issue number, sales price sale date, quantity and quality of comics, and other relevant information. Additionally, the project also includes customer data.

Excelsior will be able to operate and maintain their company smoothly due to my design for the comic store database. The well-structured database will allow the store to handle day-to-day duties like inventory tracking, customer data management, and financial performance monitoring more efficiently. The store will have easy accessibility to data such as available comic stock, client purchase tendencies, the quantity of stock, and profit with this system in place. Excelsior will be able to make more informed decisions, enhance its operations, and eventually boost profitability as a result of this.

2. DATABASE PLAN: A SCHEMATIC VIEW

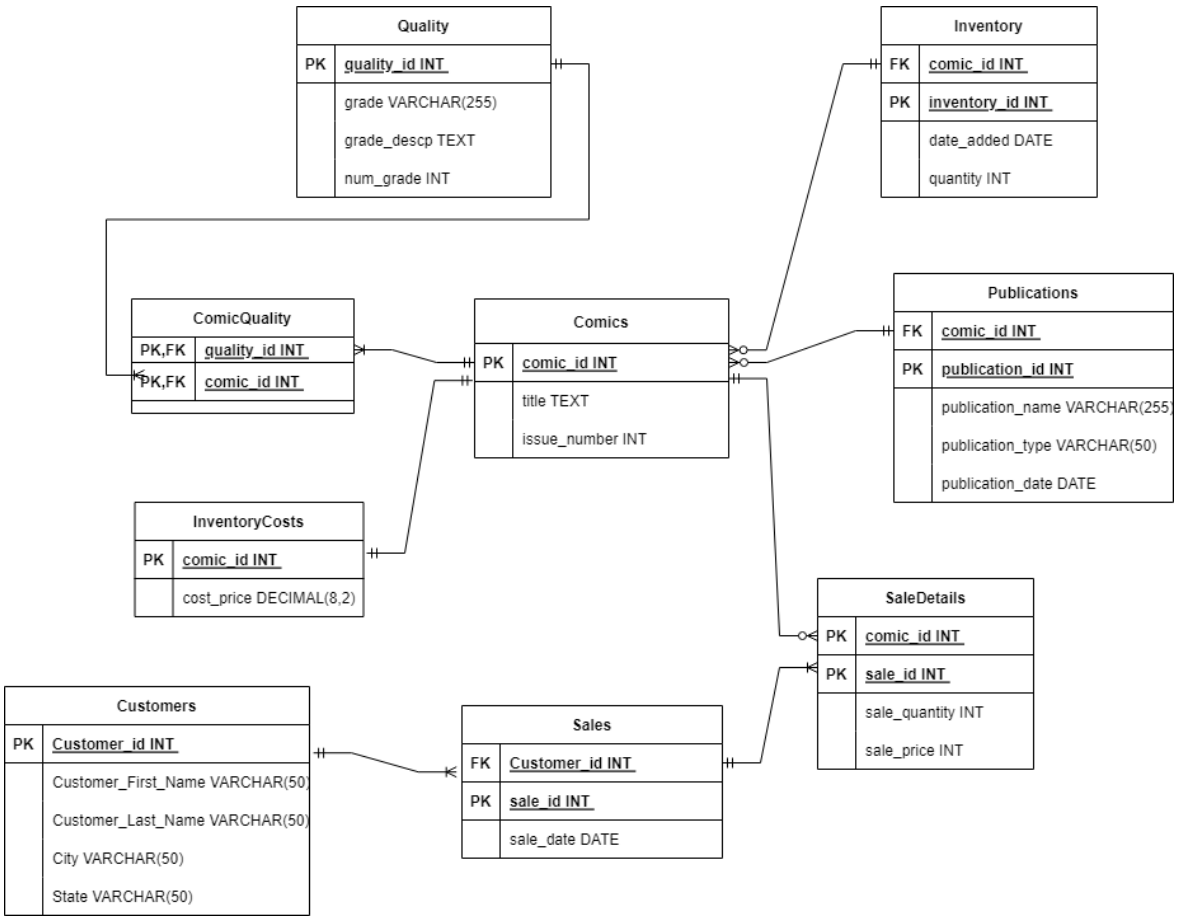

Figure 1: ER diagram

Excelsior is a structured database system that uses 9 interconnected tables to streamline comic business operations. The Entity-Relationship diagram emphasizes each table's/entity's distinct role in the system.

The principal entities in this database are:

1. **Comics**: The heart of the entire database contains information about the comics, such as the title which refers to the name of the comic, issue number states which part of the series the comic belongs to. Here comic_id is a primary key and has been used to make various connections with different entities.

2. **Quality**: This entity is in charge of determining the condition or quality of a comic book. It has properties such as grade, which reflects the state of the comic book, as well as accompanying descriptions and numerical grades, and its primary key is quality_id.

3. **ComicQuality**: The ComicQuality table connects the Comics and Quality tables by connecting each comic to its state. Because each order may contain many comics in varying conditions, there is a many-to-many relationship between the Orders table and the Comics and Quality tables..

4. **Inventory**: The Inventory table keeps track of every comic's inventory by tracking the date inserted and the quantity. This table is linked to the Comics table because numerous copies of each comic can be put in the inventory.

5. **InventoryCosts**: The InventoryCosts table keeps track of the cost of each comic in stock. Because every comic might have numerous prices, this table has a one-to-many link with the Comics table.

6. **Customers**: Customers' information, such as first and last names, city, and state, are stored in the Customers table. Because one client can place many orders, this table has a one-to-many link with the Orders table.

7. **Sales**: The Customer_id is a foreign key referring to the Customers table in order to identify the customer that made the purchase, and the sale_date specifies the date when the sale had taken place. The Sales table stores data regarding the sales made to customers, which includes the sale_id, which is a unique identifier for each sale.

8. **SaleDetails**: The SaleDetails table includes details on specific comic book sales, such as the sale_id that refers to a particular sale in the Sales table, the comic_id that refers to a particular comic book in the Comics database, the number of comics sold in the sale, and the price per comic.

   The SaleDetails table establishes a parent-child relationship by using foreign keys to refer to the primary keys of the Sales and Comics tables. Each row in the SaleDetails table is uniquely identified by its primary key, which is made up of the two columns

sale_id and comic_id. Because of this connection, it is possible to track individual sales in more detail within the context of overall sales.

9. **Publications**: Information about a comic book's many publications is kept in the publishing table. The Publications table contains data such as the publication name, kind, date, and ID of the comic to which the publication belongs in each row. Each row corresponds to a single issue or volume of a comic book publication.

   It enables one-to-many connectivity between the two tables by connecting the comic_id column in the Publications table with the comic_id field in the Comics table. This implies that a single comic can appear in more than one magazine and that each publication can only be linked to a single comic. In general, the Publications table offers both a mechanism to keep track of the various issues or editions of a comic book series and additional details about each publication.

3. DATABASE STRUCTURE: A NORMALIZED VIEW

Comics:

Each row in the "Comics" table corresponds to a different comic book that Excelsior's inventory consists of.
comic_id: Every comic in the table is uniquely identified by its "comic_id" property, which serves as the table's primary key. Since the integer data type is quick and effective in searching and indexing data, primary keys frequently employ it.

Title: The comic's title is kept in the non-null text field designated as the "title" property.
issue_number: The comic's issue number is kept in the non-null integer field "issue_number" of the property. These two characteristics help identify each comic and set it apart from the others in the table.

The "Comics" table, taken as a whole, is crucial to Excelsior's database of comic book shops. Each comic book in the collection has information on its title, and issue number stored in great detail. Making wise company decisions and effectively managing inventory and sales depend on this information. In order to improve its business operations, Excelsior may simply access and manage the information by organizing it into a well-structured table.

Given that there are no repeating groups of columns and that each column has a single value, this table satisfies the criteria for 1NF (first normal form). As all other properties are completely dependent on the primary key (comic_id), it also satisfies the criteria for 2NF (second normal form). Finally, since there are no transitive relationships between non-primary key characteristics, it also satisfies the criteria for the third normal form (3NF).

The Boyce-Codd Normal Form (BCNF) is satisfied by the table. The BCNF method of normalization is more effective than 3NF and guarantees that all functional dependencies in the table are trivial (i.e., involve only the primary key).

The comic_id serves as the table's primary key, so there aren't any non-trivial functional dependencies. The table thus meets BCNF requirements.

Quality:

quality_id: Each quality grade in the table is uniquely identified by its "quality_id" property, which is an auto-incremented integer primary key. A well-structured database must have a primary key, which guarantees that each entry in the table is distinct and can be quickly referred to by other tables. The database management system automatically allocates a distinct value to this attribute every time a fresh entry is added to the table by making "quality_id" an auto-incremented integer.

Grade: The name for the quality grade is kept in a non-null varchar field for this property. A straightforward string value called "grade" acts as the standard name for every quality grade. The database management system makes sure that each entry in the table has a value for this attribute by making "grade" a non-null column.

grade_descp: This attribute's non-null text field contains a description of the circumstance behind the quality grade. Longer explanations of each quality grade are possible using the "grade_descp" feature, giving more specific information about the condition corresponding to each grade. Since "grade_descp" and "grade" are non-null fields, each entry in the table will have a value for this property.

num_grade: The quality grade's numerical value is stored in this attribute's non-null integer field. Every quality grade is assigned a numerical value via the "num_grade" feature, which can be used to sort or filter information. The database management system makes sure that

every entry in the table has a value for this attribute by making "num_grade" a non-null field.

Since all characteristics are atomic and there are no transitive dependencies, the Quality table complies with 3NF. Each attribute stands for a discrete unit of data that cannot be subdivided further. The primary key, or quality_id attribute, is what distinctly distinguishes each entry in the table. Since there are no repeated groups, each row contains a distinct set of values.

As there are no non-trivial functional relationships between the attributes, the Quality table is also in BCNF. There are no duplicated attributes because all attributes are totally dependent on the primary key. As a result, the table is well-organized and designed to facilitate the storage and retrieval of data.

ComicQuality:

ComicQuality was firstly a part of the quality table but if was not satisfying BCNF, also there is a many-to-many relationship between the "Comics" and "Quality" tables created by means of the "ComicQuality" table. It includes the following attributes:

comic_id: a non-null integer field that points to the comic book's unique ID in the "Comics" table.

quality_id: a non-null integer field that points to the quality grade's distinctive ID in the "Quality" table.

This table's composite primary key is made up of the "comic_id" and "quality_id" values. As a result, each quality grade can be assigned to several comics and each comic may have multiple quality grades assigned to it.

Using foreign keys, the "ComicQuality" table serves as a junction table that joins the "Comics" and "Quality" tables. It permits comics and quality grades to have a many-to-many relationship.

The ComicQuality table is in third normal form (3NF) in terms of normalization because all attributes are atomic and there are no transitive relationships. Both qualities make comprise the primary key, and there are no repeating groups.

Because there are no non-trivial functional relationships between the characteristics and the table solely acts as a linking table between the Comics and Quality tables, it also exists in Boyce-Codd Normal Form (BCNF).

Inventory:

inventory_id: is a unique identifier for each inventory item in the table and is an auto-incremented integer primary key. It guarantees that every item in the inventory is separately accessible and has a unique identification.

comic_id: is a non-null integer column that points to the comic book's special ID in the "Comics" table. This field serves as a link between the inventory item and the comic to which it belongs. Only legitimate comic IDs can be entered into the table thanks to the foreign key constraint on this field.

date_added: contains the date that the inventory item was added to the inventory and is a non-null date field. This feature can be used for analysis and reporting as well as keeping track of when a comic book was added to the inventory.

quantity: is a non-null integer field that holds the number of comics that are currently in stock. This field displays the quantity of a specific comic book that is currently in stock. It can be altered as new comic books are added to or taken out of stock.

The Inventory table is in third normal form (3NF) in terms of normalization because all attributes are atomic and there are no transitive dependencies. There are no repeated groups and the inventory_id attribute serves as the primary key.

Table of inventory is following BCNF. There are no transitive relationships or non-trivial functional dependencies between the characteristics.

InventoryCosts:

This table was also a part of the Inventory table but I had to split it form the inventory table as it was not satisfying 3NF and BCNF which would have led to more data redundancy, Based

on the comic_id attribute, that acts as a foreign key in both tables, the InventoryCosts table and the Inventory table have a one-to-one link.

Data on the cost price for every comic book in the inventory is kept in the InventoryCosts database.

comic_id: The integer data type "comic_id" found in the InventoryCosts table contains the unique identifier for each comic book in the inventory. The attribute acts as the table's main key, meaning it uniquely identifies every row of data.

cost_price: The cost price of the relevant comic book is represented by the decimal data type "cost_price" attribute. When a comic book is self-published, this feature keeps track of the cost of generating it as well as the cost of buying it from the publisher. The cost price is kept with the required precision as well as scale, in this case, up to two decimal places, thanks to the usage of the decimal data type.

Publications:

The Publications table's function is to keep track of the publications connected to each comic book in the inventory. Each entry in the table represents a comic book publication and gives details about that publication, including its name, kind, and date of release.

publication_id: an integer that acts as the table's primary key which is automatically increased with each subsequent publication. It helps to identify each publication in the table in a special way. Publication_id's data type is INT.

publication_name: a string containing the publication's name. The publication_name attribute, which is non-null and has the data type VARCHAR, holds the name of the publication.

publication_type: a string that identifies the publication type. The publishing type is stored in this non-null attribute. VARCHAR is the publication_type.

comic_id: The number that represents the comic book's ID, with which the publication is linked. As a foreign key to the Comics table, it has a non-null attribute. comic_id's data type as INT.

publication_date: a date that signifies the publication date. It contains a non-null value that indicates the date the publication was made available. The value of publication_date represents a date.

Since the primary key (publication_id) is the only non-key attribute that the table is dependent upon and there are no transitive dependencies between non-key attributes, the table conforms to the third normal form (3NF) of database normalization. To ensure that only legitimate comic book IDs are used in the table, the table additionally has a foreign key constraint that references the Comics table.

AS there are no non-trivial functional connections between the characteristics, the Publications table appears to follow BCNF. Each row in the table is uniquely identified by its primary key, publication_id, while my comic_id column acts as a foreign key to the Comics table. There aren't any composite or redundant candidate keys that can cause abnormalities.

Customers:

This table's function is to keep track of customer information such as names and addresses. **Customer_id:** Every customer's "Customer_id" attribute, which has an integer data type, acts as a unique identifier. In order to guarantee that it is always filled in, it is set to auto-increment for each new client and is designated with the NOT NULL constraint. It has many to one relationship with the sales table.

Every customer's first and last name are represented via the "**Customer_First_Name**" and "**Customer_Last_Name**" properties, respectively. They are indicated with the constraint NOT NULL to guarantee that name fields are constantly filled in and are of the variable character data type (VARCHAR) with a maximum length of 50 characters.

City: stores the city name, which is of the VARCHAR data type and have a maximum length of 50 characters, indicating where the customer resides.

State: stores the state name, which is of the VARCHAR data type and has a maximum length of 50 characters, indicating where the customer resides.

The "Customers" table is in 1NF since each attribute has an atomic value. Due to the fact that there is only one candidate key (Customer_id) and all non-key attributes depend on the full key, it is likewise in 2NF. Due to the lack of transitive dependencies between non-key characteristics, it is also in 3NF. As a result, BCNF contains the "Customers" table.

Sales:

Each sale made to a consumer is recorded on the table, it has one-to-many relationships that allow for many sales from a single customer.

The "Sales" table's function is to keep track of client sales data. The "Customer_id" foreign key connects each sale to a customer in the "Customers" table.

sale_id: is a special identifier for each sale, and it bears the constraint NOT NULL, which prevents its null value. When a new row is added to the table, MySQL can automatically create a new, distinct value for this attribute thanks to the AUTO_INCREMENT property.

Customer_id: the "Customer_id" attribute of the "Customers" table is referenced by this foreign key. Each purchase is connected to a single consumer using this property. Each sale must be connected to a legitimate customer, which is ensured by the NOT NULL condition.

sale_date: is the day the sale was completed. Every sale must have a valid date associated with it since this attribute, which is of type DATE and has the constraint NOT NULL, is present.

The "Sales" table is in 1NF normalization since each attribute has an atomic value. Due to the fact that there is only one candidate key (sale_id) and all non-key attributes depend on the complete key, it is also in 2NF. Due to the lack of transitive dependencies between non-key characteristics, it is also in 3NF. As a result, BCNF contains the "Sales" table.

SaleDetails:

The sale and SaleDetail table was the same table it was split as it had many too many relationships with the comics table and as it was not satisfying BCNF normalization.
The "SaleDetails" table's function is to keep track of details about each sale, including the comics which were purchased, how many were purchased, and their individual prices.

Two foreign key constraints are present in the "SaleDetails" database, one of which refers to the "sale_id" column found in the "Sales" table while the other to the "comic_id" column of the "Comics" table. According to both the "Sales" and "Comics" tables, respectively, each row in the "SaleDetails" table corresponds to a particular sale and comic book. As a result, there is a many-to-many relationship between the "Sales" table and its "Comics" table and the "SaleDetails" table.

sale_id: The ID of the sale is represented by this attribute, which has an integer data type. Given that it bears the NOT NULL requirement, it cannot be left empty. This attribute is related to the "Sales" database and requires that its value be present in the "sale_id" column of that table due to the FOREIGN KEY requirement that refers the "sale_id" column of the "Sales" table.

comic_id: The ID of the comic book is represented by this attribute, which has an integer data type. Given that it bears the NOT NULL requirement, it cannot be left blank. This attribute is related to the "Comics" database and requires that its value be present in the "comic_id" column of that table due to the FOREIGN KEY constraint that refers the "comic_id" column found in the "Comics" table.

sale_quantity: The number of comic books sold during the sale is shown by this attribute, which has an integer data type. Given that it bears the NOT NULL requirement, it cannot be left empty. A number that has a positive value larger than zero is kept in this attribute.

sale_price: The price of each comic book sold during the sale is represented by this characteristic, which has a decimal data type. Given that it bears the NOT NULL requirement, it can't be left empty. A positive decimal number larger than zero is kept in this attribute.

The "SaleDetails" table is in 1NF since each property has an atomic value. Additionally, it belongs to 2NF because all non-key attributes are dependent on the complete key and have a composite candidate key (sale_id, comic_id). Due to the lack of transitive dependencies between non-key characteristics, it is also in 3NF. As a result, BCNF contains the "SaleDetails" table.

4. DATABASE VIEWS

1)ProfitAndLoss VIEW

A summary of the total sales volume, total sales price, total cost, and total profit for each sale_id and comic title combination is provided by the SQL view ProfitAndLoss, which joins the databases SaleDetails, Comics, and InventoryCosts.

By displaying the total cost and income earned for each comic title in a transaction, the view aims to give business insights into the profitability of each sale. Business analysts, accountants, and financial decision-makers who must keep track of the financial success of comic book sales and inventory are given this information.

Without needing users to have direct access to the underlying tables, summarizing and analyzing The profit and loss during sales of a comic was accomplished by representing a specific relationship as a SQL view. Additionally, it made reporting simpler and lessen the complexity of the SQL queries required to produce the desired results.

The ProfitAndLoss view can be used for a variety of purposes, such as-

1. creating monthly statistics on the profitability of comic book sales,
2. identifying high-performing comic book series
3. assessing how price and inventory adjustments affect overall profitability.

sale_id	title	total_sales_quantity	total_sales_price	total_cost	total_profit
1	The Amazing Spider-Man	2	10	11.98	8.02
1	Batman: The Killing Joke	1	15	8.99	6.01
2	Watchmen	1	20	12.99	7.01
2	Saga	3	7	29.97	-8.97
3	The Walking Dead	2	25	29.98	20.02
3	X-Men	4	10	27.96	12.04
4	Deadpool	1	15	10.99	4.01
4	Captain America	2	20	27.98	12.02
5	The Sandman	3	12	23.97	12.03
5	Spawn	1	16	11.99	4.01

Figure 2: Profit and Loss

2) **ComicSales VIEW**

The scene The Comics, Publications, and SaleDetails tables are joined to create ComicSales, which only includes the pertinent columns (the comic's title, the issue's publication date, and the sale price). This view offers a quick method to get data about each comic's sales, which includes the comic's publication date and selling price.

Anyone in require of this data, such as employees or managers of a comic book shop who might be curious to know which comics are selling well and how much money they are bringing in, should be able to access it using the ComicSales view. They may quickly and simply obtain this data using this view without needing to create challenging SQL queries.

The ComicSales view can be used, for.-

1)Instance, to create sales reports for a certain time frame

2)To examine the sales patterns of a specific comic.

3)In order to help store managers make wise choices about inventory and promotions, it might also be used to determine which comics are selling well and which are not.

title	publication_date	sale_price
The Amazing Spider-Man	1962-03-10	10
Batman: The Killing Joke	1939-05-01	15
Watchmen	1963-09-01	20
Saga	1942-12-01	7
The Walking Dead	1962-05-01	25
X-Men	1963-03-01	10
Deadpool	1956-10-01	15
Captain America	1940-07-01	20
The Sandman	1964-04-10	12
Spawn	1941-03-10	16

Figure 3: Comic sales

3) **SalesByPublicationType**

The scene SalesByPublicationType combines sales information according to the publishing type (such as a comic book, graphic novel, trade paperback, etc.). It computes the overall volume of every publication type sold as well as the overall revenue from those sales.

In order to analyze sales trends and determine which publications are the most well-liked and lucrative, this view presents a summary of sales data by publication type. Publishers, distributors, and retailers in the comic book industry can use this data to make wise financial decisions.

It is prudent and required to represent this relation as a SQL view because it makes it simple to obtain sales data that has been compiled by publishing type without always having to create difficult SQL queries. This speeds up and streamlines the creation of sales reports.

This view can be used for a variety of purposes, such as

1)Examining sales data over a given time period.

2)Contrasting sales data from various publication kinds.

3)Determining which publications are the most lucrative.

publication_type	total_quantity	total_revenue
Comic Book	16	232
graphic novels	4	41

Figure 4: Sales by Publication type

4) **ComicInfo View**

The scene In order to pick the comic title, quality grade, and quantity for all comics with a quality grade higher than 8, ComicInfo is connecting three tables (Comics, Quality, and Inventory). Employees in charge of overseeing and reselling the available inventory of superior comics can use this view, which offers information about it.

This View will also tell us the quality of our inventory, if the low quality of comics we need to get better quality comics.

Examples-

1)The inventory management team can utilize a report of all the high-quality comics in stock, together with their amounts, as an example of how to use the ComicInfo view to help them decide what to buy and how much to restock.

2)Another example would be to filter the view by a particular comic book title and provide the quality level and available stock, which the sales team might utilize to respond to client questions.

title	grade	quantity
The Amazing Spider-Man	Mint	50
Batman: The Killing Joke	Near Mint	25

Figure 5: Comic info

5. PROCEDURAL ELEMENTS

1) **get_comic_inventory (stored procedure)-**

For obtaining details about the inventory of a certain comic book, the stored procedure "get_comic_inventory" is a helpful resource. The only input required by the procedure is comic_id, an integer that represents a comic book's ID.

The first step of the process is to declare the "quantity" variable, which will be used to hold the inventory quantity of the comic book. The quantity is then chosen and assigned to the "quantity" variable for the specified comic_id from the Inventory database.

The function then uses the quantity variable along with the comic_id input to choose the comic book's title via the Comics table. Additionally, the technique adds a new column named "message" that, depending on the number, will either display "Stock present" or "No stock." The message will say "Stock present" if the quantity is more than zero and "No stock" otherwise.

The saved process can be used to quickly access inventory data for a specific comic book. Users can quickly verify the inventory status of a given comic book by supplying the comic_id as an input parameter without having to manually browse the inventory or comic book tables. Overall, the Excelsior database's

"get_comic_inventory" stored procedure is a helpful tool that makes it easier to retrieve inventory data for a certain comic book.

2) **search_comics_by_title(stored procedure)-**

Users can easily search for comic books in the database by their titles using the stored method search_comics_by_title. The procedure can look for any title in the Comics table that contains the search string by accepting the search string as input and utilizing the LIKE operator. This is advantageous since it lets consumers find comic books without needing to know the whole title in advance using a keyword or partial title.

The method can locate both partial matches and exact matches since it uses wildcard characters (%) to match any characters before or after the search string. This is especially helpful if there are numerous editions of the same comic book with slightly different titles or if the user is unsure of the exact title of the comic book they are looking for.

Users no longer need to memorize the precise syntax or SQL query needed to do the search by encapsulating this search capability in a stored procedure.

The search_comics_by_title stored procedure is a helpful resource for anyone who needs to locate particular comic books in the database since it offers users a quick and easy way to search for comic books in your database by their titles.

3) **prevent_deletion_if_inventory(Triggers)-**

Prevent_deletion_if_inventory is the name of the trigger mentioned above. It is set up to run before any row in the Comics table is destroyed. The trigger makes sure that deleting a comic that has inventory in the Inventory table will stop the operation.

The trigger initially determines if the comic being erased has any inventory. Utilizing a SELECT statement, it searches the Inventory database for any records where the comic_id matches the OLD. Comic_id (i.e., the comic being deleted) and the amount is more than zero (i.e., there is inventory present) in order to determine whether

there is any inventory. If such records are present, the trigger issues a SIGNAL statement error with the SQLSTATE "45000" and the custom error message "Cannot delete comic with inventory present" if such records are present. This stops the comic from being deleted and gives the user an error notice.

In conclusion, this trigger serves as a safety measure to prevent the unintentional deletion of comics that have inventory. It prevents the deletion of crucial database data, ensuring data consistency and integrity.

4) add_to_inventory(Triggers)

Maintaining data integrity between the Comics and Inventory databases is made possible by the add_to_inventory trigger. This trigger will automatically create a new record in the Inventory database whenever a new comic is added to the Comics table. This record will have the proper comic_id, date_added, and a starting quantity of 1.

Without the need for additional effort from the user or administrator, this automation helps to ensure that the Inventory table stays current with the newest additions to the Comics table, which can save time and reduce the possibility of errors or inconsistencies that might occur if the process were done manually.

In general, the add_to_inventory trigger may assist to simplify the process of managing inventory records in a database and is useful for ensuring data accuracy between related tables.

6. EXAMPLE QUERIES: YOUR DATABASE IN ACTION

1) **Retrieve all comics from the Comics table.**

--→ SELECT * FROM Comics;

All rows and columns from the "Comics" database are returned with the query "SELECT * FROM Comics;". It offers details on every comic book in the database, such as its ID, title, and issue count. The application can make use of this data to provide the user a list of available comic books.

comic_id	title	issue_number
1	The Amazing Spider-Man	1
2	Batman: The Killing Joke	1
3	Watchmen	1
4	Saga	1
5	The Walking Dead	1
6	X-Men	1
7	Deadpool	1
8	Captain America	1
9	The Sandman	1
10	Spawn	1
12	The Amazing Spider-Man	3

Figure 6: Query 1

2) Retrieve all comic books and their associated qualities.

```
SELECT Comics.title, Quality.grade

FROM Comics

JOIN ComicQuality  ON Comics.comic_id = ComicQuality.comic_id

JOIN Quality  ON ComicQuality.quality_id = Quality.quality_id;
```

By combining the "Comics" table with the "ComicQuality" and "Quality" tables, this query obtains the title of each comic and its accompanying quality rating. Only the "grade" column from the "Quality" database and the "title" column from the "Comics" table are included in the result set. This query gives the app details on each comic's quality, which can be used for filtering or sorting by quality grade, among other things.

title	grade
The Amazing Spider-Man	Mint
Watchmen	Mint
The Walking Dead	Mint
The Amazing Spider-Man	Near Mint
Batman: The Killing Joke	Near Mint
The Walking Dead	Near Mint
Watchmen	Very Fine
The Walking Dead	Very Fine
Saga	Fine
The Walking Dead	Fine

Figure 7: Query 2

3) Retrieve the total quantity of inventory for each comic book.

```
SELECT Comics.title, SUM(Inventory.quantity) as total_quantity

FROM Comics

JOIN Inventory  ON Comics.comic_id = Inventory.comic_id

GROUP BY Comics.title;
```

By combining the "Comics" table alongside the "ComicQuality" and "Quality" tables, this query obtains the title of each comic and its accompanying quality rating. Only the "grade" column from the "Quality" database and the "title" column from the "Comics" table are included in the resulting data set. This query gives the app details on each comic's quality, which can be used for filtering or ordering by quality grade, among other things.

title	total_quantity
The Amazing Spider-Man	51
Batman: The Killing Joke	25
Watchmen	100
Saga	75
The Walking Dead	10
X-Men	5
Deadpool	30

Figure 8: Query 3

4)Retrieve the top 3 best-selling comics based on sales quantity.

```
SELECT Comics.title, SUM(SaleDetails.sale_quantity) as total_sales

FROM Comics

JOIN SaleDetails  ON Comics.comic_id = SaleDetails.comic_id

GROUP BY Comics.title

ORDER BY total_sales DESC

LIMIT 3;
```

This query will retrieve the top 3 comics based on their total sales quantity. It joins the "Comics" table with the "SaleDetails" table on the "comic_id" attribute, and then groups the results by comic title. The "SUM" function is used to calculate the total sale quantity for each comic. Finally, the results are sorted in descending order by the total sales and limited to the top 3 records. This query could be useful for the application to display the most popular comics in the inventory.

title	total_sales
X-Men	4
Saga	3
The Sandman	3

Figure 9: Query 4

5)Retrieve the total revenue from all sales for a 2022 september to 2023 March

```
SELECT SUM(SaleDetails.sale_quantity * SaleDetails.sale_price) as total_revenue

FROM SaleDetails

JOIN Sales  ON SaleDetails.sale_id = Sales.sale_id

WHERE Sales.sale_date BETWEEN '2022-09-01' AND '2023-03-31';
```

This query calculates the total revenue generated from all sales that occurred between September 1, 2022, and March 31, 2023. It does this by joining the SaleDetails table with the Sales table on the sale_id column, then filtering the results based on the sale_date column in the Sales table. It then multiplies the sale_quantity and sale_price columns in the

SaleDetails table to calculate the total revenue for each sale, and then sums up these values to give the total revenue generated in the given time period.

	total_revenue
▶	35

Figure 10: Query 5

6)Find the top 3 comics with the highest total sales amount.

```
SELECT Comics.comic_id, Comics.title, SUM(SaleDetails.sale_quantity *
SaleDetails.sale_price) AS Total_Sales_Amount

FROM Comics

JOIN SaleDetails  ON Comics.comic_id = SaleDetails.comic_id

GROUP BY Comics.comic_id

ORDER BY Total_Sales_Amount DESC

LIMIT 3;
```

The top three comics by total sales income will be returned by this query. To determine the amount of the overall sales for each comic according to the amount and price of sales, it combines the "Comics" table with the "SaleDetails" table. The top three results are then selected after the results are grouped by comic ID, sorted by total sales amount, and limited.

The application can learn a lot from this query about the best-performing comics in terms of income generation. With the use of this data, shop owners may decide which comics to stock more of or to advertise to customers more aggressively.

comic_id	title	Total_Sales_Amount
5	The Walking Dead	50
6	X-Men	40
8	Captain America	40

Figure 11: Query 6

7)Find the comics that have never been sold.

```
SELECT Comics.comic_id, Comics.title

FROM Comics

LEFT JOIN SaleDetails ON Comics.comic_id = SaleDetails.comic_id

WHERE SaleDetails.sale_id IS NULL;
```

This search could be helpful for managing inventory or determining which comics might benefit from promotions or discounts to boost sales.

comic_id	title
12	The Amazing Spider-Man

Figure 12:Query 7

7. CONCLUSIONS

The framework for more complex data analysis and reporting is set in place by a well-designed and normalized database structure. The capacity to add additional information and features, including tracking sales trends, inventory control, and consumer behaviour research, is made possible by the database's scalability. The Excelsior database may develop further and assist the expansion and success of the business with this strong foundation.

The database can offer a more complete solution to handling the business operations with additional tables, including a table for keeping track of inventory and supplier information. Furthermore, adopting a user login system with suitable access control can improve database security and safeguard critical data. These are just a few instances of how the database may be improved to better aid the business's foreseeable future expansion and development.

To further evaluate and draw conclusions from the data in the database, we may also take into account implementing data analytics and machine learning approaches. This could entail establishing data visualization tools to aid users in better comprehending and interpreting the data, as well as creating predictive models to forecast sales trends and improve inventory management. Overall, the database has enormous potential for growth and improvement, which can significantly improve Excelsior's operations and company expansion.

ACKNOWLEDGMENTS

I want to sincerely thank my teacher Tony Veale and the lab mentors for their invaluable advice and assistance with this project. My understanding of the challenges of database design and development has been greatly aided by their knowledge of the subject and commitment to educating.

I want to express my gratitude to my teacher for giving me the sources, equipment, and guidance I needed to do this project. Throughout this journey, I have been inspired by their support and enthusiasm. I also want to express my gratitude to my lab mentors for their constant support and direction. Their understanding, courtesy, and willingness to go above and above to help me have been priceless.

Last but not least, I'd want to thank everyone who helped with this project in whatever manner. Your assistance has been crucial to helping me accomplish my objectives and effectively finish this job.

REFERENCES

WEBSITES-

1. Carnes, B. (2021, January 22). SQL and Databases - Full Course. freeCodeCamp. Retrieved from https://www.freecodecamp.org/news/sql-and-databases-full-course/
2. Title: SQL Tutorial Website Name: W3Schools URL: https://www.w3schools.com/sql/ Accessed Date: May 9, 2023
3. Title: Mile High Comics - Tales From the Database Website Name: Mile High Comics URL: https://www.milehighcomics.com/tales/
4. Title: SQL and Databases - Full Course Website Name: freeCodeCamp URL: https://www.freecodecamp.org/news/sql-and-databases-full-course/ , Author: Beau Carnes Published Date: January 22, 2021

YOUTUBE-

1. DBMS - Entity Relationship Diagram, URL: https://www.youtube.com/watch?v=obb7SlUmKQE , Lecture By: Mr. Arnab Chakraborty, Tutorials Point

2. Learn Database Normalization - 1NF, 2NF, 3NF, 4NF, 5NF, URL: https://www.youtube.com/watch?v=GFQaEYEc8_8&t=13s , By Decomplexify

GitHub-

grand-comics-database
https://github.com/droduit/grand-comics-database